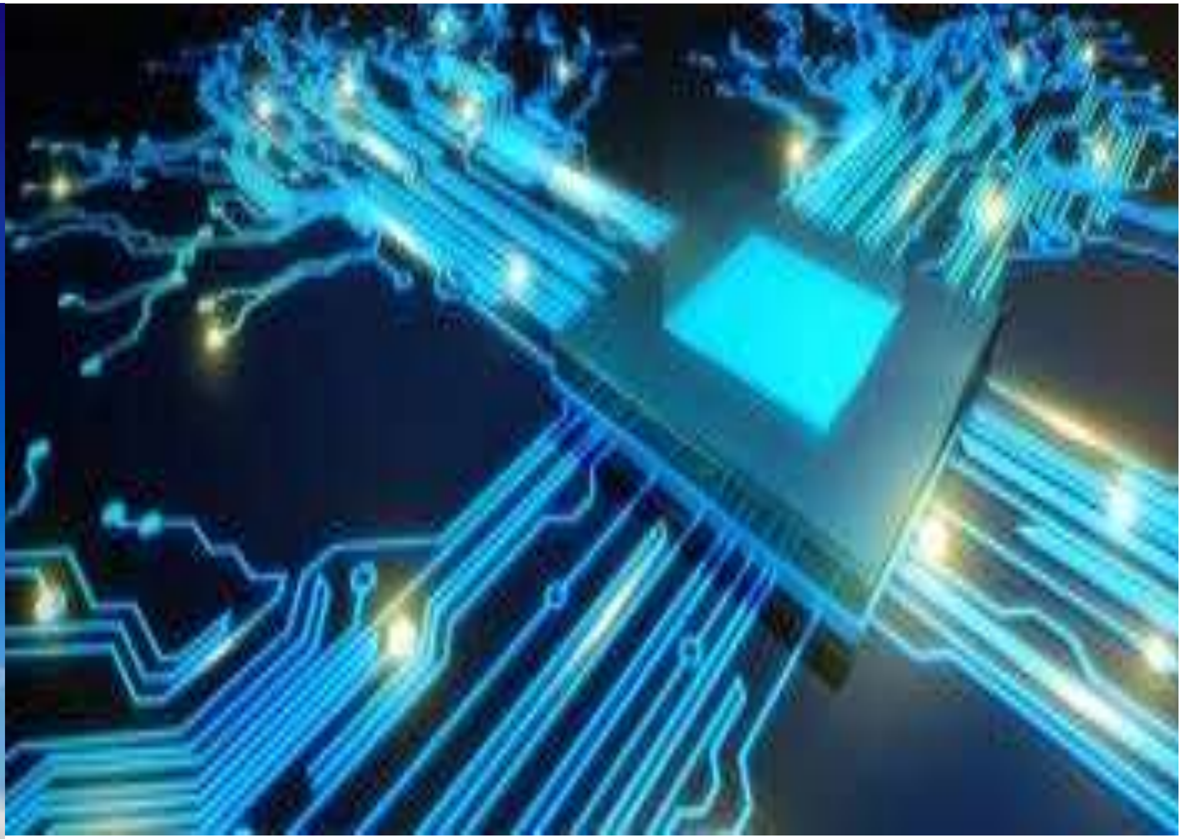


Web Security

Module-1
Introduction to Web Security

Background

- Many sensitive tasks are done through web
  - Online banking, online shopping
  - Database access
  - System administration

- Web applications and web users are targets of many attacks
  - Cross site scripting
  - SQL injection
  - Cross site request forgery
  - Information leakage
  - Session hijacking

Introduction

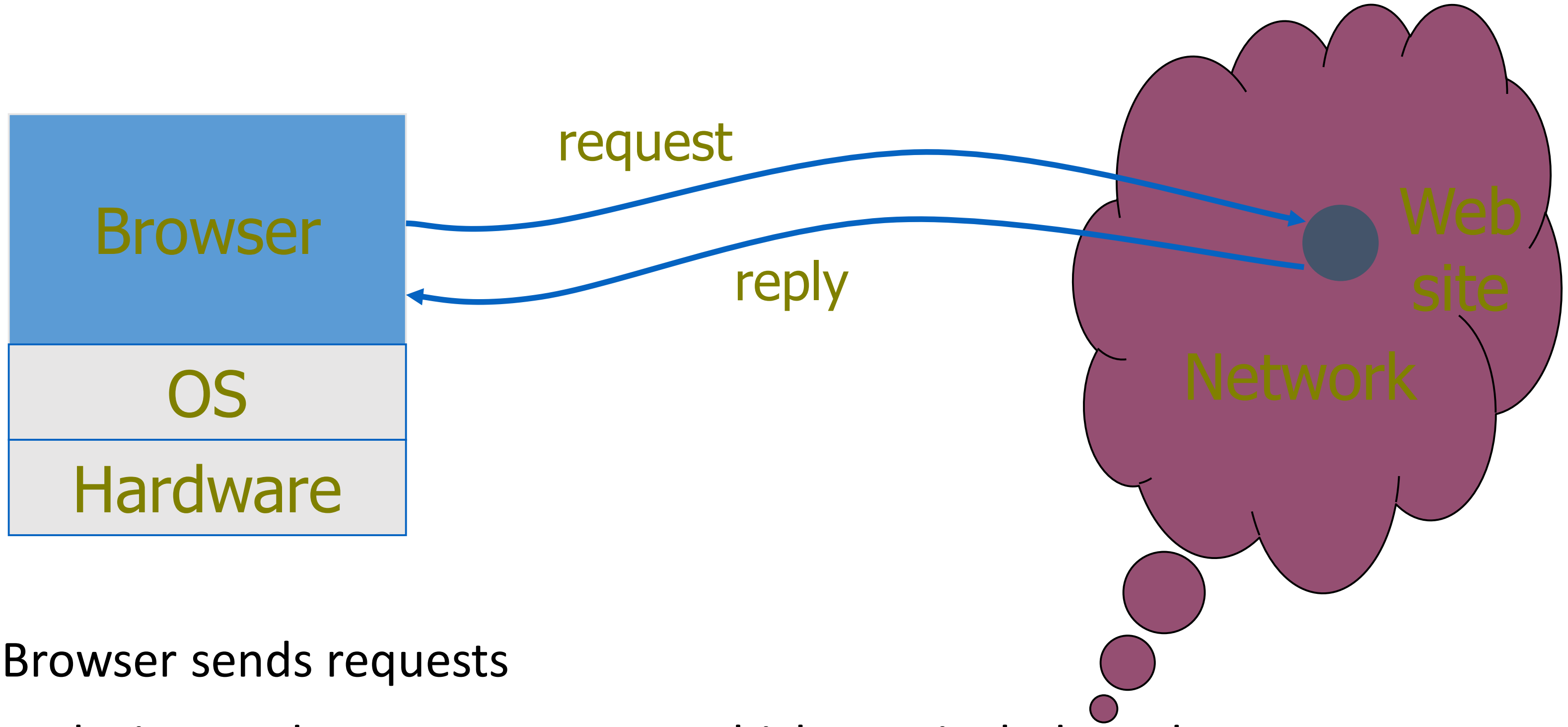
Web application security is a branch of Information Security that deals specifically with security of websites, web applications and web services.

At a high level, Web application security draws on the principles of application security but applies them specifically to Internet and Web systems.

Process of securing confidential data stored online from unauthorized access and modification.

Main goal of Web Security to prevent, Detect, and respond.

Web Browser and Network



- Browser sends requests
- Web site sends response pages, which may include code
- Interaction susceptible to network attacks

What is a “Secure” Computer System?

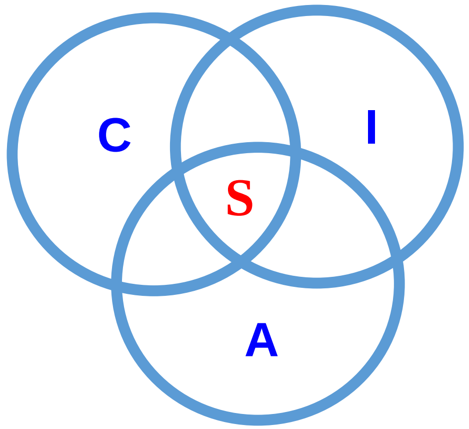
- To decide whether a computer system is “secure”, you must first decide what “secure” *means to you*, then identify the threats you care about.

You Will Never Own a Perfectly Secure System!

- Threats - examples
    - Viruses, trojan horses, etc.
    - Denial of Service
    - Stolen Customer Data
    - Modified Databases
    - Identity Theft and other threats to personal privacy
    - Equipment Theft
    - Espionage in cyberspace
    - Hack-tivism
    - Cyberterrorism
    - …

Basic Components of Security: Confidentiality, Integrity, Availability (CIA)

- CIA
  - Confidentiality: Who is authorized to use data?
  - Integrity: Is data „good?"
  - Availability: Can access data whenever need it?



S = Secure

- CIA or CIAAAN… ☺

  (other security components added to CIA)
  - Authentication
  - Authorization
  - Non-repudiation
  - …

Need to Balance CIA

- Example 1: C vs. I+A
  - Disconnect computer from Internet to increase confidentiality
  - Availability suffers, integrity suffers due to lost updates

- Example 2: I vs. C+A
  - Have extensive data checks by different people/systems to increase integrity
  - Confidentiality suffers as more people see data, availability suffers due to locks on data under verification)

Confidentiality

- "Need to know" basis for data access
  - How do we know who needs what data?

    Approach: access control specifies *who* can access *what*
  - How do we know a user is the person she claims to be?

    Need her identity and need to verify this identity

    Approach: identification and authentication
- Analogously: "Need to access/use" basis for physical assets
  - E.g., access to a computer room, use of a desktop

Integrity

- Integrity vs. Confidentiality
  - Concerned with unauthorized *modification* of assets (= resources)
    Confidentiality - concered with *access* to assets

  - Integrity is more difficult to *measure* than confidentiality
    Not binary – degrees of integrity
    Context-dependent - means different things in different contexts
    Could mean *any subset of* these asset properties:
    { precision / accuracy / currency / consistency /
    meaningfulness / usefulness / ...}
- Types of integrity—an example
  - Quote from a politician
  - Preserve the quote (data integrity) but misattribute (origin integrity)

Availability (1)

- Not understood very well yet

  „[F]ull implementation of availability is security's next challenge"

  E.g. Full implemenation of availability for Internet users (with ensuring security)

- Complex

  Context-dependent

  Could mean *any subset of* these asset (data or service) properties :

  { usefulness / sufficient capacity /

  progressing at a proper pace /

  completed in an acceptable period of time / ...}

Availability (2)

- We can say that an asset (resource) is available if:
  - Timely request response
  - Fair allocation of resources (no starvation!)
  - Fault tolerant (no total breakdown)
  - Easy to use in the intended way
  - Provides controlled concurrency (concurrency control, deadlock control, ...)

4. Vulnerabilities, Threats, and Controls

- Understanding Vulnerabilities, Threats, and Controls
  - Vulnerability = a weakness in a security system
  - Threat = circumstances that have a *potential* to cause harm
  - Controls = means and ways to block a threat, which tries to exploit one or more vulnerabilities
    - Most of the class discusses various controls and their effectiveness

[Pfleeger & Pfleeger]

- Example - New Orleans disaster (Hurricane Katrina)
  - Q: What were city vulnerabilities, threats, and controls?
  - A: Vulnerabilities: location below water level, geographical location in hurricane area, …

    Threats: hurricane, dam damage, terrorist attack, …

    Controls: dams and other civil infrastructures, emergency response plan, …

- Attack (materialization of a vulnerability/threat combination)
  - = exploitation of one or more vulnerabilities by a threat; tries to defeat controls
    - Attack may be:
      - *Successful* (a.k.a. an *exploit*)
        - resulting in a breach of security, a system penetration, etc.
      - *Unsuccessful*
        - when controls block a threat trying to exploit a vulnerability

[Pfleeger & Pfleeger]

Kinds of Threats

- Kinds of threats:
  - Interception
    - an unauthorized party (human or not) gains access to an asset
  - Interruption
    - an asset becomes lost, unavailable, or unusable
  - Modification
    - an unauthorized party changes the state of an asset
  - Fabrication
    - an unauthorized party counterfeits an asset

[Pfleeger & Pfleeger]

Levels of Vulnerabilities / Threats

- D) for other assets (resources)
  - including. people using data, s/w, h/w
- C) for data
  - „on top” of s/w, since used by s/w
- B) for software
  - „on top” of h/w, since run on h/w
- A) for hardware

[Pfleeger & Pfleeger]

Web Security/Privacy Issues

- Secure communications between client & server
  - HTTPS (HTTP over Secure Socket Layer)
- User authentication & session management
  - Cookies & other methods
- Active contents from different websites
  - Protecting resources maintained by browsers
- Web application security
- Web site authentication (e.g., anti-phishing)
- Privacy concerns

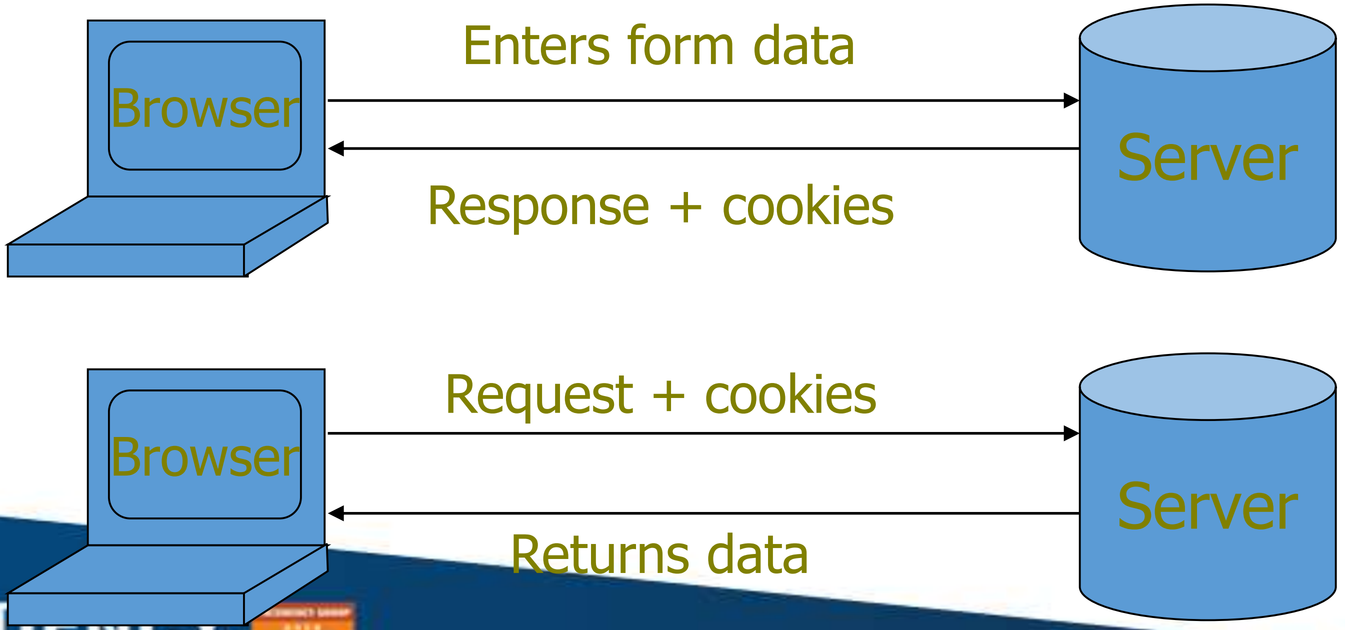
HTTP: HyperText Transfer Protocol

- Browser sends HTTP requests to the server
  - Methods: GET, POST, HEAD, …
  - GET: to retrieve a resource (html, image, script, css,…)
  - POST: to submit a form (login, register, …)
  - HEAD
- Server replies with a HTTP response
- Stateless request/response protocol
  - Each request is independent of previous requests
  - Statelessness has a significant impact on design and implementation of applications

