

CS 433 COMPUTER NETWORKS

---

**COMPREHENSIVE ANALYSIS OF WEB  
COOKIES, IDENTIFICATION OF  
VULNERABILITIES, COOKIE POISONING**

---

November 28, 2022

Dhairya Shah 20110054  
dhairya.shah@iitgn.ac.in  
Dhyey Thummar 20110059  
dhyeykumar.thummar@iitgn.ac.in  
Rahul Chembakasseril 20110158  
rahul.chembakasseril@iitgn.ac.in  
Sukruta Midigeshi 20110206  
sukruta.midigeshi@iitgn.ac.in  
Indian Institute of Technology, Gandhinagar

# 1 Abstract

On the one hand, cookies serve several crucial functions on the web, for example, storing stateful information on the user's device. While on the other hand, tracking cookies, also known as third-party cookies, are commonly used to compile long-term records of individuals's browsing histories. Thus, although cookies are an intrinsic element of web applications, their use has important implications for user privacy. Noticing this concern and experiencing the bugs of IITGN websites and some of the critical government websites, we have taken this project to research cookies, analyze them, and work on the security concerns of cookies.

# 2 Research and Review of Literature

To formulate our project proposal and get a clear understanding of the domain, we went through three research papers, out of which one focussed mainly on the samesite attribute of the cookies and was suggested by Prof. Kulkarni. The other two research papers focussed on third-party cookies and explained all the attributes of cookies. We also went through the MDN Web Docs and RFC documents on HTTP and Web cookies to get familiar with the standard terms used in this domain. We would like to mention a few learnings from the project here:

There are three purposes of cookies: Session Management (1st party cookies), Personalisation, and Tracking (3rd party cookies). 1st party cookies are placed by the domain displayed in the browser's address bar and are commonly used by e-commerce applications. 3rd party cookies are cross-site, i.e., placed by a domain different than the one shown in the browser's address bar, and are used for data analysis and monetization. We found from their results that 3rd party cookies dominate 1st party cookies in terms of placements. Nearly 80% of the cookies harvested by their crawlers were sent insecurely. The second research paper studies a web tracking method that collects 1st party cookies that are set by 3rd party Javascript code. These cookies are also called external cookies, which are used to bypass browser policies that try to block 3rd party cookies. They found that more than 50% of the top Alexa websites have 3rd party cookies that exchange tracking IDs stored in external cookies. According to them, existing mitigation techniques to block 3rd party cookies are inadequate to protect users.

We analyzed the attributes of web cookies like Domain, HttpOnly, Secure, Samesite, and many more. The research paper provided by sir made us dive deep into the Samesite attribute.

## 2.1 Samesite Policy Review

We analyzed the idea, implementation and effectiveness of the Samesite policy implemented by Chrome to restrict the scope of cookies to a same-site context thereby reducing cross-site attacks like CSRF(Cross-Site Resource Forgery) and CSIL(Cross-Site Information Leak). They did this by adding a new attribute to web-cookies called Samesite which could take one of three values - Strict, Lax, None.

If a user U has visited a site A.com and A.com installed cookies into U's browser, then if U now visits a malicious site B.com which makes hidden requests to A.com, if A.com has set its cookies's Samesite attribute

- to Strict, U's browser won't send them via B.com's requests.
- to None, U's browser will always send A's cookies via B.com's requests.
- to Lax, U's browser will send A's cookies to B.com only in certain contexts - when requests are made within the same site, or when they are cross-site top-level navigation requests (the URL bar information changes) but the HTTP methods used are safe-like GET methods and not unsafe methods like POST.

If B.com gets access to A.com's cookies, then B.com can impersonate user U now to conduct CSRF and CSIL attacks. Chrome made the Lax-by-default their new policy when no attribute is specified for Samesite.

The paper addresses four main topics-

- Trend Analysis of SameSite Cookie Usage Only about 19 percent of the top 500K websites adopted a new policy, rest resorted to using the default value. So, the explicit adoption was not made by close to 81 percent of the sites, meaning that they would be affected by the default policy.
- Functionality Breakage Since a large number of websites rely on cross-site requests to implement a lot of their functionality, a thorough review of the cookies, their purpose and their importance to implementing certain website features needs to be done on the developer's side. Many developers just resorted to the default policy, so part of their website's functionality broke. 19 percent of the affected cross-site requests made to the top 500K websites broke. Most of these were for advertisement networks and social media apps.
- Lax Adequacy and Threats The new Lax policy can be bypassed using the state-changing GET requests which change to POST and can be used to implement resource forgery. Chrome implements a Lax+POST exceptionally policy to enable the service of SSO Networks. This exception gives attackers a two-minute window to get access to cookies. If cookies's path values are different on two webpages in the website or if cookie attributes are different for different user agents, then the Samesite policy fails.

