# Session and Cookie

Building Modern Web Applications - VSP2024

**Karthik Pattabiraman**
**Abraham Chan**
**Mohsen Salehi**

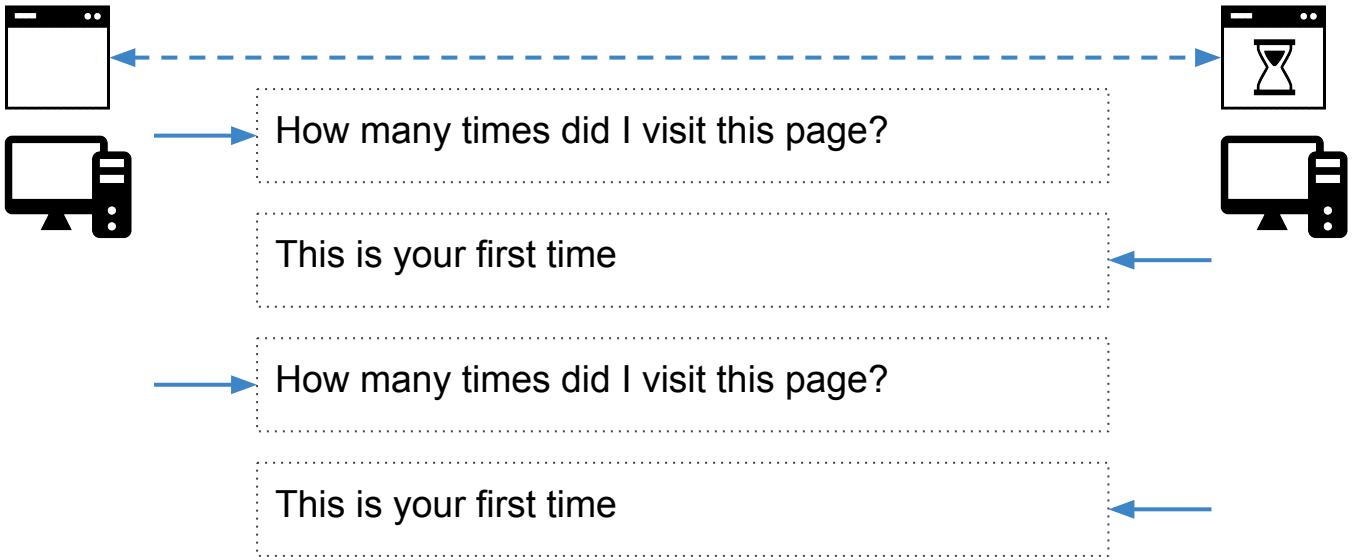
# Session

1. **Session**
2. Cookie
3. Web Security

## 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**

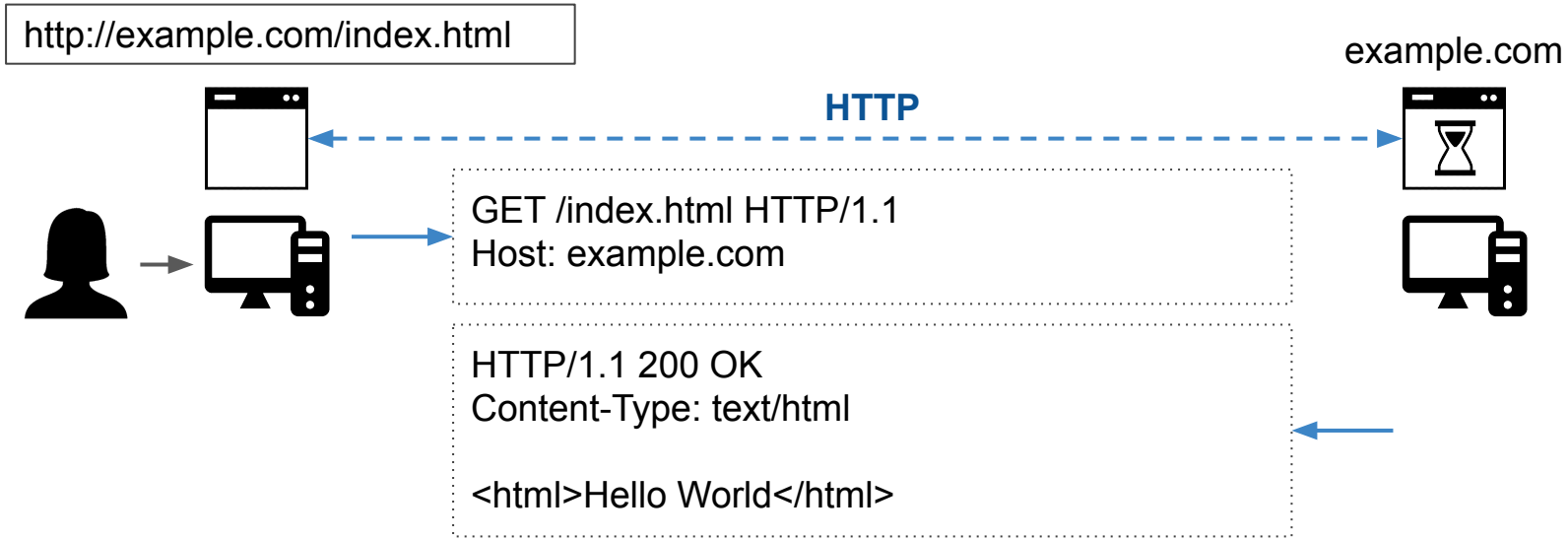
# Session: What is it?