Use Cookies to Store State Info

- Cookies
  - A cookie is a name/value pair created by a website to store information on your computer

Http is stateless protocol; cookies add state



Cookies Fields

- An example cookie from my browser

• Name	session-token
• Content	"s7yZiOvFm4YymG…."
• Domain	.amazon.com
• Path	/
• Send For Any type of connection	
• Expires	Monday, September 08, 2031 7:19:41 PM

Cookies

- Stored by the browser
- Used by the web applications
  - used for authenticating, tracking, and maintaining specific information about users
    - e.g., site preferences, contents of shopping carts
  - data may be sensitive
  - may be used to gather information about specific users

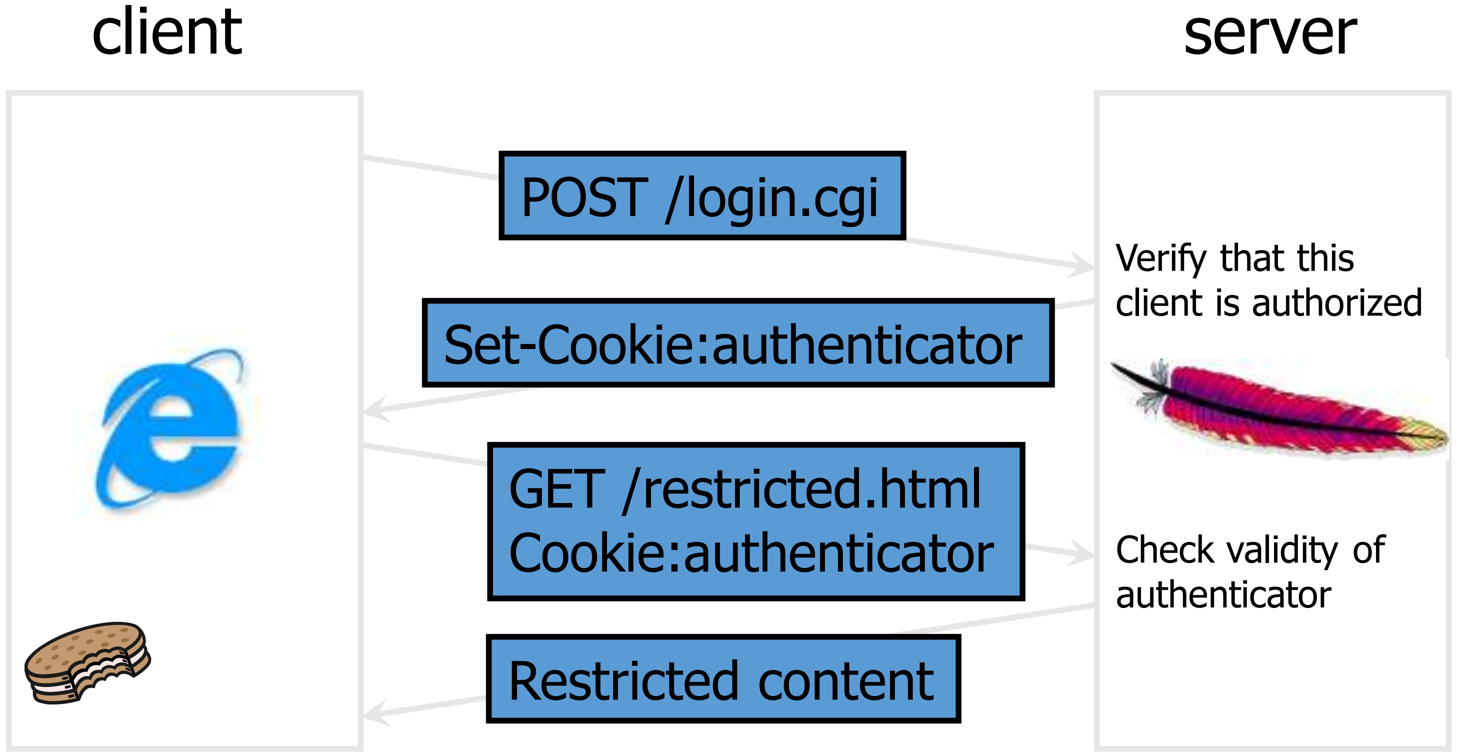
- Cookie ownership
  - Once a cookie is saved on your computer, only the website that created the cookie can read it

Web Authentication via Cookies

- HTTP is stateless
  - How does the server recognize a user who has signed in?

- Servers can use cookies to store state on client
  - After client successfully authenticates, server computes an authenticator and gives it to browser in a cookie
    - Client cannot forge authenticator on his own (session id)
  - With each request, browser presents the cookie
  - Server verifies the authenticator

A Typical Session with Cookies



Authenticators must be unforgeable and tamper-proof

(malicious clients shouldn't be able to modify an existing authenticator)

How to design it?

Security threats

- With the emergence of Web, increased information sharing through social networking and increasing business adoption of the Web as a means of doing business and delivering service, websites are often attacked directly.
- Hackers either seek to compromise the corporate network or the end-users accessing the website by subjecting them to drive-by downloading.

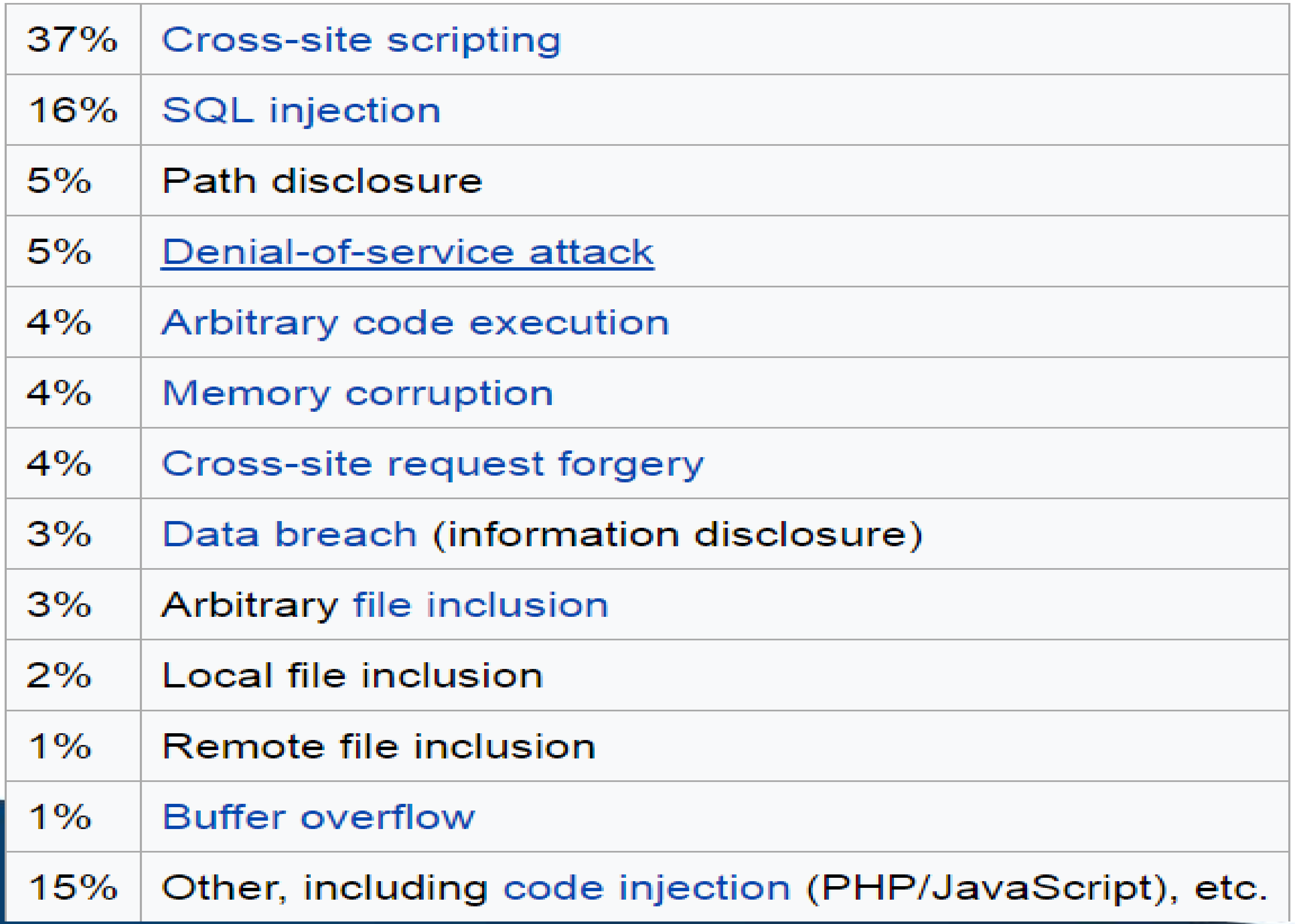
Security Threats (2)

Industry is paying increased attention to the security of the web applications themselves in addition to the security of the underlying computer network and operating systems.

The majority of web application attacks occur through cross-site scripting (XSS) and SQL injection attacks which typically result from flawed coding, and failure to sanitize input to and output from the web application.

Phishing is another common threat to the Web application and global losses from this type of attack in 2012 were estimated at $1.5 billion.

The Top Vulnerabilities



37%	Cross-site scripting
16%	SQL injection
5%	Path disclosure
5%	Denial-of-service attack
4%	Arbitrary code execution
4%	Memory corruption
4%	Cross-site request forgery
3%	Data breach (information disclosure)
3%	Arbitrary file inclusion
2%	Local file inclusion
1%	Remote file inclusion
1%	Buffer overflow
15%	Other, including code injection (PHP/JavaScript), etc.

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

Encoding Scheme

- **Character Encoding**
- **Image & Audio and Video Encoding**

Character Encoding

- ***Character encoding encodes characters into bytes***. It informs the computers how to interpret the zero and ones into real characters, numbers, and symbols. The computer understands only binary data; hence it is required to convert these characters into numeric codes.
- To achieve this, each character is converted into binary code, and for this, text documents are saved with encoding types.

There are different types of Character Encoding techniques

1. **HTML Encoding**
2. **URL Encoding**
3. **Unicode Encoding**
4. **Base64 Encoding**
5. **Hex Encoding**
6. **ASCII Encoding**

- HTML Encoding
- HTML encoding is used to display an HTML page in a proper format. With encoding, a web browser gets to know that which character set to be used.
- In HTML, there are various characters used in HTML Markup such as <, >. To encode these characters as content, we need to use an encoding.
- URL Encoding
- URL (Uniform resource locator) Encoding is used to ***convert characters in such a format that they can be transmitted over the internet***. It is also known as percent-encoding. The URL Encoding is performed to send the URL to the internet using the ASCII character-set. Non-ASCII characters are replaced with a %, followed by the hexadecimal digits.

- UNICODE Encoding
- Unicode is an encoding standard for a universal character set. It allows encoding, represent, and handle the text represented in most of the languages or writing systems that are available worldwide. It provides a code point or number for each character in every supported language. It can represent approximately all the possible characters possible in all the languages. A particular sequence of bits is known as a coding unit.
- A UNICODE standard can use 8, 16, or 32 bits to represent the characters.
- The Unicode standard defines Unicode Transformation Format (UTF) to encode the code points.

- **UTF-8 Encoding**
  The UTF8 is defined by the UNICODE standard, which is variable-width character encoding used in Electronics Communication. UTF-8 is capable of encoding all 1,112,064 valid character code points in Unicode using one to four one-byte (8-bit) code units.

- **UTF-16 Encoding**
- UTF16 Encoding represents a character's code points using one of two 16-bits integers.
- **UTF-32 Encoding**
  UTF32 Encoding represents each code point as 32-bit integers.

- Base64 Encoding
- Base64 Encoding is used to encode binary data into equivalent ASCII Characters. The Base64 encoding is used in the Mail system as mail systems such as SMTP can't work with binary data because they accept ASCII textual data only. It is also used in simple HTTP authentication to encode the credentials.

- ASCII Encoding
- **American Standard Code for Information Interchange** (ASCII) is a type of character-encoding. It was the first character encoding standard released in the year 1963.
- Th ASCII code is used to represent English characters as numbers, where each letter is assigned with a number from **0 to 127.** Most modern character-encoding schemes are based on ASCII, though they support many additional characters. It is a single byte encoding only using the bottom 7 bits. In an ASCII file, each alphabetic, numeric, or special character is represented with a 7-bit binary number. Each character of the keyboard has an equivalent ASCII value.

Image and Audio & Video Encoding

- Image and audio & video encoding are performed to save storage space. A media file such as image, audio, and video are encoded to save them in a more efficient and compressed format.
- These encoded files contain the same content with usually similar quality, but in compressed size, so that they can be saved within less space, can be transferred easily via mail, or can be downloaded on the system.
- . WAV audio file is converted into .MP3 file to reduce the size by 1/10th to its original size.

Outline

- Introduction to web application penetration testing
- Software setup
- Mapping and analyzing the application
- Bypassing client-side controls
- Attacking authentication
- Attacking session management
- Attacking data stores

Why the web?

- Let's look at some statistics:
  - Over 3.9 billion internet users in the world
  - Over 1.9 billion websites online
- We use websites for everything: e-commerce, online banking to social networking, social media, etc.
- Web security has become a major concern for businesses.
- Recent example: Equifax. The breach exposed the personal information of 143 million US users and an estimated 100,000 Canadian users.
- According to Trustwave's 2018 Global Security Report:
  - 100% of the web applications scanned by Trustwave displayed at least one vulnerability.
  - Median number of 11 vulnerabilities detected per application.

How to secure a web application?

Combination of techniques are used:

- Secure coding practices
- Web application firewalls
- Static code analysis
- **Web application penetration testing**
- Etc.

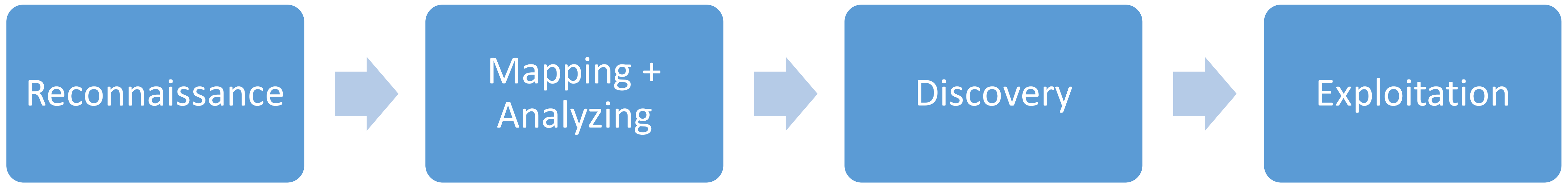
What is web app pen testing?