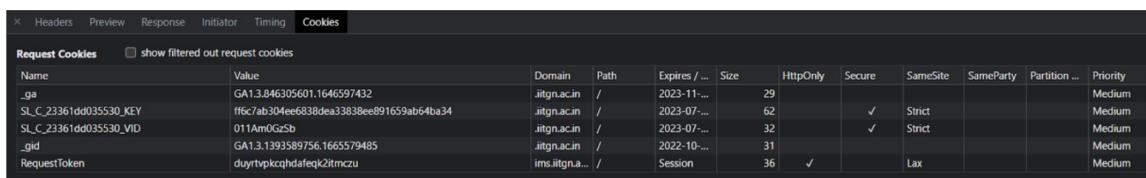
- Browser Inconsistencies and Web Frameworksâ Defaults Many popular browsers like Firefox and Safari impose a None policy and hence users are susceptible to XS attacks when they use these browsers even if the websites have set their cookie attributes correctly. There are too many moving parts in the system- users, developers, browsers, web frameworks, exceptions for SSO providers, etc. for the policy to be foolproof.

So, in conclusion, the large majority of websites are still susceptible to XS attacks even after the new Samesite policy, because it can be bypassed.

## 3 Exploring different tools and their Implementations

### 3.1 Analysing cookies manually

We first analyzed the cookies of some of the commonly used websites using the Chrome browser's Inspect panel (Ctrl + Shift + I). Inside the Application tab, we can see the cookies used by the website and some of their important attributes like Domain, Path, Expiry date, Size, Secure, et cetera. Given below is the information about the cookies used by the IMS website:



Name	Value	Domain	Path	Expires / ...	Size	HttpOnly	Secure	SameSite	SameParty	Partition ...	Priority
_ga	GA1.3.846305601.1646597432	.iitgn.ac.in	/	2023-11-...	29						Medium
SL_C_23361dd035530_KEY	ff6c7ab304ee6838dea33838ee891659ab64ba34	.iitgn.ac.in	/	2023-07-...	62		✓	Strict			Medium
SL_C_23361dd035530_VID	011AmOGzSb	.iitgn.ac.in	/	2023-07-...	32		✓	Strict			Medium
_gid	GA1.3.1393589756.1665579485	.iitgn.ac.in	/	2022-10-...	31						Medium
RequestToken	duyrtvpcqzhdafeqk2itmczu	ims.iitgn.a...	/	Session	36	✓		Lax			Medium

As analyzing cookies manually for all the IITGN websites is practically impossible, we tried to automate our task of getting the cookies for a particular URL. We researched a bit for this, and as many of us were more familiar with python than other languages and had experience with web- scraping, we decided to go with python.

### 3.2 Automating listing of cookies for a given website

Initially, we started by using the requests library of Python.

```
import requests

url = "https://www.google.com"

response = requests.get(url)
cookieJar = response.cookies
cookies = cookieJar.get_dict()
print(cookies)
```

This is a very crude implementation of using the requests library. This would print all the cookies for the given URL. After some processing, we could print the result as a table. Thus, we could get all the attributes of the cookie for the given URL, such as its domain, Path, Expires, HTTP-Only, Secure, and SameSite.

```
PS D:\Networks Project> python .\test.py
Enter URL: https://www.google.com/
Status: 200
```

Cookies for: https://www.google.com/								
Name	Value	Domain	Path	Expires	HttpOnly	Secure	SameSite	
1P_JAR	2022-11-27-16	.google.com	/	1672156814	False	False	False	
AEC	AakniGMr1ii8Ny-XkQY6BVvr3oViVEvonw20TMG1t3Ry4QGPIUp_ZHuaw	.google.com	/	1685116814	True	False	True	
NID	511=AptGig_TXlJmkT2w2Noy0z3DmIVB_56piGOQJjFBmCPywfIksiY9huEvkGhDyaeJ3DBGQuRpdY1Z5mpFiKDvjTX68Gv1n15TfojTfUKJ8mnX_fTnoJ9p4pRq3j5-jaZ1ohkvEwbvaa1hjwgtGBZKQsrUBf00uIBvalgIscwi4NVE	.google.com	/	1685376014	True	False	False	

However, we soon realized that this approach of using the requests library was not working on many of the sites, and we could not get all the cookies for some sites. The issue was that this approach could not simulate the session of a typical browser session, and thus many of those cookies required for the session were not being transferred and thus not being captured by the code. Based on this, we decided to think of another approach, and that is when we decided to use the Selenium library.

*Selenium is an open-source umbrella project for a range of tools and libraries aimed at supporting browser automation.*

Selenium has a WebDriver API in python used for controlling browsers and performing browser automation. Hence, we went ahead and implemented this in our code.

Now, we can get the cookies for almost all the URLs. We also added the support of reading all the URLs from a .txt file, getting all the data, and saving all the information to an Excel Sheet. The motive for doing this was to create a database for some set of sites: a few iitgn sites, some government websites, and some popular sites.

