S.Y. M.C.A SIDDHI S. KOTRE
SEMESTER III ROLL NO: 50

ETHICAL HACKING

Assignment 1: SQL Injection for Website Hacking

Aim : To demonstrate an SQL Injection attack on a login page and explore ways to prevent SQL Injection vulnerabilities in PHP applications.

Theory

SQL Injection is a code injection technique that exploits security vulnerabilities in a web application's software by manipulating SQL queries in the database layer. This vulnerability occurs when the application's input fields, such as login forms, are not properly sanitized, allowing attackers to input malicious SQL code.

By crafting specially designed input strings, an attacker can manipulate the SQL queries executed by the database, potentially gaining unauthorized access to sensitive information, modifying database entries, or executing administrative operations.

1. PHP Scripts:
· Create login.php, index.php, and logout.php.

Index.php

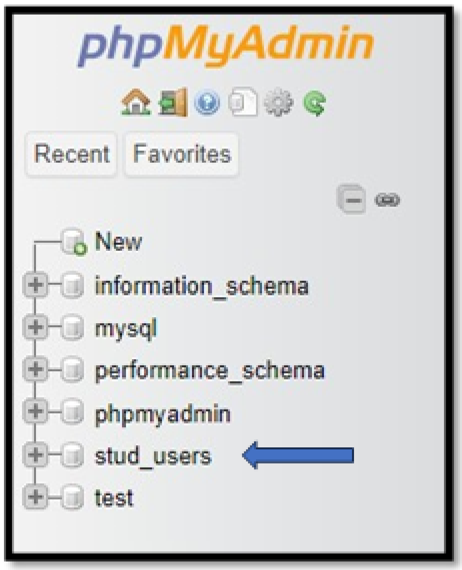
```
<?php
session_start();
if (!isset($_SESSION['loggedin'])) {
   header("Location: login.php");
   exit();
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Dashboard</title>
   <link rel="stylesheet" href="styles.css">
</head>
<body>
   <div class="container">
      <h2>Welcome to the Dashboard!</h2>
      <form action="logout.php" method="post">
         <button type="submit" class="logout-btn">Logout</button>
      </form>
   </div>
```