- Combination of manual and automated tests to identify vulnerabilities, security flaws and/or threats in a web application.
- Categorized into three types:
  - White box: Tester has complete access and in-depth knowledge of the system. Access to source code is usually given.
  - Black box: Tester is given little to no information about the system. Just the URL of the application is usually given.
  - Grey box: Combination of white box and black box penetration testing. Limited information and access is given to the tester.
- Several methodologies and guidelines: OWASP, PTES, PCI DSS, etc.
- Most important thing to keep in mind: you need permission to perform a security test!

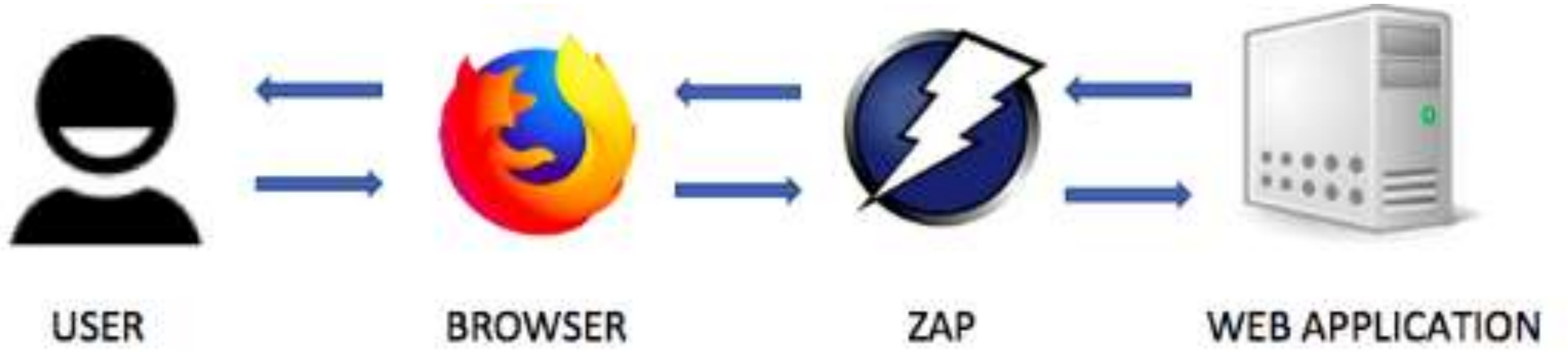
What is owasp?

- Stands for Open Web Application Security Project
- International open source community “dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted”.
- Contains widely used and popular tools such as the ZAP.
- OWASP TOP 10 Project

Abstract pen testing Methodology



How does a proxy work?



Mapping the application

- Explore the visible content
- Review public resources
- Identify any hidden content

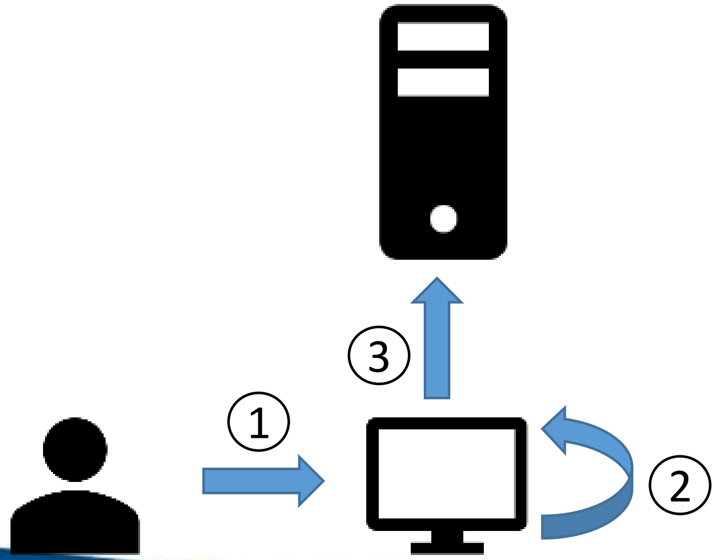
ANALYZING the application

- Identify functionality
- Identify data entry points
- Identify the technologies used
- Map the attack surface

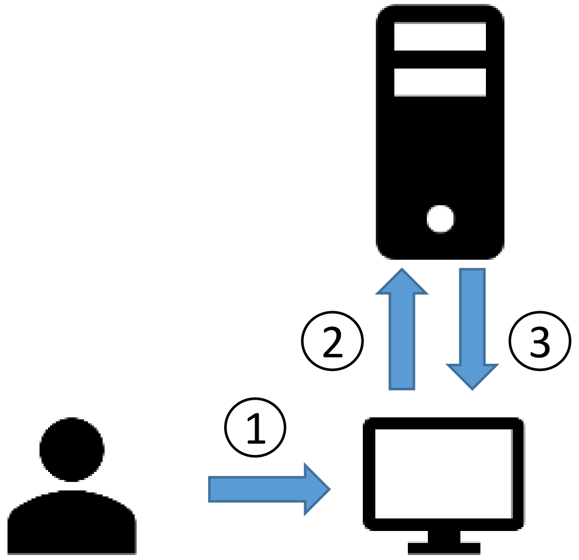
Bypassing Client-Side Controls

Client-Side vs server-side validation

Client-side



Server-side



Bypassing Client-Side Controls

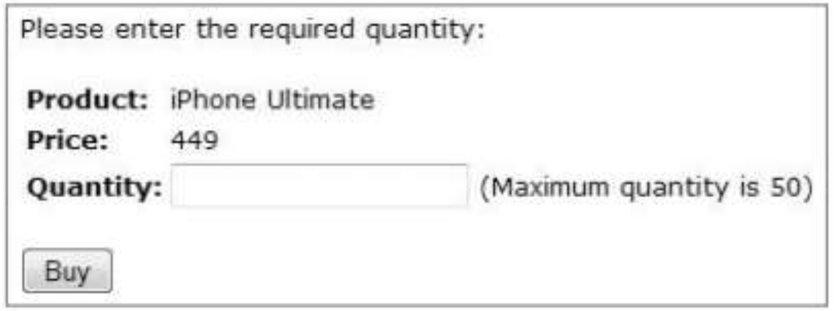
- Allowing clients to submit arbitrary input is a core security problem in web applications.
  - Users have full control of everything submitted from the client.
  - Can cause a range of problems including corrupting data stores, allowing unauthorized access to users and buffer overflows.
- In general, there are two ways client-side controls are used to restrict user input:
  - Transmitting data via the client using mechanisms that “prevent” user interaction. Examples include hidden form fields, disabled elements, referrer header, URL parameters, etc.
  - Controlling user input using measures that “restrict” user input. Examples include HTML form features, client-side scripts, etc.

Transmitting data via the client

➢ Example #1: Hidden Form Field

Code

```
<form method="post" action="Shop.aspx?prod=1">
Product: iPhone 5 <br/>
Price: 449 <br/>
Quantity: <input type="text" name="quantity"> (Maximum quantity is 50)
<br/>
<input type="hidden" name="price" value="449">
<input type="submit" value="Buy">
</form>
```



Please enter the required quantity:

Product:	iPhone Ultimate
Price:	449
Quantity:	(Maximum quantity is 50)

Request

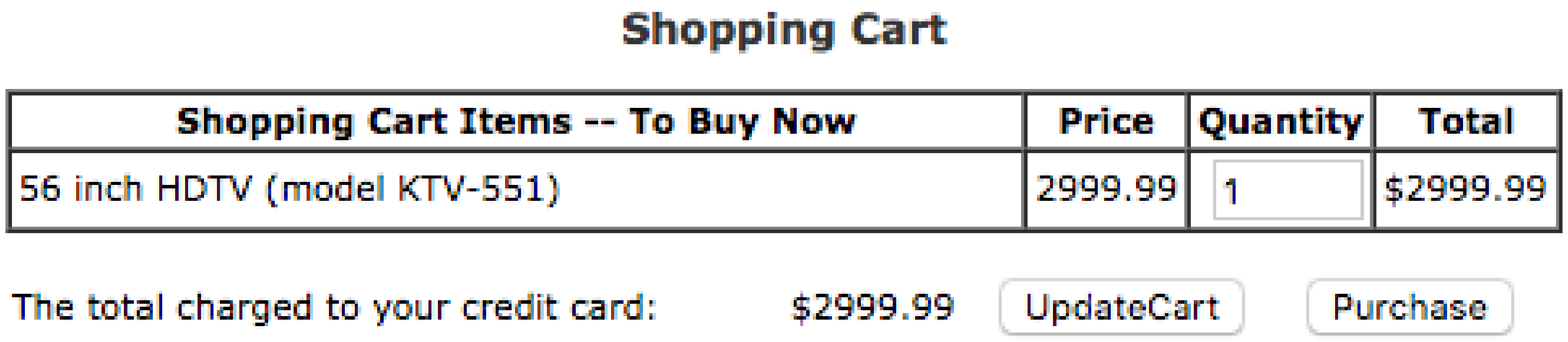
```
POST /shop/28/Shop.aspx?prod=1 HTTP/1.1
Host: mdsec.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 20

quantity=1&price=449
```

exercise #2: WEBGOAT

Exploit Hidden Fields

Try to purchase the HDTV for less than the purchase price, if you have not done so already.



Shopping Cart

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
56 inch HDTV (model KTV-551)	2999.99	1	$2999.99

The total charged to your credit card: $2999.99

authentication

- Authentication is a mechanism for validating a user.
- There are many authentication technologies:
    - HTML forms-based authentication
    - Multifactor authentication
    - Client SSL certificates and/or smartcards
    - etc
- In general, there are two factors that result in insecure authentication:
    - Design flaws in authentication mechanisms
    - Implementation flaws in authentication

Design flaws in authentication mechanisms

- Bad passwords*
- Brute-forcible logins
- Verbose Failure messages
- Vulnerable transmission of credentials
- Weaknesses in password change functionality
- Weaknesses in forgotten password functionality*
- etc.

Bad passwords

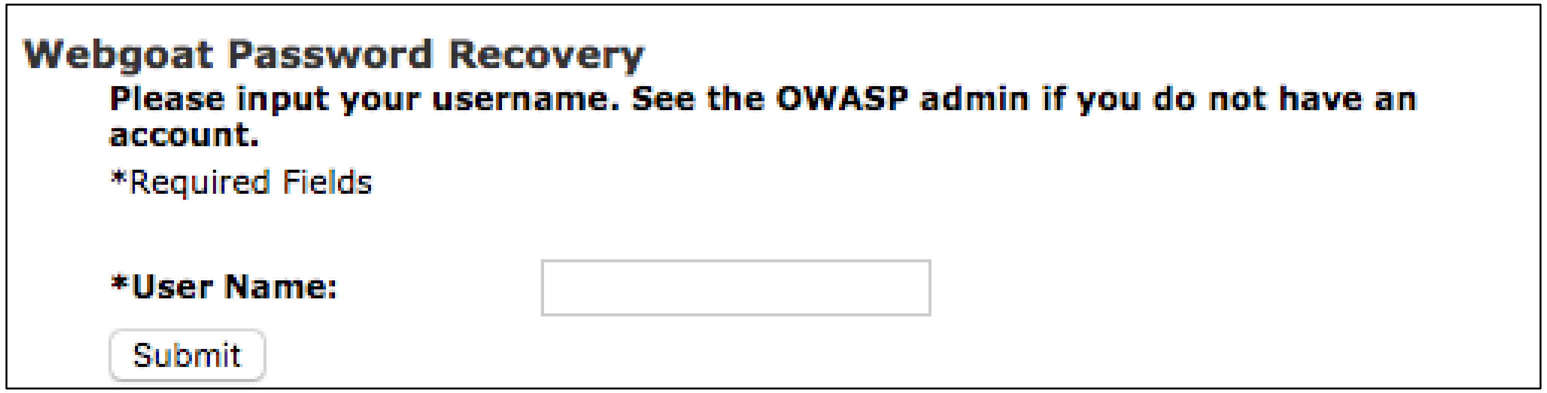
- Very short or blank passwords
- Common dictionary words or names
- The same as the username
- Still set to the default value

- Check the strength of your password: https://password.kaspersky.com/

exercise #5: WEBGOAT

Forgot Password

The goal is to retrieve the password of another user.



Webgoat Password Recovery

Please input your username. See the OWASP admin if you do not have an account.

*Required Fields

***User Name:**

IMPLEMENTATION FLAWS IN AUTHENTICATION

- Fail-open login mechanisms
- Defects in multistage login mechanisms*
  - Assumption that access to a later stage means that the user cleared prior stages.
  - Trusting client side data across stages
- Insecure storage of credentials
- etc.

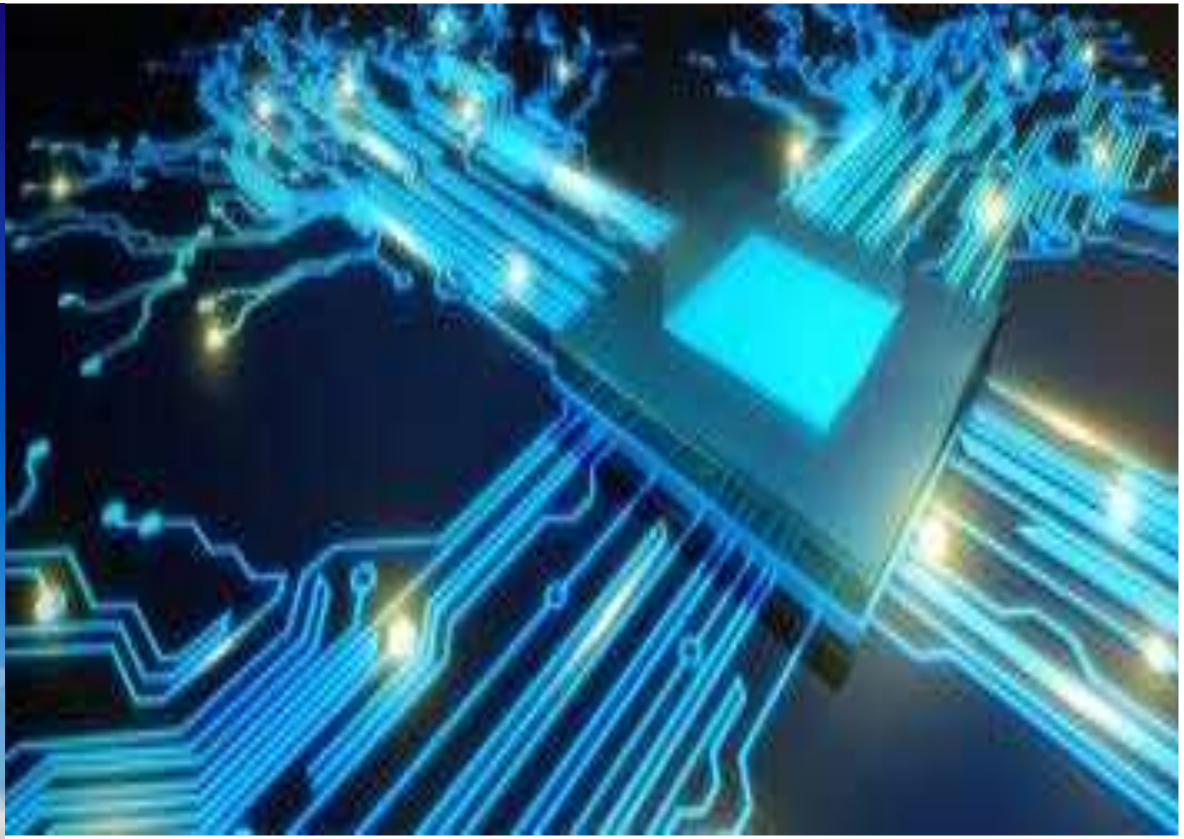
Attacking session management

- Understand the mechanism
- Test session tokens for meaning
- Test session tokens for predictability
- Check for session termination

- Altoro Mutual (testfire.net)
- Register Account (tutorialsninja.com)

Surprise Test-1

- Set-A
- Q1. Explain basic components of web security-3M
- Q2. List Top 10 Vulnerabilities in web security-3M
- Q3.With suitable example, explain white list validation.-4M
- Set-B
- Q1.Discuss any three web encoding schemes-3M
- Q2. List and explain the design and implementation flaws in authentication-3M
- Q3. With suitable example , explain blacklist validation.-4M



Web Security

Module-2
Web Application Authentication

PRESIDENCY UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

Two Factor Authentication

Two-factor authentication requires the use of two of the three authentication factors:

Something only the user:

1. Knows (e.g. password, PIN, secret answer)
2. Has (e.g. ATM card, mobile phone, hard token)
3. Is (e.g. biometric – iris, fingerprint, etc.)