The script for this is the cookie-scraper.py. A glimpse of this database can be found here: [database](#)

```

from selenium import webdriver
from selenium.webdriver.chrome.service import Service

DRIVER_PATH = "chromedriver_win32/chromedriver.exe"
s = Service(DRIVER_PATH)
options = webdriver.ChromeOptions()
options.binary_location = r"C:/Program Files/BraveSoftware/Brave-Browser/Application/brave.exe"
# options.add_argument(
#     "--user-data-dir=C:\\Users\\dhyey\\AppData\\Local\\BraveSoftware\\Brave-Browser\\User Data"
# )
# options.add_argument("--profile-directory=Default")
options.add_argument("--log-level=3")
options.headless = True
driver = webdriver.Chrome(options=options, service=s)

def getAllParams(driver, urls):
    cookies = []
    for url in urls:
        print('Getting cookies for: ' + url)
        try:
            driver.get(url=url)
            cookies.append(driver.get_cookies())
            driver.implicitly_wait(3)
        except Exception as e:
            print(e)

```

url	name	value	domain	path	expires	httpOnly	secure	sameSite
<a href="https://www.linkedin.com/">https://www.linkedin.com/</a>	USCOOKIE	v=16z0ZZ11271	www.linkedin.co	/	1,063,119,130	TRUE	TRUE	None
	lang	v=2&lang=en-us	linkedin.com	/		FALSE	TRUE	None
	JSESSIONID	ajax:151214191	www.linkedin.co	/		FALSE	TRUE	None
<a href="https://www.pinterest.com/">https://www.pinterest.com/</a>	sessionFunnelEver	1	www.pinterest.co	/	1,669,610,332	FALSE	TRUE	
	_auth	0	pinterest.com	/	1,685,119,132	TRUE	TRUE	
	_pinterest_sess	TWc9PSZhV2Fc	pinterest.com	/	1,685,119,132	TRUE	TRUE	None
	_routing_id	222205d1-030e-	www.pinterest.co	/	1,669,653,532	TRUE	FALSE	
	csrftoken	998e88fddb456	www.pinterest.co	/	1,685,119,132	FALSE	TRUE	Lax
<a href="https://www.tumblr.com/">https://www.tumblr.com/</a>	tz	Asia%2FCalcutta	www.tumblr.com	/	1,670,171,935	FALSE	FALSE	
	_parseely_visitor	{%22id%22:%22	tumblr.com	/	1,670,171,935	FALSE	FALSE	
	_parseely_session	{%22sid%22:1%	tumblr.com	/	1,669,568,935	FALSE	FALSE	
	test		www.tumblr.com	/explore		FALSE	FALSE	
<a href="https://www.reddit.com/">https://www.reddit.com/</a>	session_tracker	qgohahaqbceair	.reddit.com	/	1,669,574,341	FALSE	TRUE	None
	datadome	3cW4u7pHAe-X	.reddit.com	/	1,670,171,941	FALSE	TRUE	Lax
	USER	eyJwcmVmcyl0e	.reddit.com	/	1,670,171,940	FALSE	FALSE	
	edgebucket	jZ28qYpgPoUHC	.reddit.com	/	1,685,119,137	FALSE	TRUE	
	csv	2	.reddit.com	/	1,685,119,137	FALSE	TRUE	None
	show_announceme	yes	.reddit.com	/	1,670,171,940	FALSE	FALSE	
	token_v2	eyJhbGciOiJIUz	.reddit.com	/	1,685,119,137	TRUE	TRUE	
	loid	0000000000ulkk	.reddit.com	/	1,685,119,137	FALSE	TRUE	None

Now that we could get the cookies, we tried to get those cookies that were being used to store the login information, and we tried to automate by attaching those cookies with the request or when we were getting the cookies but we were not very successful.

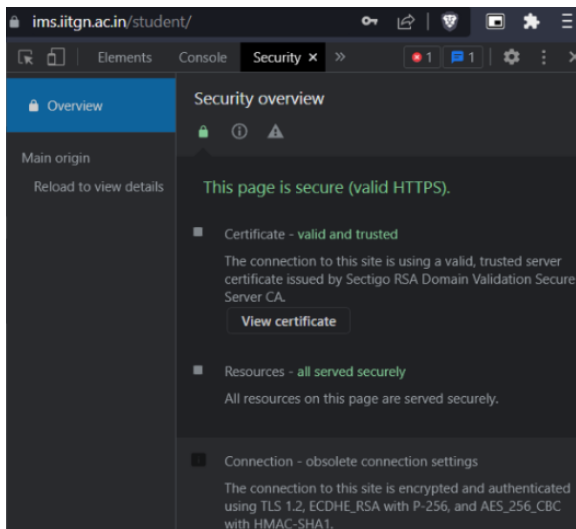


3. Second part filters out the sites containing iitgn in their urls.
4. Last part saves the output in the file found.txt.

With the help of hakrawler we built a database of more than 2500 IITGN websites. Although after analyzing the database, we found that many URLs were static, for example, the course web pages of all the professors. These static websites did not collect any cookies and hence were of no use to us. We also found that a lot of iitgn websites were "http" instead of "https". After visiting all those URLs, we learned that most of these websites redirect to another URL that displays the same content as before, except it is "https". Although a handful of websites did not redirect to its "https" version and were still not secure. One of them is the newsletter website, where we give our email ID and other crucial information to the website. This motivated us to dive deep into the secure attribute and TLS version of the IITGN websites.