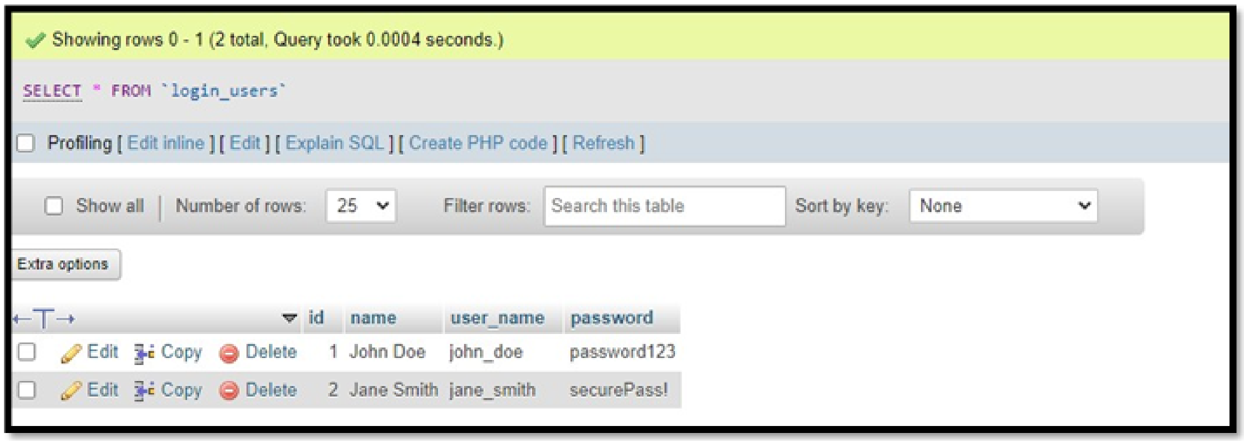
```
</body>
</html>
```

Logout.php

```
<?php
session_start();
$conn = mysqli_connect('localhost', 'root', '', 'stud_users');

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Vulnerable query for SQL Injection demonstration
    $sql = "SELECT * FROM login_users WHERE user_name = '$username' AND password =
'$password'";
    $result = mysqli_query($conn, $sql);

    if (mysqli_num_rows($result) > 0) {
        $_SESSION['loggedin'] = true;
        header("Location: index.php");
    } else {
        echo "<p class='error-message'>Invalid login credentials.</p>";
    }
}

$conn->close();
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h2>Login</h2>
        <form method="POST" action="">
            <input type="text" name="username" placeholder="Username" required><br>
            <input type="password" name="password" placeholder="Password" required><br>
            <input type="submit" value="Login">
        </form>
    </div>
</body>
</html>
```

Output:





Showing rows 0 - 1 (2 total, Query took 0.0004 seconds.)