- Two-factor authentication (2FA), sometimes referred to as two-step verification or dual-factor authentication, is a security process in which users provide two different authentication factors to verify themselves.
- 2FA is implemented to better protect both a user's credentials and the resources the user can access. Two-factor authentication provides a higher level of security than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically, a password or passcode. Two-factor authentication methods rely on a user providing a password as the first factor and a second, different factor -- usually either a security token or a biometric factor, such as a fingerprint or facial scan.

What are authentication factors?

- A knowledge factor is something the user knows, such as a password, a personal identification number (PIN) or some other type of shared secret.
- A possession factor is something the user has, such as an ID card, a security token, a cellphone, a mobile device or a smartphone app, to approve authentication requests.

- A biometric factor, also known as an inherence factor, is something inherent in the user's physical self. These may be personal attributes mapped from physical characteristics, such as fingerprints authenticated through a fingerprint reader. Other commonly used inherence factors include facial and voice recognition or behavioral biometrics, such as keystroke dynamics, gait or speech patterns.
- A location factor is usually denoted by the location from which an authentication attempt is being made. This can be enforced by limiting authentication attempts to specific devices in a particular location or by tracking the geographic source of an authentication attempt based on the source Internet Protocol address or some other geolocation information, such as Global Positioning System (GPS) data, derived from the user's mobile phone or other device.

- A time factor restricts user authentication to a specific time window in which logging on is permitted and restricts access to the system outside of that window.

How does two-factor authentication work?

1. The user is prompted to log in by the application or the website.
2. The user enters what they know -- usually, username and password. Then, the site's server finds a match and recognizes the user.
3. For processes that don't require passwords, the website generates a unique security key for the user. The authentication tool processes the key, and the site's server validates it.
4. The site then prompts the user to initiate the second login step. Although this step can take a number of forms, the user has to prove that they have something only they would have, such as biometrics, a security token, an ID card, a smartphone or other mobile device. This is the inherence or possession factor.
5. Then, the user may have to enter a one-time code that was generated during step four.
6. After providing both factors, the user is authenticated and granted access to the application or website.

Who Uses Two-Factor?

How Two-Factor Authentication Helps

- Two-factor authentication prevents attackers from accessing your account even if they obtain your username and password.

What is 3-Factor Authentication (3FA)?

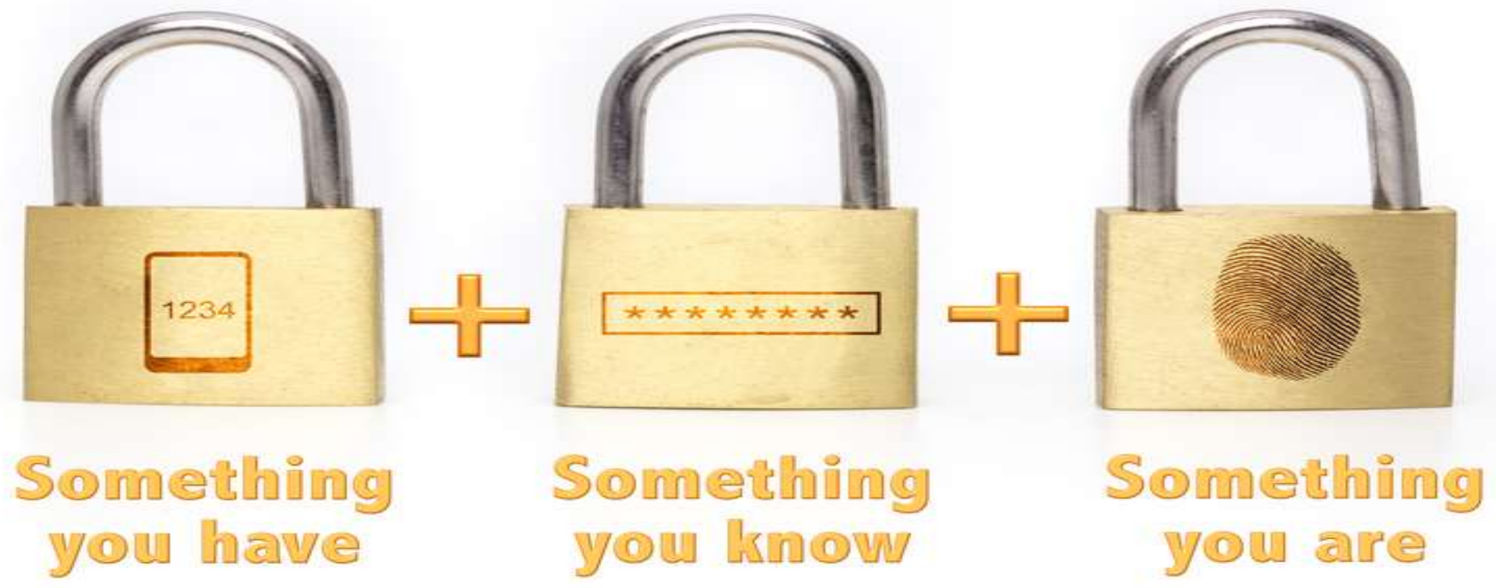
- Three-factor authentication is the use of a person’s live biometric data in addition to their security credentials and an authentication code sent by SMS.
- 3FA provides the highest possible level of user security for accessing accounts and making transactions.
- Biometric authentication may include a fingerprint, a retina scan, or a scan of the person’s entire face. This makes it almost impossible for hackers to bypass.

What is the benefit of 3-Factor Authentication?

- The key benefit of 3FA and 2FA is that they vastly reduce the chances of fraud and identity theft as a result of data breaches and stolen passwords. While it is theoretically possible to breach 2FA systems if the fraudster has control of a person’s mobile phone – the addition of a biometric check means that remote fraud is not possible.
- This allows companies to add a layer of confidence when authorizing access to customer accounts remotely.

- **What are the three factors that make up 3FA?**
- **Something you know**: This is a piece of information that a person knows – it could be a username, password, answer to a security question, or a touch gesture on a smart phone screen.
- **Something you have**: This is a physical object that a person has in their possession. It could be a smart card, a device like a phone or USB drive, or most commonly a token device that provides a pin code to access an application or online service.
- **Something you are**: This can include any unique physical facet of a person that can be used to verify their identity. It could be anything from a simple fingerprint to a facial scan or even DNA

Multi factor authentication



What is 4-Factor Authentication (4FA)?

- 4FA is simply a 3FA authentication solution with the addition of a check on a person's physical location, based on geometric data or network location. This could be used to block transactions outside of a prescribed territory, or even outside of a company's premises.

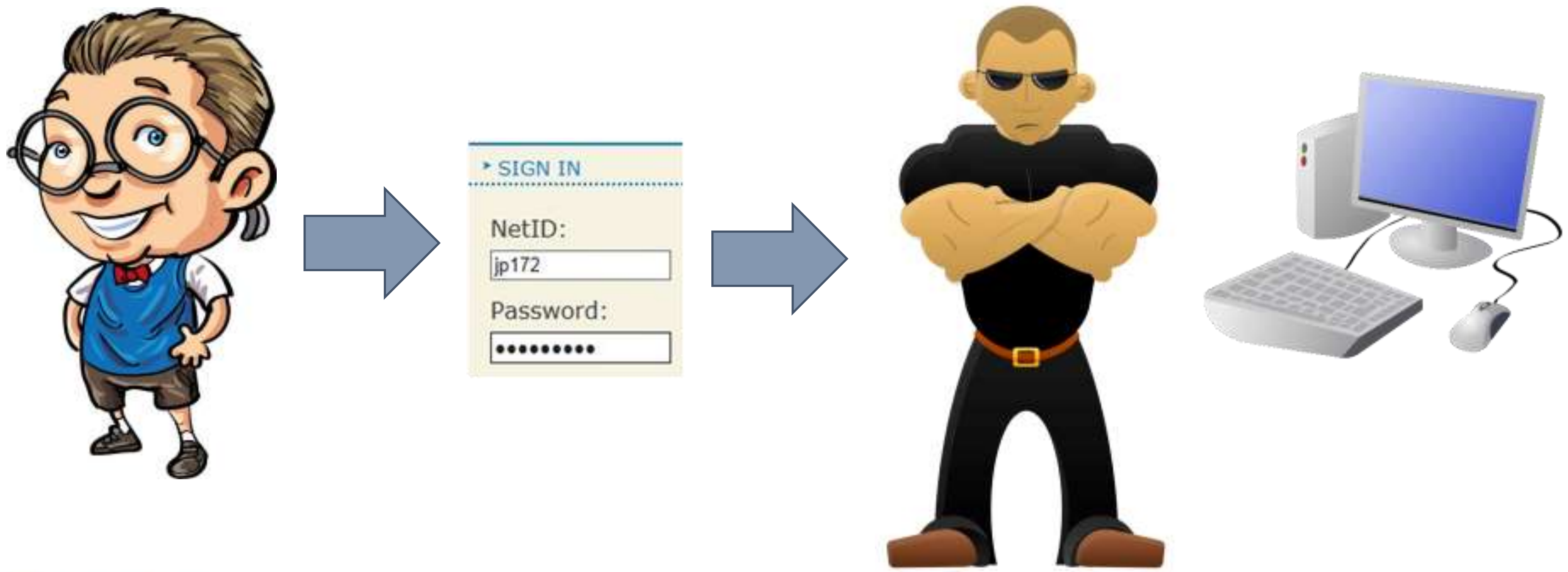
What is Multi-Factor Authentication??

Multi-factor authentication, also referred to as advanced or two-factor authentication, provides an additional layer of security when logging in or performing transactions online.

When logging in, a user is required to enter a password and also authenticate using a second factor, typically a phone or hardware token.

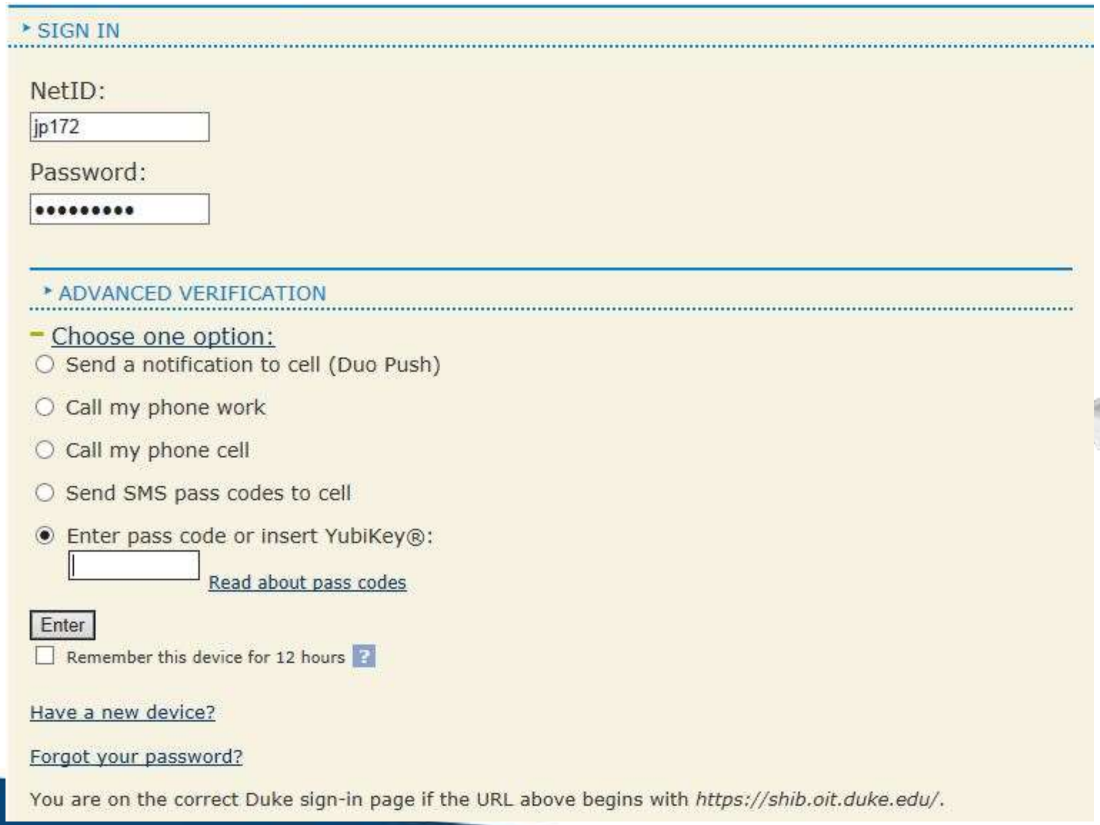
Multi-factor Authentication (MFA)

Multi-factor Authentication (MFA)



Mr. MFA

Multi-factor Authentication (MFA)



SIGN IN

NetID:
jp172

Password:
••••••••••

ADVANCED VERIFICATION

Choose one option:

- ○ Send a notification to cell (Duo Push)
- ○ Call my phone work
- ○ Call my phone cell
- ○ Send SMS pass codes to cell
- ● Enter pass code or insert YubiKey®:

Read about pass codes

Enter

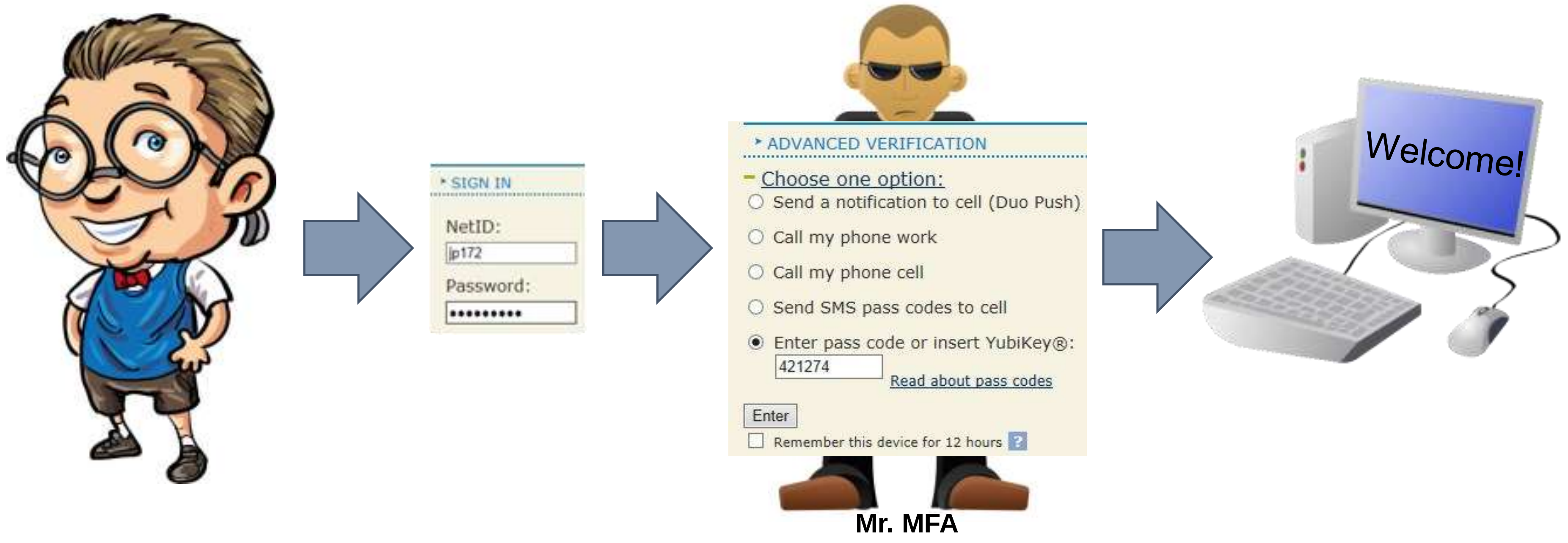
☐ Remember this device for 12 hours ?