### 3.5 Secure attribute and the TLS version

Next part, we set our eyes on getting the TLS/SSL information about the websites and how to automate that. We knew we could get the information from the browser from the developer tools. Here we can observe TLS 1.2 in the Connection Section in the Security Tab in the Developer Tools window of the browser.



The challenging part about this was figuring out how to get this information from the browser. So we looked at several things:

- First, we searched extensively about whether a python library could get us this information directly from the URL. But we could not find one.



- Next, we tried looking at Mozilla OpenSource projects and seeing if any framework had been developed or used in some way. But since we were unfamiliar with other languages, we decided to skip this part after devoting some time to this.
- Finally, after discussing this with the course professor, he provided us with some code that was related to getting the TLS info.

Time	TLS version	SNI	Source IP address	Destination IP address	Source port	Destination Port	Protocol	Downloaded Data size (bytes)	TLS session duration (s)	Foreground /Background	SSL Certificate information
1667930465.521010000	NA	NA	10.0.2.15	10.0.136.7	37789	53	UDP	688	NA	0	0
1667930465.524583000	NA	NA	10.0.136.7	10.0.2.15	53	37789	UDP	3136	NA	0	0
1667930465.525012000	TLS_1_2	www.wikipedia.org	10.0.2.15	103.102.166.224	33872	443	TCP	6768	0.58471989631652	0	0
1667930465.610786000	TLS_1_2	NA	103.102.166.2	10.0.2.15	443	33872	TCP	648744	0.49926495552062	0	0
1667930480.141397000	NA	NA	10.0.2.15	10.0.136.7	57404	53	UDP	656	NA	0	0
1667930480.145262000	NA	NA	10.0.136.7	10.0.2.15	53	57404	UDP	1432	NA	0	0
1667930480.145995000	TLS_1_2	ims.iitgn.ac.in	10.0.2.15	10.0.137.79	38178	443	TCP	7464	0.06303000450134	0	0
1667930480.148762000	TLS_1_2	NA	10.0.137.79	10.0.2.15	443	38178	TCP	443240	0.06023812294006	0	0

This was one of the outputs that we could get. Now, if we take a look at the TLS versions, we can observe that it is displaying just TLS 1.2.

After careful deliberation, we figured out what the issue was. The problem was that the code was using the pyshark library of python. Now, pyshark is a python wrapper for tshark (CLI version of wireshark). PyShark is still in its development phase and thus is incapable of identifying the TLS 1.3 connections. (That is what some of the issues on its GitHub Repository said.)

And one other issue was that the code worked on analyzing the .pcap files (PCAP files are data files created using a program, usually Wireshark or Tcpdump. These files contain packet data of a network and are used to analyze the network characteristics.) So it did not work directly on URLs; however, we had to capture the packets and then analyze them.

However, some good did come out of this ordeal, we got the idea to utilize tshark to get our work done. By using the subprocess API of Python, we could start listening to the packets and then issue a get request to the given URL. After that, stop listening and analyze the recorded packets using tshark's filter function and get the TLS info after some regex string matching from the filtered packet information. Below is a snippet of the said code:

The `tls-version-checker.py` in the GitHub repository.

However, we had written the code inside the Kali VM environment, so the code should only be run inside Linux environments.

One of the challenges while writing the code for TLS using the subprocess API was concurrency and order because even though the code was meant to execute different commands sequentially, the internal scheduling made things difficult. Hence it took some time to make sure the order of commands remained consistent. For this, we had to use implicit sleeping using the `asyncio` library of python. Finally, this was the output for some of the government websites:

```

async def parse_tls(path):
    tls_versions = []
    f = open(path, "r")
    data = f.read()
    v = re.findall(r"TLSv\S\S\S", data)
    set_v = set(v)
    for i in set_v:
        tls_versions.append([i, v.count(i)])
    tls_versions.sort(key=lambda x: x[1], reverse=True)
    return tls_versions, set_v

async def run_tshark(url, path):
    subprocess.Popen("tshark -Y tls > " + path, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    await asyncio.sleep(3)
    subprocess.Popen("curl " + url, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    await asyncio.sleep(3)
    subprocess.Popen("kill tshark", shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    await asyncio.sleep(1)
    return

async def get_TLS_info(url, path):
    await run_tshark(url, path)
    v, set_v = await parse_tls(path)
    while len(set_v) == 0:
        await run_tshark(url, path)
        v, set_v = await parse_tls(path)
        await asyncio.sleep(1)
    print("TLS version(s) supported by " + url + " : " + str(v))
    global tls_versions
    tls_versions.append([url, v[0][0]])
    # print("Main " + url + " TLS version : " + str(v[0][0]))
    return

```

```

(sky@kali) - [~/Desktop/Networks_project]
$ sudo python3 test.py
TLS version(s) supported by http://www.uidai.gov.in/ : [['TLSv1.3', 12]]
TLS version(s) supported by http://www.aptsec.org/ : [['TLSv1.3', 6]]
TLS version(s) supported by http://www.assocharm.org/ : [['TLSv1.3', 6]]
TLS version(s) supported by http://www.bbnl.nic.in/ : [['TLSv1.3', 15]]
TLS version(s) supported by http://www.bsnl.co.in/ : [['TLSv1.3', 6]]
TLS version(s) supported by http://www.coai.com/ : [['TLSv1.2', 2]]
TLS version(s) supported by http://www.cdor.in/ : [['TLSv1.2', 18]]
TLS version(s) supported by http://commerce.gov.in/ : [['TLSv1.3', 3]]