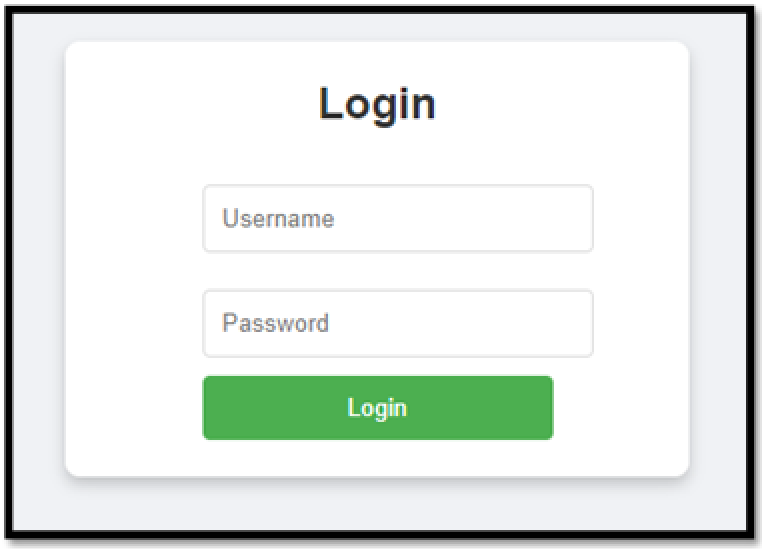
```
SELECT * FROM `login_users`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	id	name	user_name	password
Edit Copy Delete	1	John Doe	john_doe	password123
Edit Copy Delete	2	Jane Smith	jane_smith	securePass!

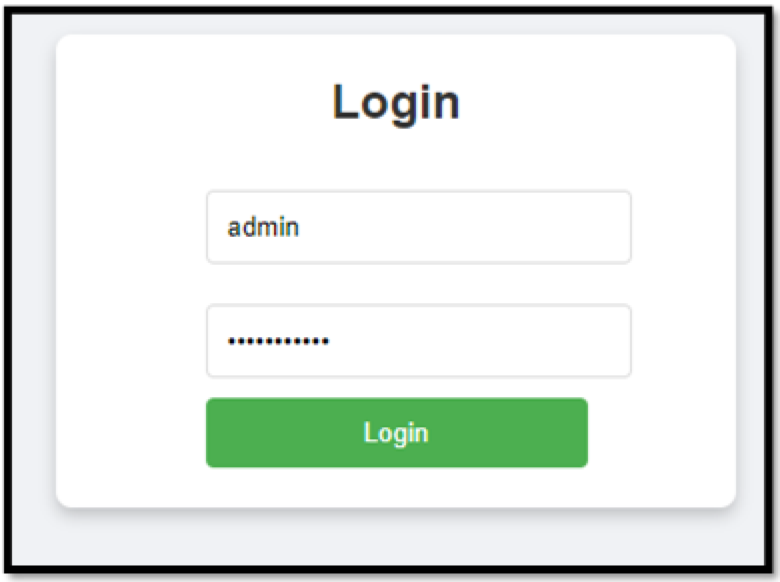


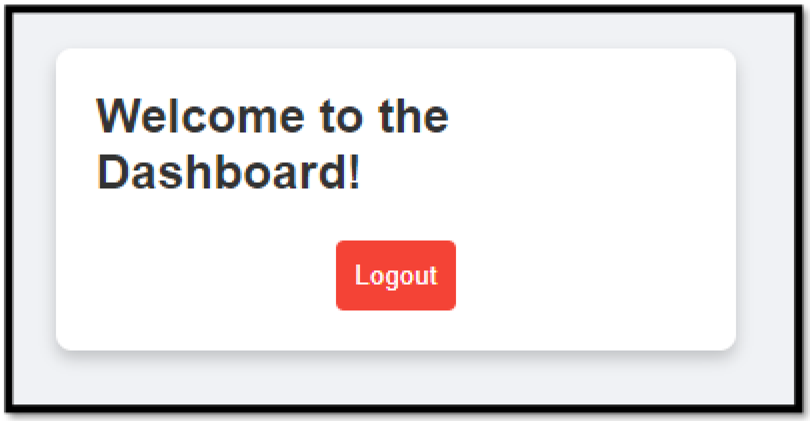
Demonstrating SQL Injection

1. SQL Injection Inputs

To demonstrate an SQL Injection attack on your login system, use the following inputs:

- **Username**: `admin`
- **Password**: `' OR '1'='1`





Welcome to the Dashboard!

Input Validation and Sanitization:

- Validate and sanitize all user inputs to ensure they conform to expected formats (e.g., correct data types, lengths, and characters). Reject any input that doesn't meet these criteria.

Limit Database User Privileges:

- Use the principle of least privilege by assigning database users only the necessary permissions they need for their tasks. This limits the potential damage from a successful SQL injection attack.

Error Handling:

- Avoid displaying detailed error messages to users. Instead, log errors internally and provide generic error messages. This prevents attackers from gaining insights into the database structure or query mechanics.

Assignment 2: Demonstrating Session Hijacking