Have a new device?

Forgot your password?

You are on the correct Duke sign-in page if the URL above begins with *https://shib.oit.duke.edu/*.

Multi-factor Authentication (MFA)



SIGN IN

NetID: jp172

Password: ••••••••••

ADVANCED VERIFICATION

Choose one option:

- ○ Send a notification to cell (Duo Push)
- ○ Call my phone work
- ○ Call my phone cell
- ○ Send SMS pass codes to cell
- ● Enter pass code or insert YubiKey®: 421274

Read about pass codes

Enter

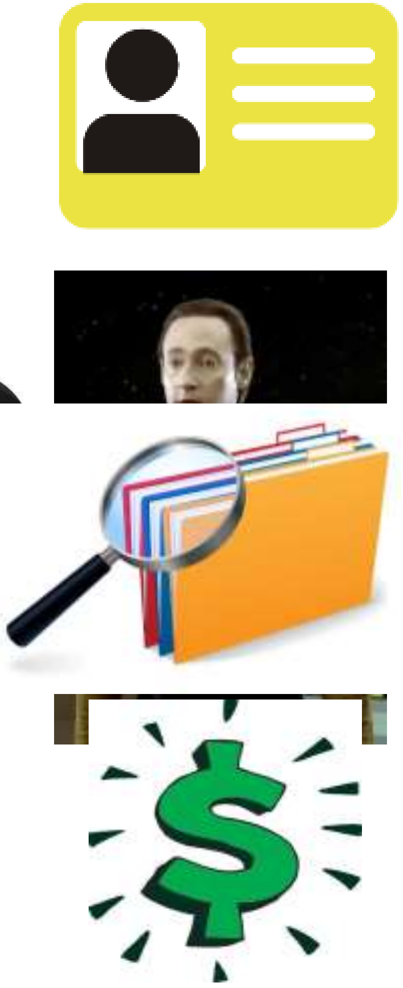
☐ Remember this device for 12 hours

Multi-factor Authentication (MFA)

Why do we need it?

- Prevent unauthorized users from logging into your account
- Protect your identity
- Protect your data
- Protect your money!

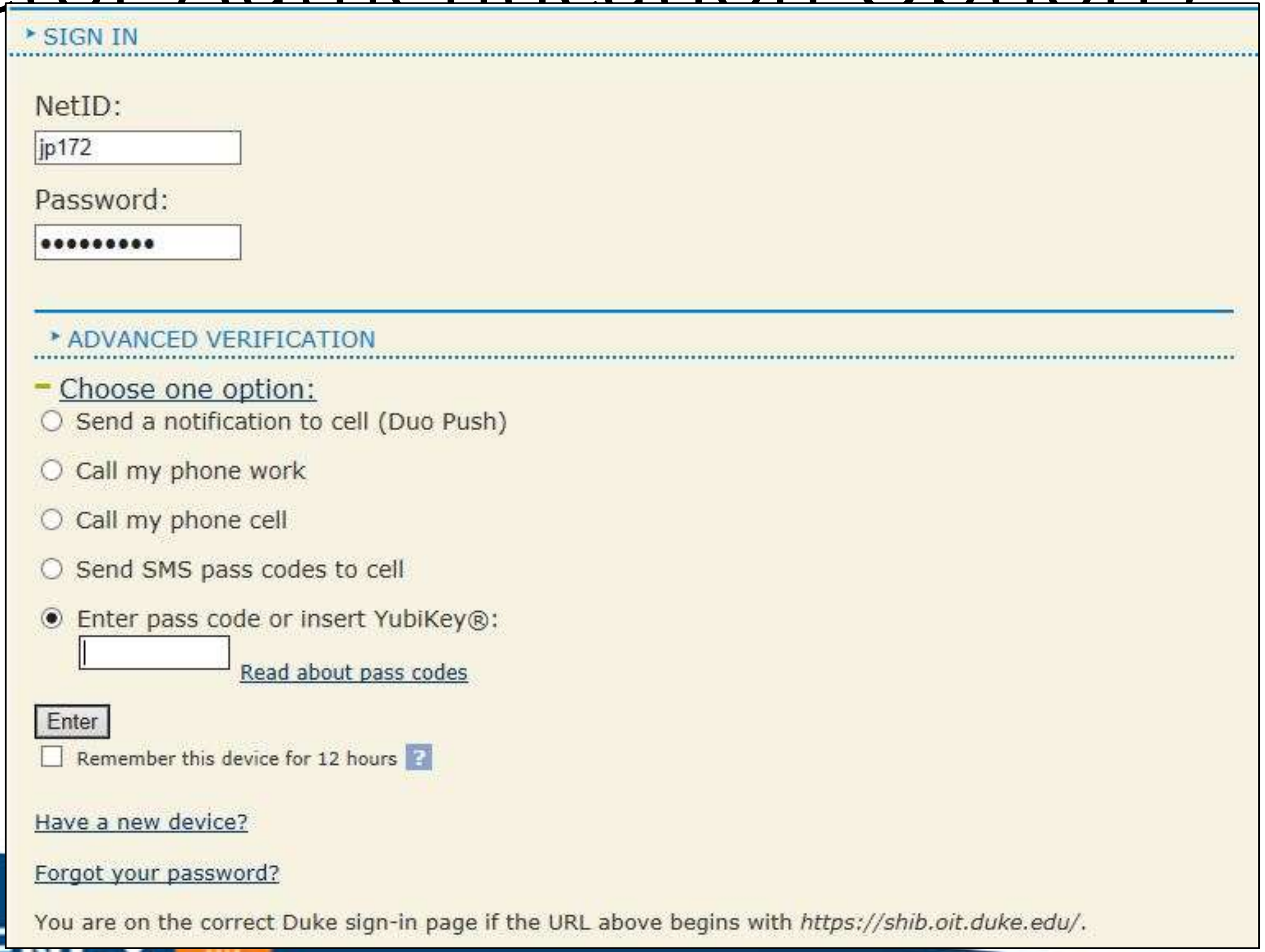
Multi-factor Authentication (MFA)



Ways of MFA Authentication

- Call your phone (desk or cell)
- Send text message with pass codes to cell
- Use the Duo Mobile app to create a pass code or send a notification to cell
- YubiKey authentication

Multi-factor Authentication Options



SIGN IN

NetID: jp172

Password: ••••••••••

ADVANCED VERIFICATION

Choose one option:

- Send a notification to cell (Duo Push)
- Call my phone work
- Call my phone cell
- Send SMS pass codes to cell
- [✓ Enter pass code or insert YubiKey®:]

Read about pass codes

Enter

Remember this device for 12 hours

Have a new device?

Forgot your password?

You are on the correct Duke sign-in page if the URL above begins with *https://shib.oit.duke.edu/*.

The Basic Authentication Scheme of HTTP

Access Restriction

- Sometimes, we want to restrict access to certain Web pages to certain *users*
- A *user* is identified by a *name* and a *password*
- Several mechanisms are used for controlling the access to pages on the Web
- A basic mechanism, provided by HTTP, is called "Basic Authentication Scheme"

Basic Authentication Scheme

- For each URL that the server wishes to restrict, a list of authorized users is maintained
- Using HTTP headers, the server declares that a the requested page is restricted (authentication is required)
- The client passes the name and password within a HTTP header
- The decision on which pages are restricted and to which users is implemented by the server (not a part of HTTP)

Basic Authentication Scheme (cont)

- The user's name and password need to be sent with each request for a protected resource
- When the server gets a request for a protected resource, it checks whether that request has the HTTP header

Authorization: Basic *username:password*

- *username:password* undergoes some non-secure encoding to allow for special characters
- If the name and password are accepted by the server (i.e., are those of a user that has the privilege to get the page), then the requested page is returned

Browser Cooperation

- Throughout the session, the browser stores the username and password and automatically sends the authorization header in either one of the following cases:
  - The requested resource is under the directory of the originally authenticated resource
  - The browser received 401 from the Web server and the WWW-Authenticate header has the same realm as the previous protected resource

Single Sign on (SSO) Authentication

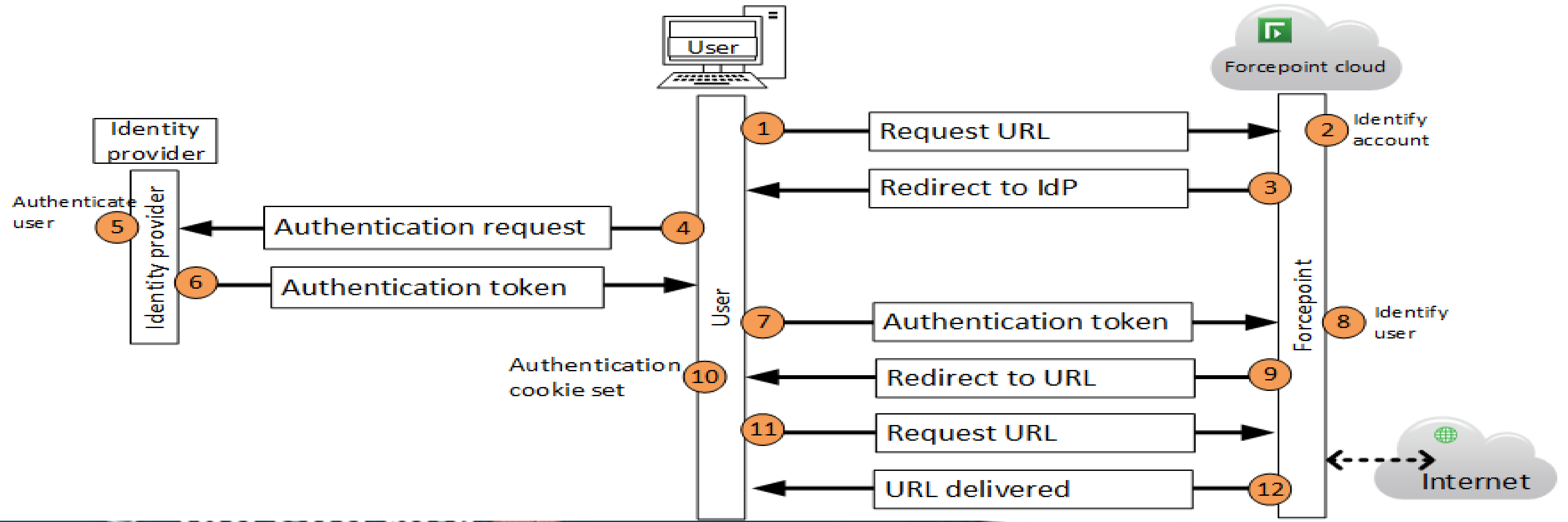
- Single sign-on (SSO) is a property of access control of multiple related, but independent software systems. With this property a user logs in once and gains access to all systems without being prompted to log in again at each of them. Conversely, single sign-off is the property whereby a single action of signing out terminates access to multiple software systems.

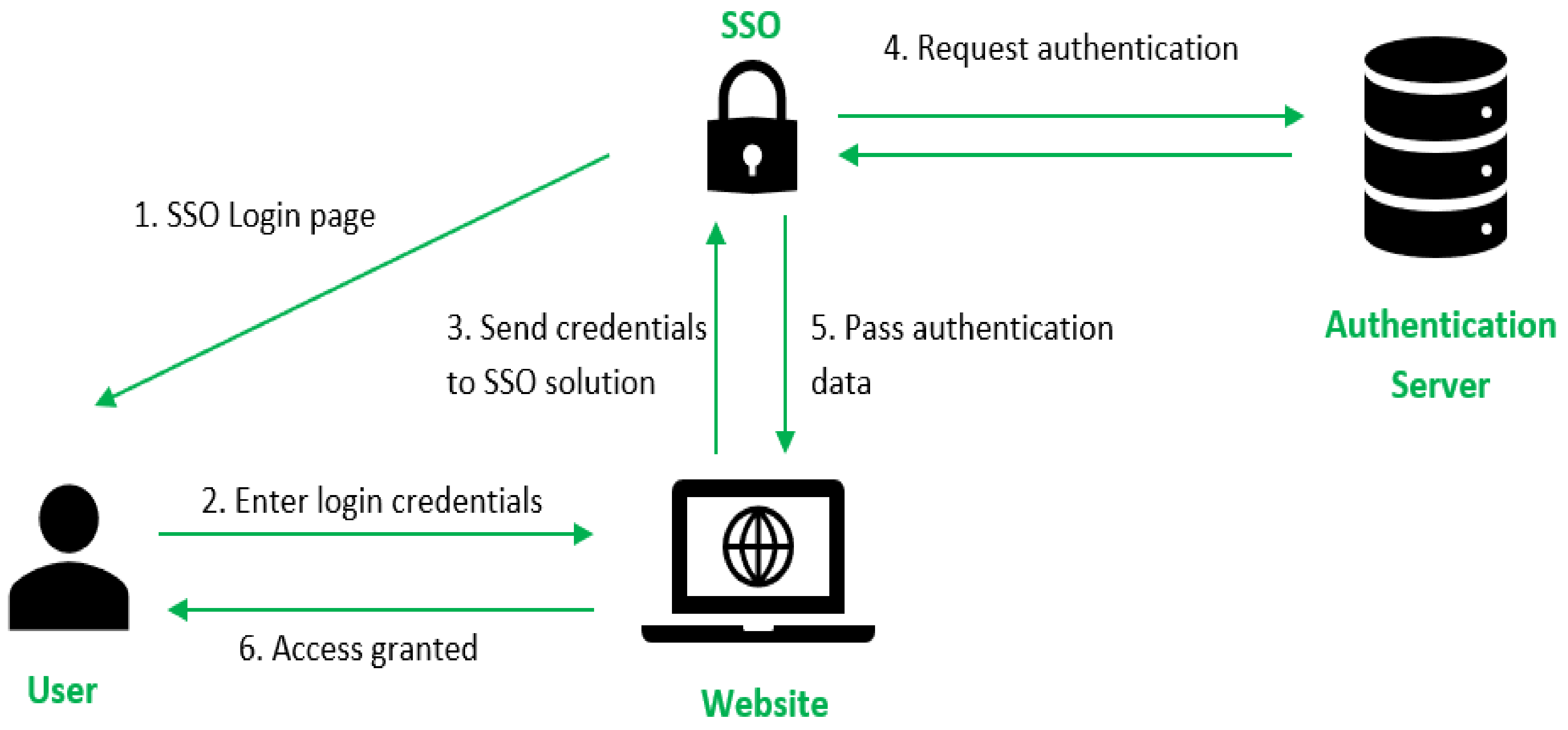
- As different applications and resources support different authentication mechanisms, single sign-on has to internally translate to and store different credentials compared to what is used for initial authentication.