```

## 3.6 Attempted Demo of Basic ARP Spoofing

### 3.6.1 Required Dependencies and Run Instructions

- installation of scapy library - **pip install scapy**
- To run the code, go to the directory where **argspoofer.py** lives.
- Now, run the code using **python argspoofer.py**
- Change the victim IP and router IP depending on your subnet and then run the code. Use Ctrl+C to exit the code's execution.
- Analyze **change or no change** in victim's ARP table entries for gateway router.

### 3.6.2 Definition

ARP spoofing, ARP cache poisoning, or ARP poison routing, is a technique by which an attacker sends (spoofed) Address Resolution Protocol (ARP) messages onto a local area network. Generally, the aim is to associate the attacker's MAC address with the IP address of another host, such as the default gateway, causing any traffic meant for that IP address to be sent to the attacker instead.

### 3.6.3 Uses

This method, if successful, may allow an attacker to intercept data frames on a network, modify the traffic, or stop all traffic. Often the attack is used as an opening for other attacks, such as denial of service, man-in-the-middle, or session hijacking attacks.

### 3.6.4 Hacker IP and MAC

Obtain the hacker's (your) IP and MAC address using the ipconfig command. Every device that supports IPv4 addressing would have an ARP table.

### 3.6.5 Attack Idea

The main attack idea was to change the ARP Table Entry on the victim's machine for the gateway router to the attacker's MAC Address and the ARP Table Entry on the Router for the victim machine to the attacker's MAC Address. In this way, the attacker becomes the man-in-the-middle.

Theoretically and practically (using Wireshark), if I can intercept traffic from the victim, I can steal any cookies he sends to sites he revisits. Then I can use this cookie data to impersonate the victim.

```

Wireless LAN adapter Wi-Fi:

Connection-specific DNS Suffix . : 
Description . . . . . : Intel(R) Wi-Fi 6 AX201 160MHz
Physical Address. . . . . : 2C-DB-07-A0-1F-4F
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes
Link-local IPv6 Address . . . . . : fe80::6bd:c4f0:31c1:87e3%16(Preferred)
IPv4 Address. . . . . : 10.7.56.141(Preferred)
Subnet Mask . . . . . : 255.255.192.0
Lease Obtained. . . . . : 22 November 2022 10:04:28
Lease Expires . . . . . : 12 December 2022 22:26:03
Default Gateway . . . . . : 10.7.0.1
DHCP Server . . . . . : 1.1.1.1

```

- ARP table entry for the Gateway router in the table is dynamic and can be edited.
- This way, I can intercept traffic that the victim intends to send to the router and in the other direction as well.

### 3.6.6 Attack Implementation

The attack primarily uses the **scapy** library and its packet manipulation capabilities.

- We use the **scapy.ARP()** method and its
  - **pdst** attribute - to specify IP address of destination to which this ARP packet must go to.
  - **psrc** attribute - to specify the IP address on the ARP table of **pdst** which needs to be updated.
  - **hwdst** attribute - to specify the MAC Address of the receiver/victim.
  - **hwsrc** attribute - the MAC Address of the new entry corresponding to **psrc** in the ARP table of **pdst**.
- To get the MAC address corresponding to a particular IP address in the subnetwork, we send an **Ether** broadcast message to the entire and obtain the MAC address of interface corresponding to **pdst** = victim's IP.
- The next step is to send the spoofed ARP packets with the correct attributes repeatedly because ARP table entries are periodically refreshed by the gateway routers- so the attack needs to be persistent and not one-time.

```
def mac_from_ip(ip):
    arp_req = scapy.ARP(pdst = ip)                                #set the required ip whose mac wants to be known a
    broadcast = scapy.Ether(dst = "ff:ff:ff:ff:ff:ff")            #send the packet out on a broadcast to the whole s

    arp_req_broadcast = broadcast / arp_req                        #can automatically set common fields u

    answered_list = scapy.sr(arp_req_broadcast, timeout = 5, verbose = False)[0]                            #assuming response
    return answered_list[0][1].hwsrc
```

- After the attack is over, we can restore the ARP tables at the gateway and the victim by sending the un-spoofed ARP packets with the correct set attributes.

### 3.7 Observations

## 3.8 Cookie Poisoning

Cookie poisoning, also known as session hijacking, is a type of attack in which the attacker modifies, tampers with, hijacks, or otherwise "poisons" a cookie that is sent back to the server but is otherwise legitimate in order to steal data, get around security, or both. Thus, once information about a user's cookies is gained, the attacker can pretend to be the user and gain access to user information and resources. Since cookies often hold very sensitive information, this can lead to extreme information leaks and security breaches.