Aim: To understand the process of session hijacking, identify potential vulnerabilities, and explore secure methods to prevent such attacks, including HTTPS, VPN, and secure coding practices.

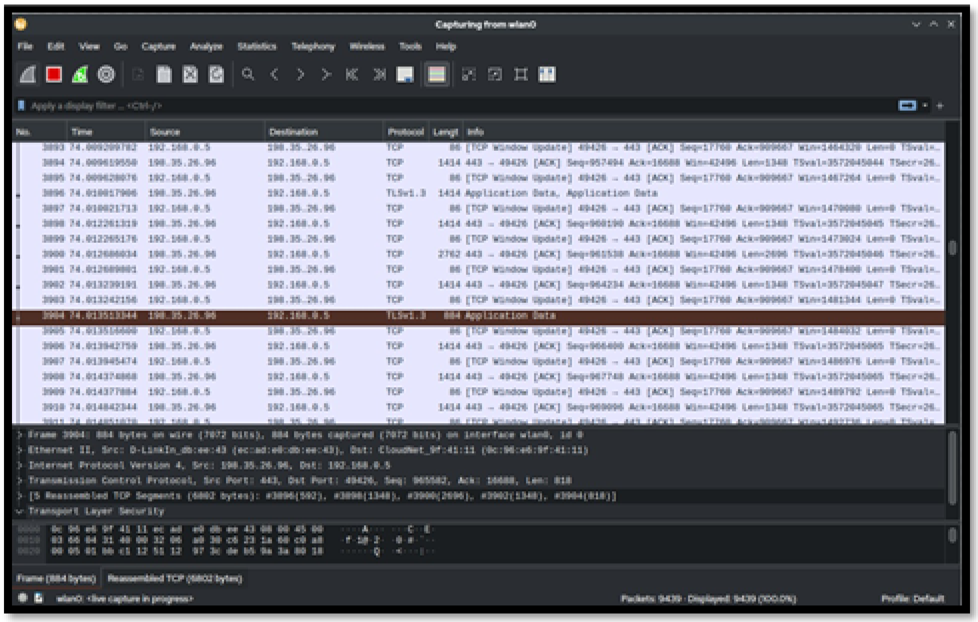
Theory

Session Hijacking is a web attack technique where an attacker exploits a valid computer session—often through the use of session IDs—to gain unauthorized access to information or services in a web application. This allows the attacker to impersonate the legitimate user, potentially accessing sensitive data or performing unauthorized actions on their behalf.

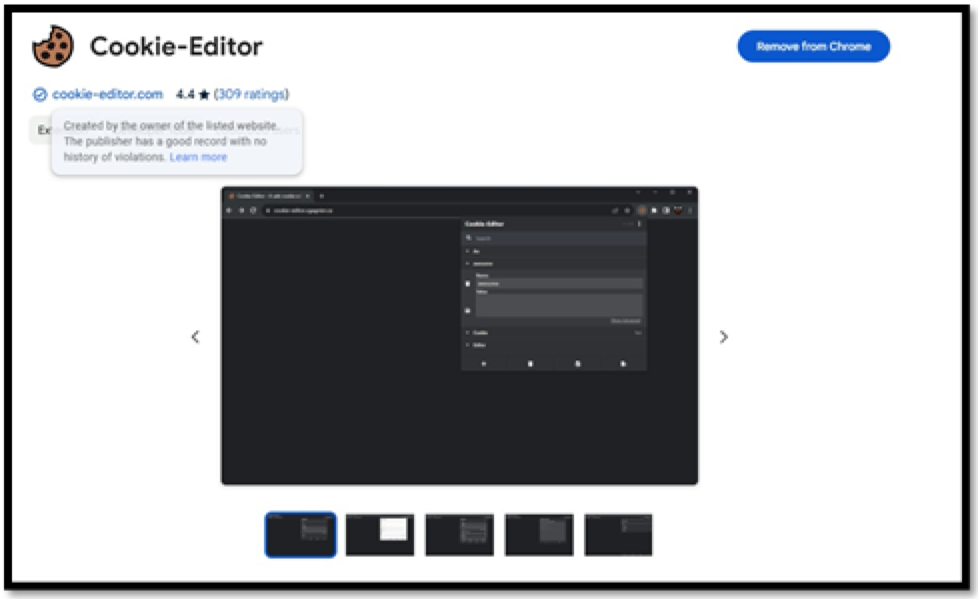
Mechanism of Session Hijacking:

1. Session Identification: When a user logs into a web application, a session ID is generated and stored on the client (often in cookies or local storage).
2. Session Theft: An attacker can use various methods (e.g., packet sniffing, cross-site scripting) to obtain this session ID.
3. Impersonation: Once the attacker has the session ID, they can use it to gain access to the victim's account and perform actions as if they were the legitimate user.

Output:

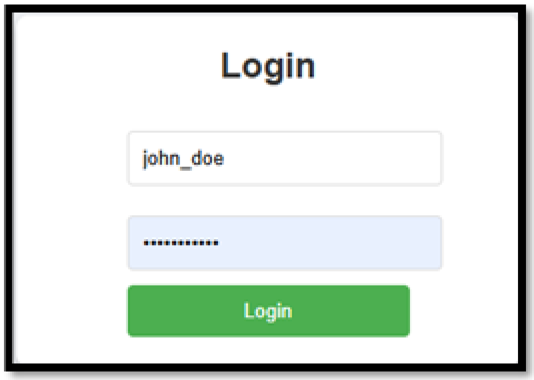

Capturing from wlan0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Frame (884 bytes)
Reassembled TCP (6802 bytes)