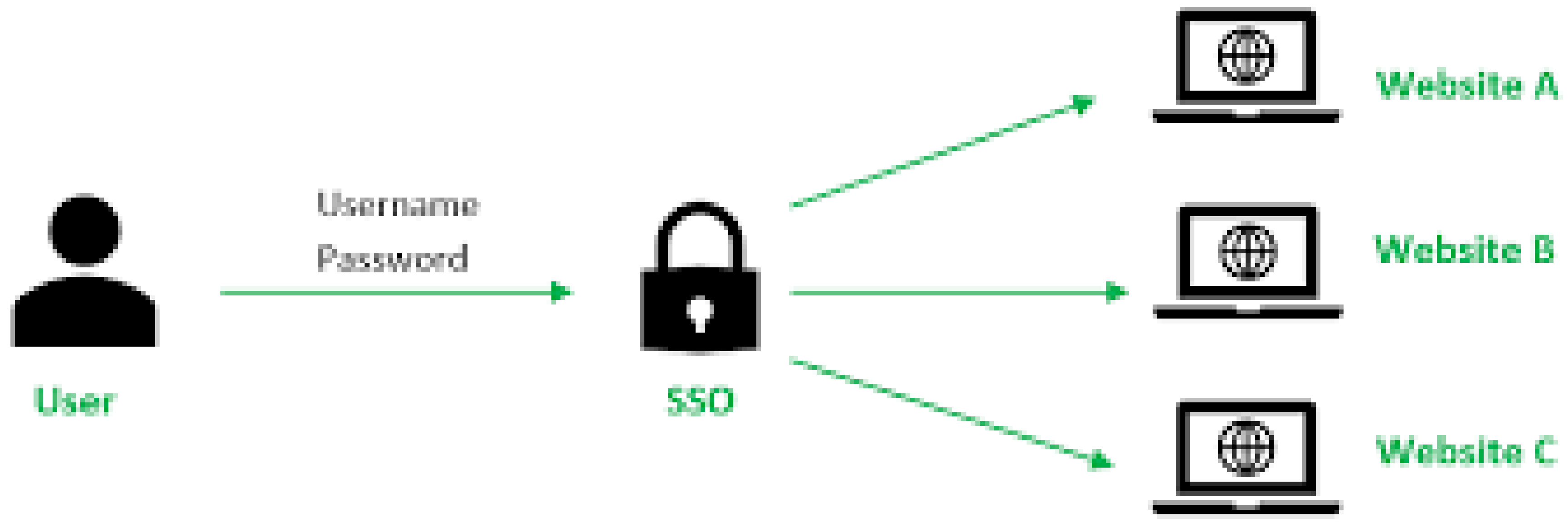
Drawback of Traditional Login systems

- Takes more time and reduces speed
- Difficult to remember the password and user name for all application
- It makes you feel irritated
- Increases the help desk workload
- Poor security habits

SSO work Flow




SSO
4. Request authentication
1. SSO Login page
3. Send credentials to SSO solution
5. Pass authentication data
Authentication Server
2. Enter login credentials
6. Access granted
User
Website


Username
Password
User
SSO
Website A
Website B
Website C

Advantages of SSO

- Improve security Habits
- Improves Identify Protection
- Increases Speed
- Relieves help desk workload

Demerits

- Single sign-on - Criticisms 1 The term enterprise reduced sign-on is preferred by some authors who believe single sign-on to be impossible in real use cases.
- Single sign-on - Criticisms 2 As single sign-on provides access to many resources once the user is initially authenticated ("keys to the castle") it increases the negative impact in case the credentials are available to other persons and misused. Therefore, single sign-on requires an increased focus on the protection of the user credentials, and should ideally be combined with strong authentication methods like smart cards and one-time password tokens.

- Single sign-on - Criticisms 3 Single sign-on also makes the authentication systems highly critical; a loss of their availability can result in denial of access to all systems unified under the SSO. SSO can thus be undesirable for systems to which access must be guaranteed at all times, such as security or plant-floor systems.

Validating Credentials

- The process of validating credentials is actually more complex than simply “determining whether or not the supplied password is associated with the supplied username.” There are several common ways to determine whether or not the correct password has been entered, and they depend on how the passwords are being stored in the back-end system

- **Comparison Logic in the Application with Plaintext Passwords**
- Using this approach, the application sends a request (for example, SQL query or LDAP query) to the back-end database to retrieve the record associated with the username. If the username does not exist, then the validation process fails. If a record is associated with the username, then the application-based logic compares the plaintext password provided by the user to the plaintext password from the retrieved record. If they match, then the credentials are valid.
- **Comparison Logic in the Database with Plaintext Passwords**
- This technique involves crafting a SQL query or an LDAP request to the back-end system with a conditional statement that asks the back end to return any records with matching fields that correspond to the supplied username and the supplied password. The database performs the username and password comparisons against the corresponding fields in this case. If any records are returned, then the application can safely assume that the user provided a legitimate set of credentials (unless SQL injection is used).

- **Comparison Logic in the Application with Hashed Passwords**
- With hashed passwords being compared within the application, a request is first made to the back-end datastore to retrieve the record associated with the user-supplied username. Because hashing is a one-way transformation, the user-supplied password must be hashed using the same algorithm that was used to hash the stored password in order for them to be compared. If the two hashed values are equal, then the user-supplied password was valid, and then authentication succeeds.

- **Comparison Logic in the Database with Hashed Passwords**
- Comparing hashed passwords on the back end also involves hashing the user-supplied password before it is sent to the datastore. The back-end system logic attempts to match the supplied username and the hashed form of the supplied password with a record in the system. If a match occurs, then the record is returned, and the application assumes that the credentials were valid (again, unless SQL injection is used)

Custom Authentication Systems

- Whenever a developer has coded their own application logic to process credentials, then a custom authentication system has been created
- **Web Authentication Process**
  - Using the browser, a user requests the login page from the web application, such as http://www.website.cxx/login.html.
  - The web application returns the login page to the browser.
  - The user enters their username and password into the input fields and submits the form, which now contains the credentials, to the web application. When the form is submitted to the web application, the browser will include the form fields as part of the HTTP POST request that is submitted.
  - Upon receiving the request, the web application parses the information in the HTTP message and extracts the values of the username and password.

- The application logic then queries a back-end data store (for example, a database or LDAP server) to determine whether or not the password entered is associated with the username entered.
- If the matching is unsuccessful, then the web application will send the user an error message along with the login page again.
- If the password is successfully matched to the associated username, then the application will establish a session with the user. Most applications establish a session with a user by generating a session ID value and returning it to the user.

- When the browser receives and parses the HTTP response, it will observe the Set-Cookie directive in the HTTP header and store the value of the session ID.
- Because the session ID value was set in a cookie, the browser will now automatically submit the session ID value alongside all requests made to the web application.
- This acts as a form of persistent authentication because the user no longer needs to enter their username and password to authenticate every request going to the application.
- Whenever the web application parses an HTTP request, it will see the session ID value and know that an existing session has already been established. It will use this session ID to authorize each request within the application logic.

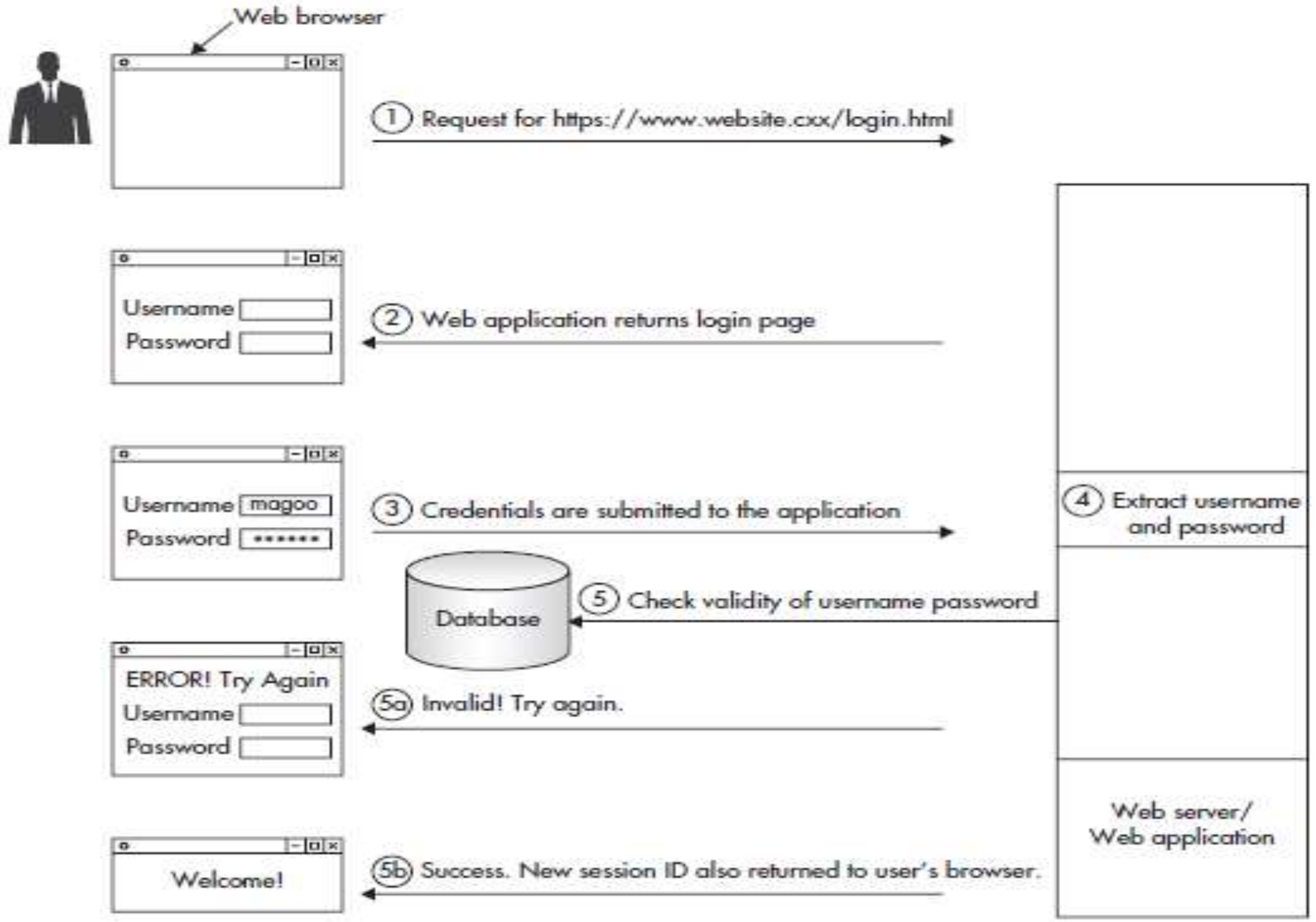

Web browser
1 Request for https://www.website.cxx/login.html
Username
Password
2 Web application returns login page
Username magoo
Password ******
3 Credentials are submitted to the application
4 Extract username and password
Database
5 Check validity of username password
ERROR! Try Again
Username
Password
5a Invalid! Try again.
Web server/ Web application
Welcome!
5b Success. New session ID also returned to user's browser.

Figure 3-9 Login with a username and password

Securing Password-Based Authentication

- Passwords are by far the most popular way of confirming your identity to a web application.
- **Attacks Against Passwords**
- Because the use of passwords is pervasive as an authentication factor in web applications, they are also a very popular target of attackers. All attacks against passwords essentially boil down to repeatedly guessing at the password in an attempt to determine the plaintext value of the password. When attempting to guess a password, you can attempt it either against the live system (online) or against the hashed or encrypted password values (offline). Common attack variations include:
- Dictionary attack
- Brute-force attack
- Precomputed dictionary attack
- Rubber-hose attack