Following are some fundamental ways of performing cookie poisoning and their implementation on some of the IITGN websites:

### 3.8.1 Client-Side Cookie Tampering

The cookies that the currently set can be viewed and altered in the developers console. A vulnerable web application may keep critical state data, such as a flag allowing admin access, directly in cookies. In that instance, manually altering cookie names and values and reloading the page might be sufficient to obtain elevated access. Although it would be difficult in the era of web frameworks to find a production website or application that manually codes cookies in such a vulnerable way, it is theoretically feasible. While such vulnerabilities are not usually found in websites, one can use the `document.cookie` property in JavaScript to manipulate the cookie values from script. Thus, in case of a sensitive cookie like a session cookie not having the `httpOnly` flag, it can be read and altered by an attacker following a cross-site scripting attack (XSS).

### 3.8.2 XSS and the same origin policy (SOP)

SOP stops one website from reading the data from another website by checking three things in the protocol. Protocol, origin and the policy. Only if all of them are the same then only when cross origin read and write are allowed. XSS attacks bypass this security measure by directly injecting code into the target website. Websites are usually aware about such attacks and take precautionary measures to ensure that it does not happen, like detection of language tags, which are then discarded. There are mainly 3 types of XSS attacks -

- Reflected XSS, where the current HTTP request is where the malicious script originates.
- Stored XSS, where the vulnerability comes from the website's database.
- DOM XSS, which used the vulnerabilities present in the client-side.

We tried injecting various script codes into the website . However even the rare commands without the script tag, did not successfully penetrate the system.

## Subscribe to our newsletters

---

\* required fields are marked red

The following required values are missing: Title

Email address \*

Confirm your email address \*

☒ I prefer to receive emails in HTML format

Title \*

Full Name \*

Phone

State

Country

Company

Address

```
<body onmessage=print()>
<body onpageshow=alert(1)>
```

Figure 1: XSS Attack on the newsletter portal

### **3.8.3 SQL injection :**

An attacker can tamper with database queries that an application makes thanks to a vulnerability in internet security known as SQL injection. In most circumstances, it gives an attacker access to data that they otherwise would not have. This may include data from other users or any other data the programmer has access to. In many instances, an attacker can update or remove this data, permanently altering the application's behavior or content. An attacker may sometimes use a denial-of-service attack or escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure. SQL attacks can also be used, in theory, to extract information about a target website's cookies. The websites successfully evaded all the SQL injection attacks that we tried.

### **3.8.4 Session Hijacking :**

Session hijacking is a type of attack where a malicious actor takes over a user's session on a network in order to obtain sensitive information. Some of the most common types of session hijacking are IP spoofing and man-in-the-middle attacks. IP spoofing involves a hacker disguising his or her IP address as a legitimate IP address on a network. Man-in-the-middle attacks occur when a hacker inserts himself in the communication channel between two or more systems on a network. We used a set of tools and commands present on kali in order to try and perform session hijacking.

Session hijacking involves an attacker taking over a user's session on a network during a session in order to gain sensitive data. IP spoofing and man-in-the-middle attacks are two of the most prevalent methods of session hijacking. IP spoofing is the practice of a hacker posing as a valid IP address on a network. Attacks known as "man-in-the-middle" take place when a hacker enters the communication path between two or more systems on a network. We used a set of tools and commands present on kali in order to try and perform session hijacking.

### **EtterCap**

Ettercap is a software suite that enables users to launch man-in-the-middle attacks. In addition to this, Ettercap contains features that allow users to perform network sniffing and content filtering techniques. With these features, users are able to perform protocol analysis on target networks and hosts. We used a software package called Ettercap which enables users to start man-in-the-middle attacks on a given target host. Additionally, Ettercap has tools that enable users to employ content filtering and network sniffing strategies. These capabilities enable users to analyze protocols on target networks and hosts. However, we soon realized that the application could not be directly implemented in kali due to how the network settings in the virtual box were configured.