```
No.   Time          Source        Destination   Protocol Lengt Info
3893 74.009209782 192.168.0.5   198.35.26.96  TCP       86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1464320 Len=0 TSval=…
3894 74.009619550 198.35.26.96  192.168.0.5   TCP     1414 443 → 49426 [ACK] Seq=957494 Ack=16688 Win=42496 Len=1348 TSval=3572045044 TSecr=26…
3895 74.009628076 192.168.0.5   198.35.26.96  TCP       86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1467264 Len=0 TSval=…
3896 74.010017906 198.35.26.96  192.168.0.5   TLSv1.3 1414 Application Data, Application Data
3897 74.010021713 192.168.0.5   198.35.26.96  TCP       86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1470080 Len=0 TSval=…
3898 74.012261319 198.35.26.96  192.168.0.5   TCP     1414 443 → 49426 [ACK] Seq=960190 Ack=16688 Win=42496 Len=1348 TSval=3572045045 TSecr=26…
3899 74.012265176 192.168.0.5   198.35.26.96  TCP       86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1473024 Len=0 TSval=…
3900 74.012686034 198.35.26.96  192.168.0.5   TCP     2762 443 → 49426 [ACK] Seq=961538 Ack=16688 Win=42496 Len=2696 TSval=3572045046 TSecr=26…
3901 74.012689001 192.168.0.5   198.35.26.96  TCP       86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1478400 Len=0 TSval=…
3902 74.013239191 198.35.26.96  192.168.0.5   TCP     1414 443 → 49426 [ACK] Seq=964234 Ack=16688 Win=42496 Len=1348 TSval=3572045047 TSecr=26…
3903 74.013242156 192.168.0.5   198.35.26.96  TCP       86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1481344 Len=0 TSval=…
3904 74.013513344 198.35.26.96  192.168.0.5   TLSv1.3  884 Application Data
3905 74.013516600 192.168.0.5   198.35.26.96  TCP       86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1484032 Len=0 TSval=…
3906 74.013942759 198.35.26.96  192.168.0.5   TCP     1414 443 → 49426 [ACK] Seq=966400 Ack=16688 Win=42496 Len=1348 TSval=3572045065 TSecr=26…
3907 74.013945474 192.168.0.5   198.35.26.96  TCP       86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1486976 Len=0 TSval=…
3908 74.014374868 198.35.26.96  192.168.0.5   TCP     1414 443 → 49426 [ACK] Seq=967748 Ack=16688 Win=42496 Len=1348 TSval=3572045065 TSecr=26…
3909 74.014377884 192.168.0.5   198.35.26.96  TCP       86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1489792 Len=0 TSval=…
3910 74.014842344 198.35.26.96  192.168.0.5   TCP     1414 443 → 49426 [ACK] Seq=969096 Ack=16688 Win=42496 Len=1348 TSval=3572045065 TSecr=26…
3911 74.014851070 192.168.0.5   198.35.26.96  TCP       86 [TCP Window Update] 49426 → 443 [ACK] Seq=17760 Ack=909667 Win=1492736 Len=0 TSval=…