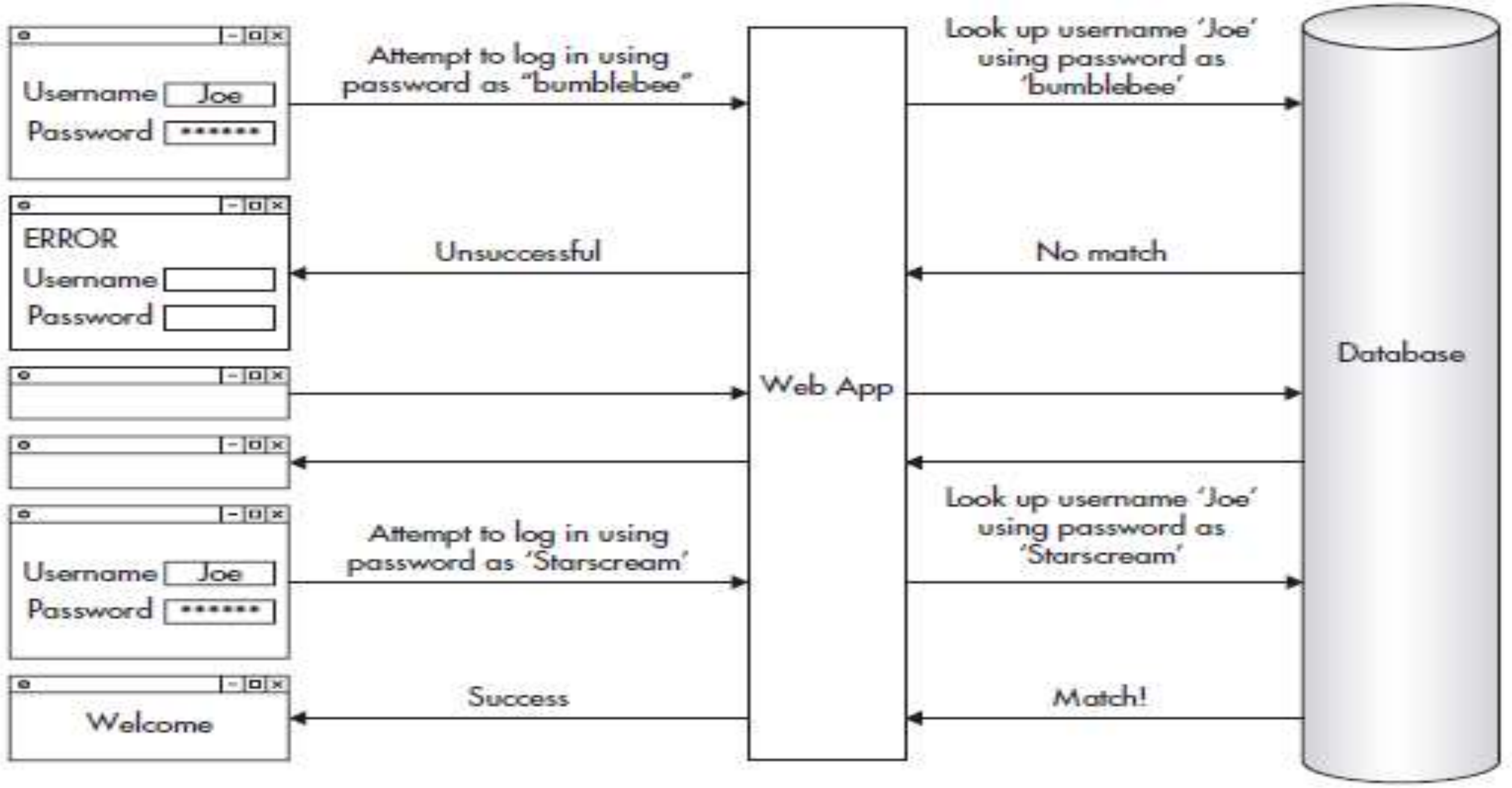
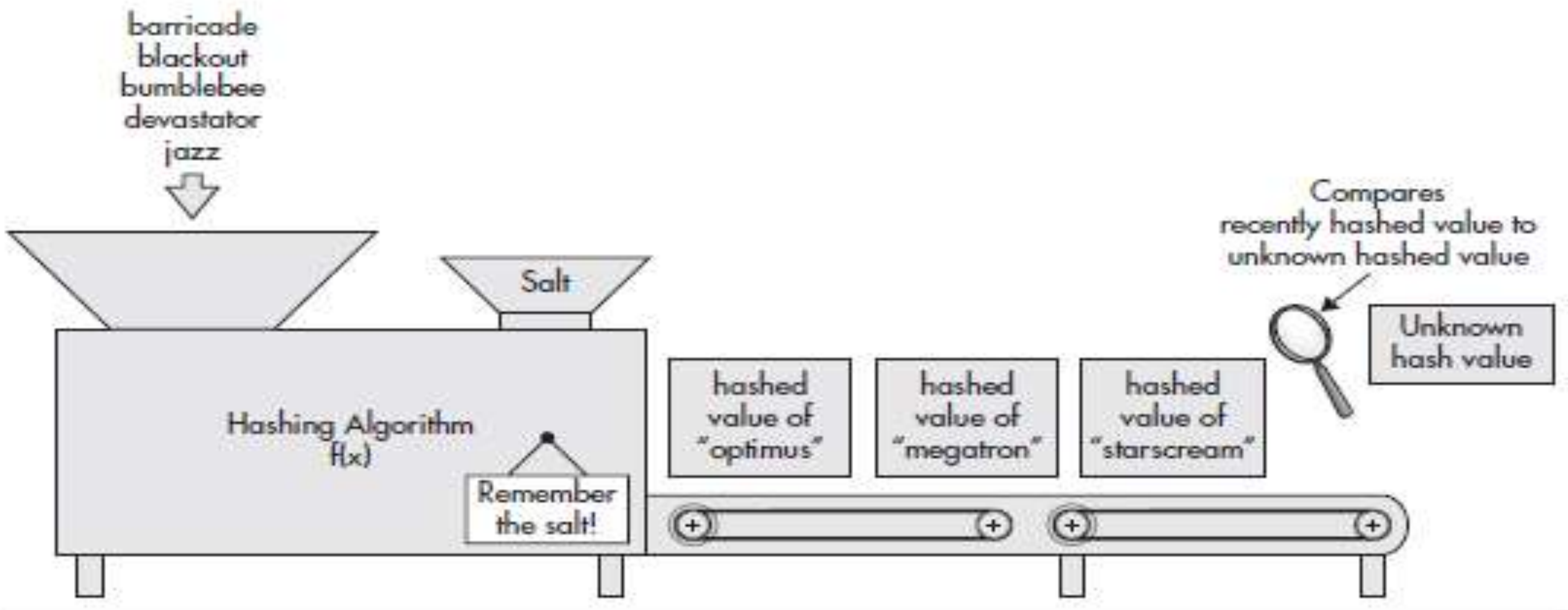

Username
Joe
Password
******
Attempt to log in using password as "bumblebee"
Look up username 'Joe' using password as 'bumblebee'
ERROR
Username
Password
Unsuccessful
No match
Web App
Database
Attempt to log in using password as 'Starscream'
Look up username 'Joe' using password as 'Starscream'
Username
Joe
Password
******
Welcome
Success
Match!

Figure 3-10 Online attacks are slow.



barricade
blackout
bumblebee
devastator
jazz
Salt
Hashing Algorithm f(x)
Remember the salt!
hashed value of "optimus"
hashed value of "megatron"
hashed value of "starscream"
Compares recently hashed value to unknown hashed value
Unknown hash value

Figure 3-11 Offline attacks are much faster.

- **Dictionary Attack**
- A *dictionary attack* is based on the premise that users have a predilection for selecting passwords based on real words that can be found in dictionaries. The most likely candidate words can be collected in a list that is referred to as a "dictionary." In some cases, real dictionaries may be employed, and creating permutations, such as appending a digit or special character at the end of each word, may be done as well. With a dictionary in hand, an attacker will successively attempt each password until they have successfully guessed the password or the list is exhausted.
- **Brute-Force Attack**
- A *brute-force attack* is also referred to as an exhaustive key search, and in theory involves attempting every single possible key. In reality, limits are usually placed on a brute-force attack based on length and character set (for example, alphabet, digits, whitespace, special characters).

- **Precomputed Dictionary Attack**
- **Rubber-Hose Attack**
- A “rubber-hose” attack refers to instances in which an intruder uses any sort of physical coercion to extract the value of the password from an individual.

- **The Importance of Password Complexity**
- **Password Best Practices**
  - **Require Minimum Password Length**
  - **Enforce Minimum Password Complexity**
  - **Rotate Passwords**
  - **Require Password Uniqueness**
  - **Password Cannot Equal Username**
  - **Allow Accounts to Be Disabled**
  - **Properly Store Passwords**
    - **Don’t Store in Plaintext**
    - **Don’t Encrypt**
    - **Use a Strong Hash**
    - **Multiple Rounds of Hashing**

Securing Web Authentication Mechanisms

- Secure the Transmission
- Allow Account Lockout
- Allow Accounts to Be Disabled
- No Default Accounts
- Don't Hard-Code Credentials
- Avoid Remember Me (Stay Signed In)

Authorization

- Authorization is the process of determining whether a subject has sufficient permission to perform a given operation against a target resource.
- **Authorization Fundamentals**
- AuthZ is the process of deciding whether a user can perform a certain action.

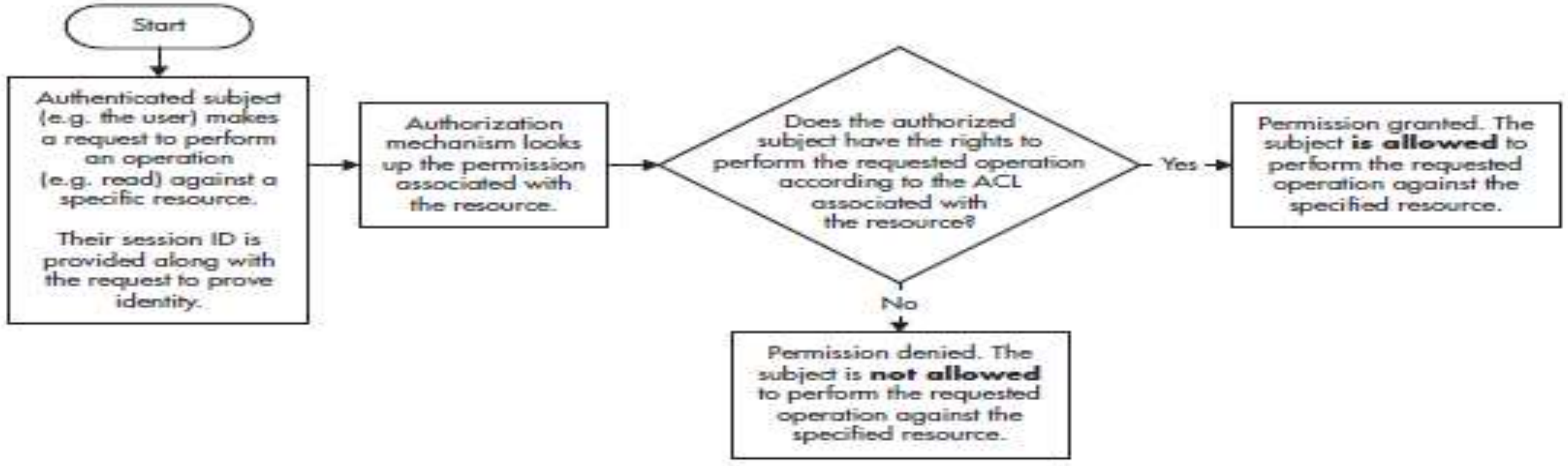

Start
Authenticated subject (e.g. the user) makes a request to perform an operation (e.g. read) against a specific resource.
Their session ID is provided along with the request to prove identity.
Authorization mechanism looks up the permission associated with the resource.
Does the authorized subject have the rights to perform the requested operation according to the ACL associated with the resource?
Yes
Permission granted. The subject is allowed to perform the requested operation against the specified resource.
No
Permission denied. The subject is not allowed to perform the requested operation against the specified resource.

Figure 4-1 A simple model of authorization

Authorization Goals

We authorize for the following reasons:

- To ensure that users can only perform actions within their privilege level.
- To control access to protected resources using criteria based on a user’s role or privilege level.
- To mitigate privilege escalation attacks, such as might enable a user to access administrative functions while logged on as a non-administrative user or potentially even an anonymous guest user.

- **Subjects**
- a subject is the thing requesting access to protected resources
- In the web application world, a subject is commonly any of
    - An actual human user accessing the web application
    - A web application accessing another web service
    - A web application accessing a back-end database
    - Any one of the web services or back-end databases accessing their local operating systems
    - Another computer system or host
    - Essentially, anything that has been assigned an identity

- **Resources**
    - Resources are simply the protected objects, either data or functionality, that the subject is attempting to access. Practically anything can be a resource, such as a protected file that a user is trying to open, a table or a record in a database, programs and other units of software, or even connectivity to networks

Access Control

- **Discretionary Access Control (DAC)** In DAC, access control is left to the discretion of the owner of an object or other resource. Although access is primarily controlled by the object owners, there are system-wide or application-wide access control rules such as the ability to debug a process running under a different account or the ability to load kernel code. Such rules can typically be overridden by the owners of objects and resources.
- **Mandatory Access Control (MAC)** In MAC, access control is determined by the system, or by system administrators, rather than object owners. Some web applications use this model because of its stronger limits on what can potentially happen within the application, as well as the simplification of design and user interface that comes with not needing to provide users with a means to manage permissions."

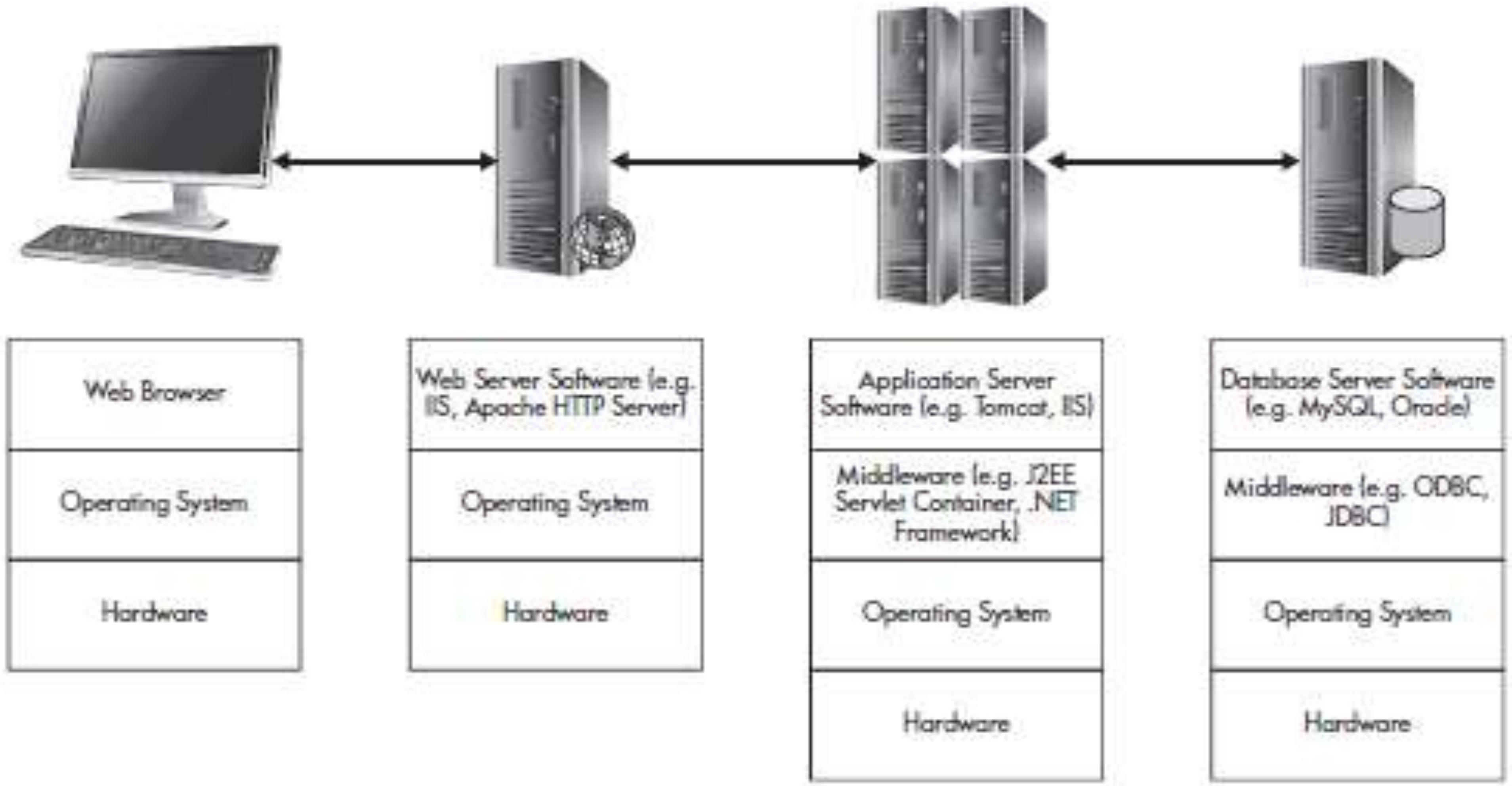
- **Role-Based Access Control (RBAC)** ❑ This is another nondiscretionary model, like MAC, but which implements access control by means of *roles*, Access determinations are still made by the system (or by system administrators), but are made in the context of a more general framework. Administrators can, if necessary, define new roles and assign uses to them, which is often not possible in strict MAC-oriented systems.
- **Hybrid systems** ❑ Nothing says you can't mix and match these three different access control models, and indeed, many web applications do just this. The social media site Facebook, for example, mixes RBAC and DAC; roles such as user and administrator codify broad permission policies, but the application also incorporates elements of discretionary access by allowing users to control who can see what information on their "wall."