**The settings in Virtual Box and the reason for not working** - Virtual is in the NAT



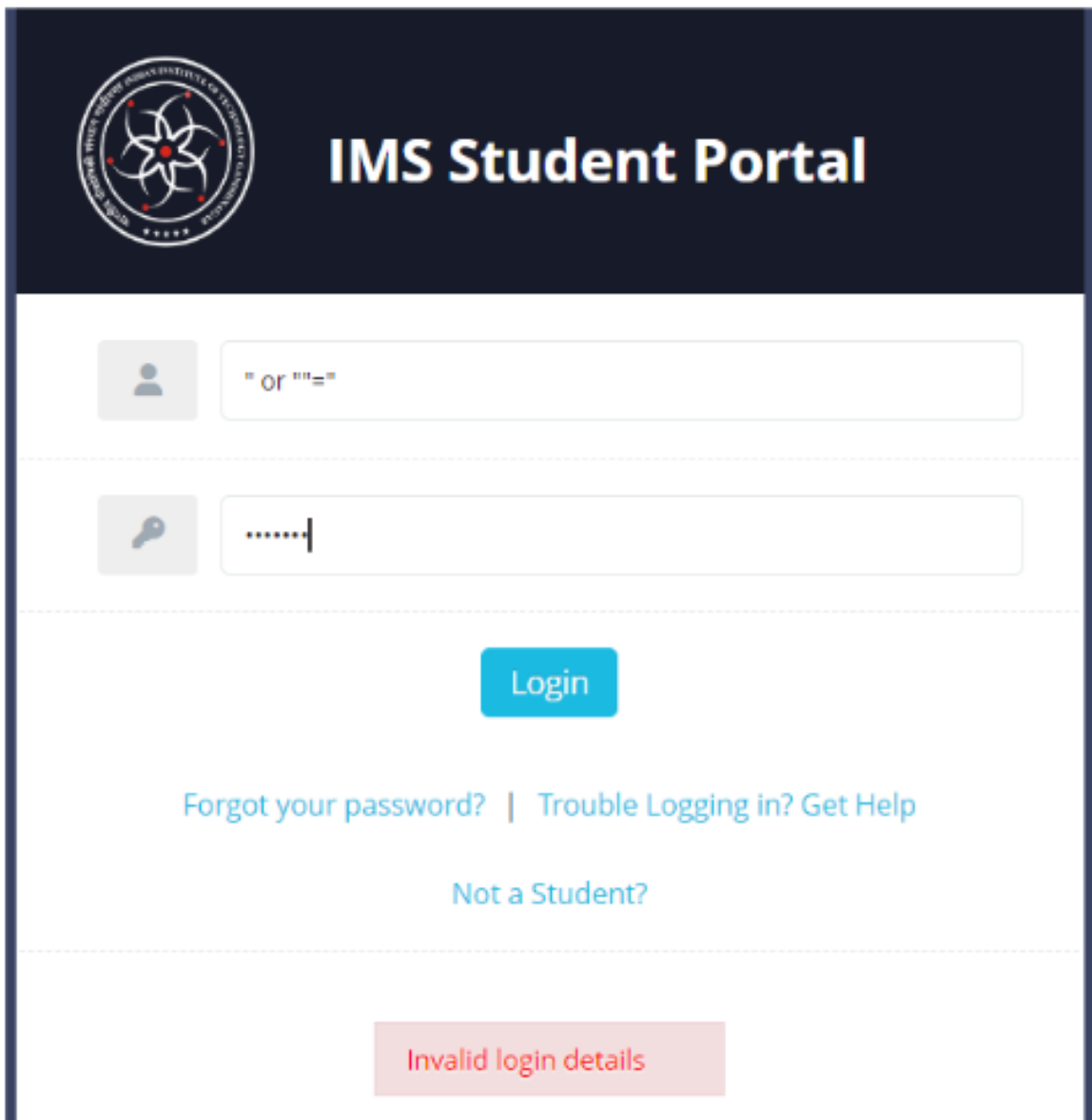


Figure 2: SQL Attack on the ims portal

mode by default, which means that it lives in a completely separate "physical" network from the LAN that the host is connected to. Although VirtualBox itself acts as a router that supports network-layer (L3) communication, ARP is a link-layer (L2) method that only works inside a single subnet and does not span routers. The VM must be in "bridged"

networking mode in order for this to function because it will then be directly connected to the host computer's network. Additionally, we realized that there is no certainty that bridging will function over Wi-Fi; many methods must be used by VirtualBox to make this happen. Specifically, the issue is that VirtualBox must somehow change ARP answers so that your VMs' actual MAC addresses are exposed over Wi-Fi; otherwise, they would appear to have the host's MAC. Since there was an uncertainty of this working over a wireless connection, we passed the hosts MAC address to the virtual box so that even though attacks could not be performed on the host, we would perform it on any other client on the IITGN network.

**Running and testing :** We tried performing a range of attacks on the network to try and check for vulnerabilities. After configuring the network setting accordingly, we were able to track all the IP address and their corresponding MAC addresses, that were connected to the network:

IP Address	MAC Address	Description
10.7.56.141	2C:DB:07:A0:1F:4F	
10.7.56.143	50:ED:3C:4D:D6:EA	
10.7.56.145	74:F2:FA:2C:12:12	
10.7.56.146	64:BC:58:02:37:5D	
10.7.56.149	84:FD:D1:15:7A:FC	
10.7.56.150	D8:B0:53:57:D9:27	
10.7.56.159	10:00:20:04:B7:D5	
10.7.56.160	46:8E:42:37:7E:4D	
Delete Host		Add to Target 1
		Add to Target 2
ARP poisoning victims:		
GROUP 1 : 10.7.0.1 00:00:5E:00:01:F6		
GROUP 2 : 10.7.56.141 2C:DB:07:A0:1F:4F		

Figure 3: Performing MITM attacks

Now a variety of man in the middle attacks were performed using Ettercap including the following:

- ARP attack
- ICMP attack
- NDP poisoning

The network was successfully able to evade these attacks. If successful, we would be able to

gain sensitive information about any of the users in the corresponding network, including their cookie information.