> Frame 3904: 884 bytes on wire (7072 bits), 884 bytes captured (7072 bits) on interface wlan0, id 0
> Ethernet II, Src: D-LinkIn_db:ee:43 (ec:ad:e0:db:ee:43), Dst: CloudNet_9f:41:11 (0c:96:e6:9f:41:11)
> Internet Protocol Version 4, Src: 198.35.26.96, Dst: 192.168.0.5
> Transmission Control Protocol, Src Port: 443, Dst Port: 49426, Seq: 965582, Ack: 16688, Len: 818
> [5 Reassembled TCP Segments (6802 bytes): #3896(592), #3898(1348), #3900(2696), #3902(1348), #3904(818)]
v Transport Layer Security

0000  0c 96 e6 9f 41 11 ec ad  e0 db ee 43 08 00 45 00   ····A·····C··E·
0010  03 66 04 31 40 00 32 06  a0 30 c6 23 1a 60 c0 a8   ·f·1@·2··0·#·`··
0020  00 05 01 bb c1 12 51 12  97 3c de b5 9a 3a 80 18   ······Q··<···:··

wlan0: <live capture in progress>          Packets: 9439 · Displayed: 9439 (100.0%)          Profile: Default
```


Cookie-Editor

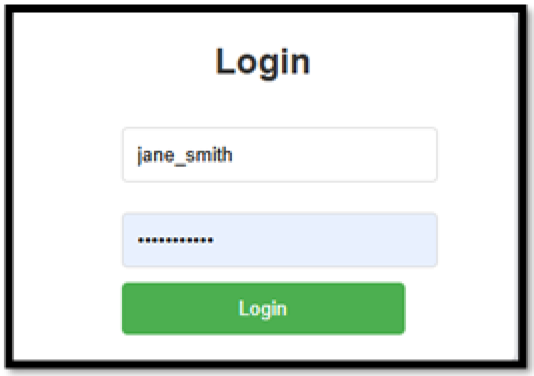
cookie-editor.com 4.4 ★ (309 ratings)

Created by the owner of the listed website. The publisher has a good record with no history of violations. Learn more


Login

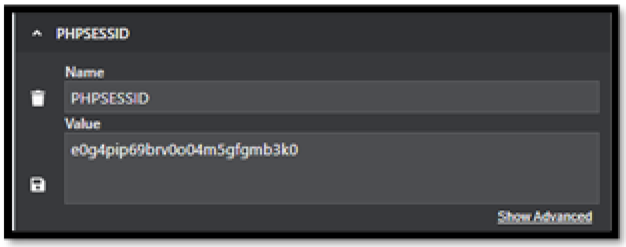
john_doe

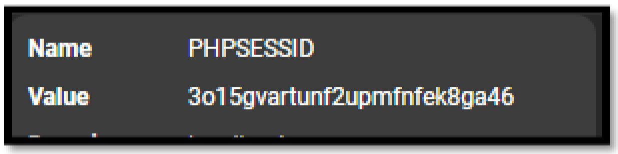
••••••••••••


Login

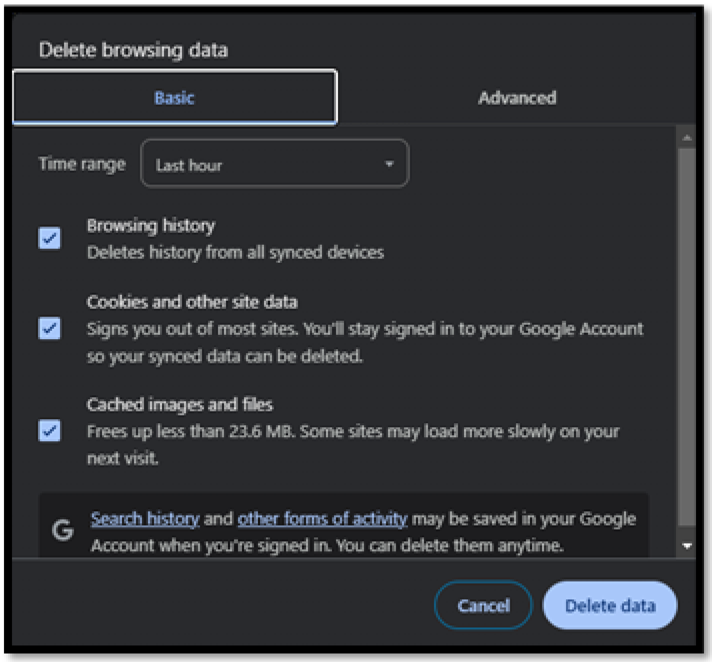
jane_smith

••••••••••••


PHPSESSID
Name
PHPSESSID
Value
e0g4pip69brv0o04m5gfgmb3k0



Name	PHPSESSID
Value	3o15gvartunf2upmfnfek8ga46

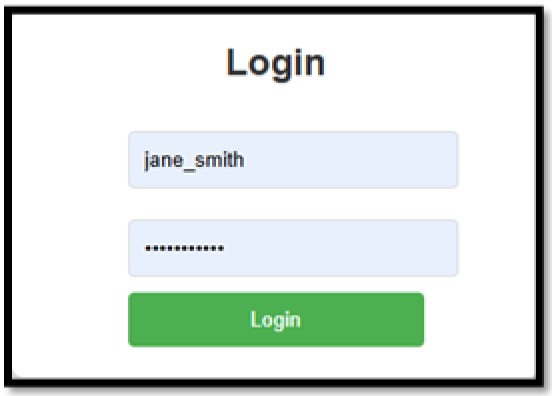

Delete browsing data

Time range: Last hour

- [✓] Browsing history
  Deletes history from all synced devices
- [✓] Cookies and other site data
  Signs you out of most sites. You'll stay signed in to your Google Account so your synced data can be deleted.
- [✓] Cached images and files
  Frees up less than 23.6 MB. Some sites may load more slowly on your next visit.

Search history and other forms of activity may be saved in your Google Account when you're signed in. You can delete them anytime.

After inputting user2 session ID in user 1 browser and refreshing the page


Login
jane_smith
••••••••••••