Types of Permissions

- Read Access
- Write Access
- Execute Access

Authorization Layers

- Authorization is not a one-time thing. Authorization should happen at many points and many times within a web application. These points come at certain common boundaries that exist in most web applications, forming “layers” that can be thought about, designed, and implemented in a holistic fashion


Web Browser
Operating System
Hardware
Web Server Software (e.g. IIS, Apache HTTP Server)
Operating System
Hardware
Application Server Software (e.g. Tomcat, IIS)
Middleware (e.g. J2EE Servlet Container, .NET Framework)
Operating System
Hardware
Database Server Software (e.g. MySQL, Oracle)
Middleware (e.g. ODBC, JDBC)
Operating System
Hardware

gure 4-2 Checkpoints where AuthZ occurs

- Web client
- Front-end web server
- Back-end application servers
- Back-end database

- **User layer** -The topmost layer on a client system. For the web client machine, the user is the actual human being, as proxied by the user interface software, which translates mouse moves and keyboard clicks into commands.
- **Application layer** -For a front-end web server, an application server, or a back-end database, the topmost layer is likely the web server software, application server software, or database server software, respectively.
- **Middleware layer** Whatever components fit the definition of “the thing on top of which this part of the web application is running.” For the web-server layer, the middleware is likely to be a web server such as Microsoft’s Internet Information Server (IIS) or the open-source Apache server

- **Operating system layer**-The low-level software that manages the physical resources of the computer: memory, disk space, CPU time, network bandwidth, and so on. In most applications, access to the file system is also mediated by the operating system layer.
- **Hardware layer** -The physical material of the computer. Although this is the layer where the actions of a user are “made real,” the hardware layer doesn’t interact strongly with a typical web application’s authorization system.

Securing Web Application Authorization

- **Web Server Layer**
- **IP Address Blacklisting**
  - At the front-line boundary of your web application, you can check the source IP address on the incoming request, and reject it if necessary. For example, you might deny requests from IP addresses that have tried to attack your server previously with a distributed denial-of-service attack (DDoS attack). For such requests, the application might return a "403 Forbidden" HTTP response, or might simply ignore the request without sending any response at all (also known as "dropping" the request).
- **IP Address Whitelisting**
  - Conversely, if your application is such that you know in advance the exact IP addresses or range of addresses that should ever be allowed to access the application, you can simply watch for those source IP numbers and reject everything else.
- **URL Authorization**
  - Some web servers (including Microsoft IIS, Apache, and nearly all Java web servers) and some web application frameworks provide facilities for limiting access to specific URLs

- `<location path="topsecret.aspx"> <system.webServer>`
- `<security>`
- ` <authorization>`
- `<add accessType="Allow" users="TopSecretUser" /> </authorization>`
- `</security>`
- `</system.webServer>`
- `</location>`

- **Operating System Authorization**
  - The operating system’s job is to manage (and thus, protect) the computer’s physical resources as well as logical resources exposed by the OS itself. Because the web server and/or web application must run in some context within the OS, and because such contexts are typically integrated into the same overall security framework as regular users, usually there is some kind of user account associated with your server and application

Application Server Layer

- **Application Compartmentalization**
- **Servlet and App Server Restrictions**
  - `<security-constraint> <web-resource-collection> <web-resource-name>secure</web-resource-name> <url-pattern>/secure/*</url-pattern> <http-method>GET</http-method> <http-method>POST</http-method> </web-resource-collection> <auth-constraint> <role-name>admin</role-name> </auth-constraint></security-constraint>`

Application Server Code

Use a built-in framework

Microsoft's .NET platform has built-in security modules for role-based security.

The ASP.NET platform has a Membership framework that is primarily intended for form-based AuthN, but does allow you to validate and manage users through prebuilt libraries.

ASP.NET also provides role-based management for AuthZ.

Use an existing, open plug-in AuthZ module

There are a number of AuthZ modules that plug in to various web development frameworks in order to provide authorization features not present (or deemed insufficient) within the frameworks themselves. These include OAuth, BBAuth, AuthSub, and others.

Develop a custom framework

Designing and developing a proper authorization framework is a significant undertaking.

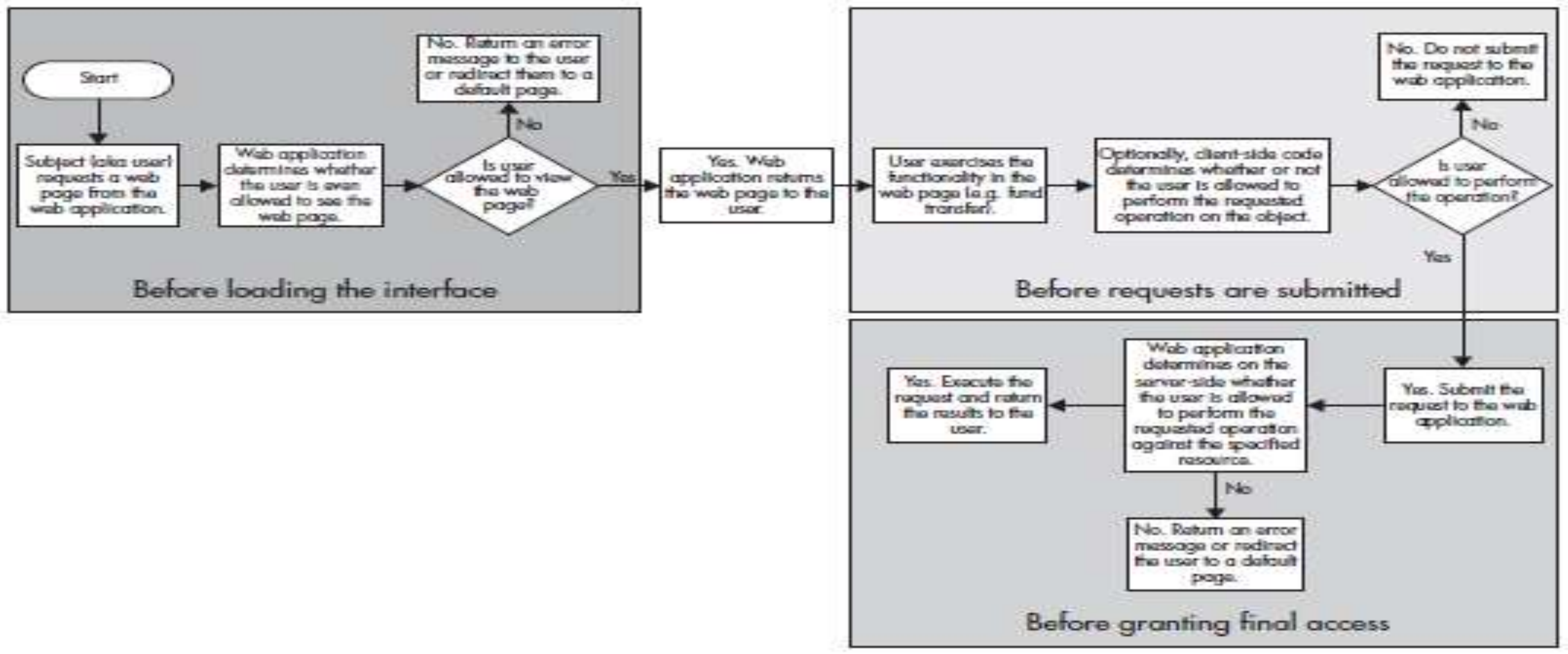
Code Access Security

- CAS is typically used to prevent uploaded or downloaded components from performing dangerous actions

Database Server Layer

- Wrap every request to create, read, update, or delete data within properly parameterized stored procedures. This means that you will not have your web application generating SQL query strings and submitting them to the database directly.
- Map all the interactions between the application and the database to a set of database user accounts, and reduce the permissions those accounts have to the bare minimum.

Custom Authorization Mechanisms



Start
Subject (aka user) requests a web page from the web application.
Web application determines whether the user is even allowed to see the web page.
Is user allowed to view the web page?
No
No. Return an error message to the user or redirect them to a default page.
Yes
Before loading the interface
Yes. Web application returns the web page to the user.
User exercises the functionality in the web page (e.g. fund transfer).
Optionally, client-side code determines whether or not the user is allowed to perform the requested operation on the object.
Is user allowed to perform the operation?
No
No. Do not submit the request to the web application.
Yes
Before requests are submitted
Yes. Submit the request to the web application.
Web application determines on the server-side whether the user is allowed to perform the requested operation against the specified resource.
Yes. Execute the request and return the results to the user.
No
No. Return an error message or redirect the user to a default page.
Before granting final access

Figure 4-3 A 3x3 authorization system in action

- **Users/subjects** Any entity that’s making a request against a resource.
- **Operations** The functionality resources in your web application; the specific actions that subjects can take.
- **Objects** These are the resources managed by your web application, the underlying things, such as data, that your web application cares about.

Web Authorization Best Practices

- **Failing Closed**
  - Design your application such that if anything fails, it will fail into a secure state. The result of bad input, a bug raising an unexpected exception within your application code, and so on,
- **Operating with Least Privilege**
  - Operating with least privilege means designing your web application to work with the fewest possible rights and permissions while still allowing the application to accomplish what it needs to do.
- **Separating Duties**
  - Business user accounts should not be administrators, and vice versa. Regular users and administrators have very different jobs, and perform very different kinds of actions within the application.
- **Defining Strong Policies**
  - Even the best-designed authorization system will fail if it is not used correctly. Define strong permission policies up front.

- **Keeping Accounts Unique**
  - Don’t let users share accounts. Ever. Make sure every user has a unique account—or more than one, as in the case of administrators who sometimes also need to be users. If you let users share accounts, then the process of authorization becomes virtually impossible to do rigorously.
- **Authorizing on Every Request**
  - A good practice is for the web application to perform a final authorization check immediately before performing any action on the user’s behalf. This necessitates that the application performs at least one authorization check on every request that comes into the system
- **Centralizing the Authorization Mechanism**
  - Design a clean authorization architecture that can be easily shared across the entire application. A common mistake is to craft some authentication code for use in one spot in an application, then copy that same code to everywhere else in the application that needs it.
- **Minimizing Custom Authorization Code**
  - writing custom authorization code is difficult, expensive, and prone to give you worse results than if you used a reliable, well-tested, off-the-shelf authorization module. Use it if you need to, but minimize the amount of custom AuthZ code you write

- **Protecting Static Resources**
  - Be very cautious about trusting the host operating system's file system with anything. Ideally, your application should generate any highly sensitive content dynamically, at the moment of request.
- **Avoiding Insecure Client-Side Authorization Tokens**
  - Don't use insecure client-side tokens—that is, data that is stored at the web client and submitted with each request—to hold authorization data of any kind. It is often tempting for web developers to take this route, because the mechanisms for client-side token storage (cookies and the like) are incredibly simple to use and thus avoid the complexity of implementing server-side sessions and session management.
- **Using Server-Side Authorization**
  - Authorization checks should happen on the server and never solely on the client
- **Mistrusting Everybody**
  - Never trust a user to provide an accurate accounting of that user's roles or permissions. A secure server is a paranoid server. Servers should rely on only trusted data sources

Attacks Against Authorization

- **Forceful Browsing**
- **Input Authorization/Parameter Tampering**
- **HTTP Header Manipulation**
- **Cross-Site Request Forgery (CSRF)**

Module-3 Session Management

OVERVIEW

- **The Need**
- Weaknesses of Token Generation
- Weaknesses of Session Token Handling
- Securing Session Management
- Summary

THE NEED

- Reminder: HTTP Protocol is stateless
- Majority of web "sites" are actually web applications
- The session management mechanism is a fundamental security component in most web applications

SESSION MANAGEMENT VULNERABILITIES

- HTTPOnly Flag Not Set
- Secure Flag Not Set
- Session Porting Permitted
- Persistent Cookie
- Cookieless Sessions in Use
- Session Token Content Weaknesses
- Session Token Not Regenerated on Login
- Cookie Domain and Path not Restricted

WHAT HAPPENS IF THE ATTACKER SUCCEEDS?

1. Attacker can bypass authentication
2. Attacker can masquerade as a legitimate user
3. Attacker can compromise an administrative user or own the entire application

The list goes on…

OVERVIEW

- The Need
- **Weaknesses of Token Generation**
- Weaknesses of Session Token Handling
- Securing Session Management
- Summary

WEAKNESSES IN TOKEN GENERATION

- **Meaningful Tokens**
- Predictable Tokens
- Concealed Sequences
- **Weak Random Number Generation**

MEANINGFUL TOKENS

```
757365723d6461663b6170703d61646d696e3b646174653d30312f31322f3131
```

- Tokens containing account username, first or last names, date/time stamp, client IP, etc.
- Attackers can use a hexadecimal decoder to reveal the session token easily
- Examples of online decoders include: www.string-functions.com or www.converstring.com

```
User=daf;app=admin;date=10/09/11
```

WEAK RANDOM NUMBER GENERATION

- Predictable pseudorandom generator used
- After a visual inspection, a more rigorous approach to test the quality of randomness is necessary
- Burp Sequencer is a tool that will test randomness of web application tokens
- Obtaining a sample size of 20,000 tokens, will achieve compliance with FIPS test for randomness

OVERVIEW

- The Need
- Weaknesses of Token Generation
- **Weaknesses of Session Token Handling**
- Securing Session Management
- Summary

WEAKNESSES IN TOKEN HANDLING

Disclosure of tokens on network

- Occurs when tokens are transmitted in an unencrypted form
- For example, a site that uses HTTPS to protect login, but reverts to HTTP for the remainder of the user session

Disclosure of Tokens in System Logs

- An application may use the URL query string as a mechanism for transmitting tokens
- For example, google search inurl:jsessionid will produce a list of applications that transmit the Java platform session token

DO’S & DON’TS

- Tokens should only be transmitted over HTTPS
- Tokens should never be transmitted in the URL
- Visibility of session token for administrative or diagnostic purposes should be limited
- Logout functionality should be implemented
- Session expiration should be implemented
- Concurrent logins should be prevented
- Restrict domain and path scope of application should be restricted as much as possible

OVERVIEW

- The Need
- Weaknesses of Token Generation
- Weaknesses of Session Token Handling
- **Securing Session Management**
- Summary

HOW CAN WE ENSURE SECURE SESSIONS?

SECURING SESSION MANAGEMENT

- Appears simple…as generating strong tokens and providing token protection throughout life cycle

- But…requires developers to have an in-depth understanding of protocols, algorithms, and black-hat community attacks

OVERVIEW

- The Need
- Weaknesses of Token Generation
- Weaknesses of Session Token Handling
- Securing Session Management
- **Summary**

Cross-Site Scripting

- More web sites are vulnerable to *cross-site scripting* (or *XSS*) attacks than any other type of web application attack. According to statistics from the Web Application Security Consortium (WASC), almost 40 percent of all web applications tested for security flaws have at least one XSS vulnerability.
- cross-site scripting is a vulnerability that allows an attacker to add his own script code to a vulnerable web application’s pages. When a user visits an “infected” page in the application—sometimes by following a specially crafted link in an e-mail message, but sometimes just by browsing the web site