### 3.8.5 Some basic guidelines :

Maintenance of proper cookie hygiene is of almost importance in any web based application. Other than the fundamental attacks there are few practices that are usually advised to be followed in order to maintain the web service and prevent attacks :

- **Make use of the appropriate cookie flags and attributes:** There are numerous approaches to limit an attacker's ability to access cookies. It is possible to prevent scripts from accessing a cookie by using the `httpOnly` option. Setting the `secure` option to guarantee that the cookie is delivered over HTTPS exclusively is also advised, especially for session cookies. Cookie misuse may be reduced by carefully choosing the domain attribute's values; for more details, read our cookie security white paper.
- **Utilize distinct, safe session cookies:** Attackers shouldn't be able to obtain session IDs, and they should be produced randomly such that brute-forcing or guessing is difficult. Additionally, they ought to remain inaccessible once the session ends.
- **Use just one cookie per task:** Multipurpose cookies might be alluring, but doing so complicates operations and exposes users to security threats. Particularly session cookies shouldn't ever be utilized as password reset or anti-CSRF tokens.
- **Optimal session management:** Cookie security is crucial for session management because a single error might leave you vulnerable to assaults. Instead of starting from scratch, leveraging built-in session management tools from a reliable and established framework is the first step to adhering to best practices and avoiding basic mistakes.

## 4 Summary

We started from the research on cookies and their attributes. To collect the cookies, we implemented our own web crawler. We then moved to a faster and more professional web crawler also known as `hakrawler` to build the database of the IITGN websites and their cookies. We used `selenium` API to collect those cookies. We then moved to the TLS version and secure attribute of all the IITGN and many government websites and analyzed their vulnerabilities. Observing their vulnerabilities we deep-dived into the field of cookie poisoning and implemented a couple of basic attacks.

We have uploaded all the project related files in our Github repository.

Our final presentation can be seen through this link.

## 5 Takeaways and Specific Learnings

There were many takeaways for our team during the course of our project-

- Even if we take up a topic about which we have no practical knowledge, we can still end up with a strong output and learn a lot of theoretical and practical knowledge on the go.
- Working on a weekly timeline is a great incentive and motivator and this model helped us get the best output for our efforts. Meeting weekly with **Professor Sameer Kulkarni** gave the main direction to our project.
- Having a broad topic is not always a bad idea, we can initially explore the breadth and then analyze a few aspects in depth. This was the model we followed during the course of our project.
- Defining the problem statement at a later stage can help us learn a lot more about tools and programming than a rigid initial statement. We all would try to follow such a model in our future exploratory projects as well.

There were also many specific learnings that we all took from this project.

- We learnt in depth about cookies, their specific attributes, current industry implementation of cookies and cookie policies.
- We learnt about implementing web crawlers and parsing through HTML files to obtain information about cookies and store them in our database.
- Next, we also learnt how to use professional tools like Hakrawler to optimize our process further.
- We also had an in-depth analysis of TLS encryption of websites both in IITGN and government websites and prepared a tool using tshark and asyncio to extract the TLS version of websites and check if they are up-to-date or not. This can pose serious security threats.
- After this, we tried to implement basic security attacks like ARP spoofing and other man-in-the-middle attacks and this helped us learn more about the ARP protocol, scapy library, etc.

## 6 Acknowledgements

We would like to sincerely thank Prof. Sameer Kulkarni for his insightful remarks and recommendations for enhancing the paper's quality and guidance throughout the span of the term. Additionally, we are also grateful for the help we received in coding the TLS

version detection software from him. We would also like to thank our T.As Shoaib Alam and Pinky Kumari for guiding us and assisting us with our queries and doubts. We also appreciate all the friends that gave their valuable insights into the project and helped us along the way. Finally, we are grateful for IIT Gandhinagar’s Department of Computer Science for providing us with the opportunity and the resources required to work on this project.

## References

- [1] A. Cahn, S. Alfeld, P. Barford, and S. Muthukrishnan, “An empirical study of web cookies,” *Proceedings of the 25th International Conference on World Wide Web*, 2016.
- [2] S. Khodayari and G. Pellegrino, “The state of the samesite: Studying the usage, effectiveness, and adequacy of samesite cookies,” *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.
- [3] Q. Chen, P. Iliia, M. Polychronakis, and A. Kapravelos, “Cookie swap party: Abusing first-party cookies for web tracking,” *Proceedings of the Web Conference 2021*, 2021.
- [4] A. Ramesh, “Python - how to create an arp spoofer using scapy?” Jul 2020. [Online]. Available: <https://www.geeksforgeeks.org/python-how-to-create-an-arp-spoofing-using-scapy/>
- [5] M. Postument, “Arp spoofer with python and scapy,” Mar 2019. [Online]. Available: <https://mpostument.medium.com/arp-spoofing-with-python-and-scapy-b848d7bc15b3>
- [6] M. Vojtko, “The ultimate guide to session hijacking aka cookie hijacking,” Aug 2021. [Online]. Available: <https://www.thesslstore.com/blog/the-ultimate-guide-to-session-hijacking-aka-cookie-hijacking/>
- [7] R. Awati, “What is cookie poisoning and how can you protect yourself?” Nov 2021. [Online]. Available: <https://www.techtarget.com/searchsecurity/definition/cookie-poisoning>