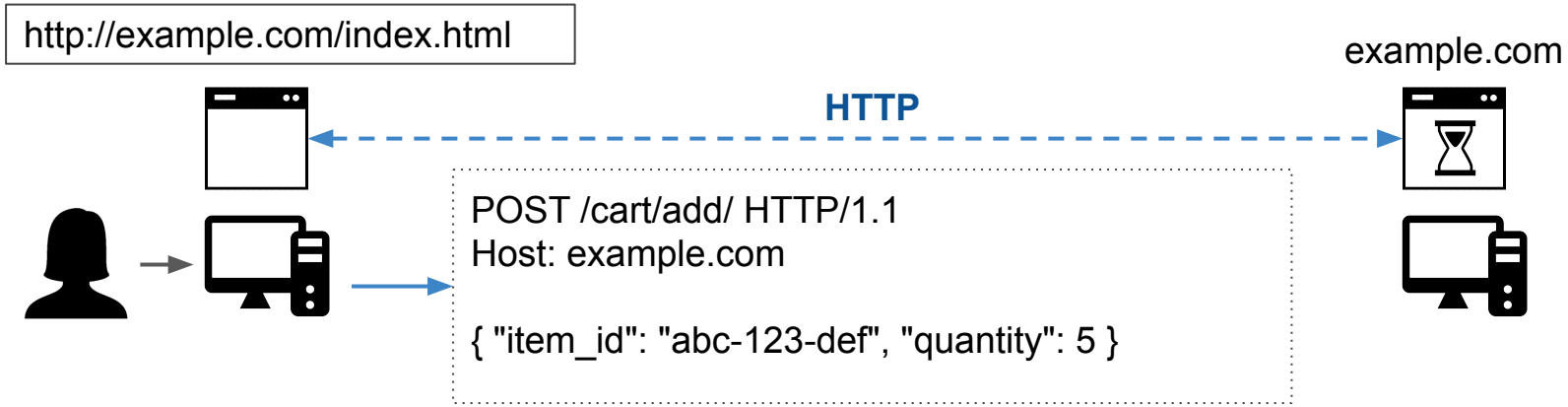
## Session: Why is it relevant to Web Applications?

- 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 **require** some **state to be maintained**

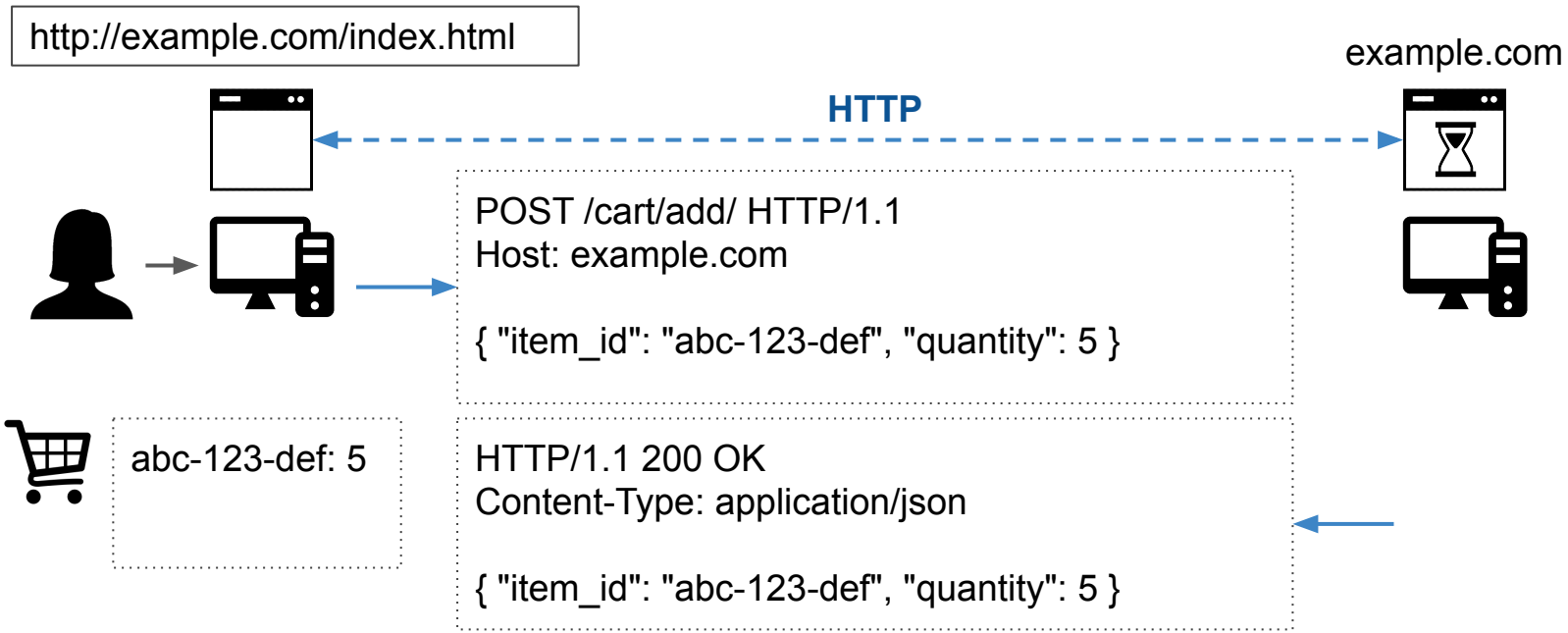
# Session: Why is it relevant to Web Applications?



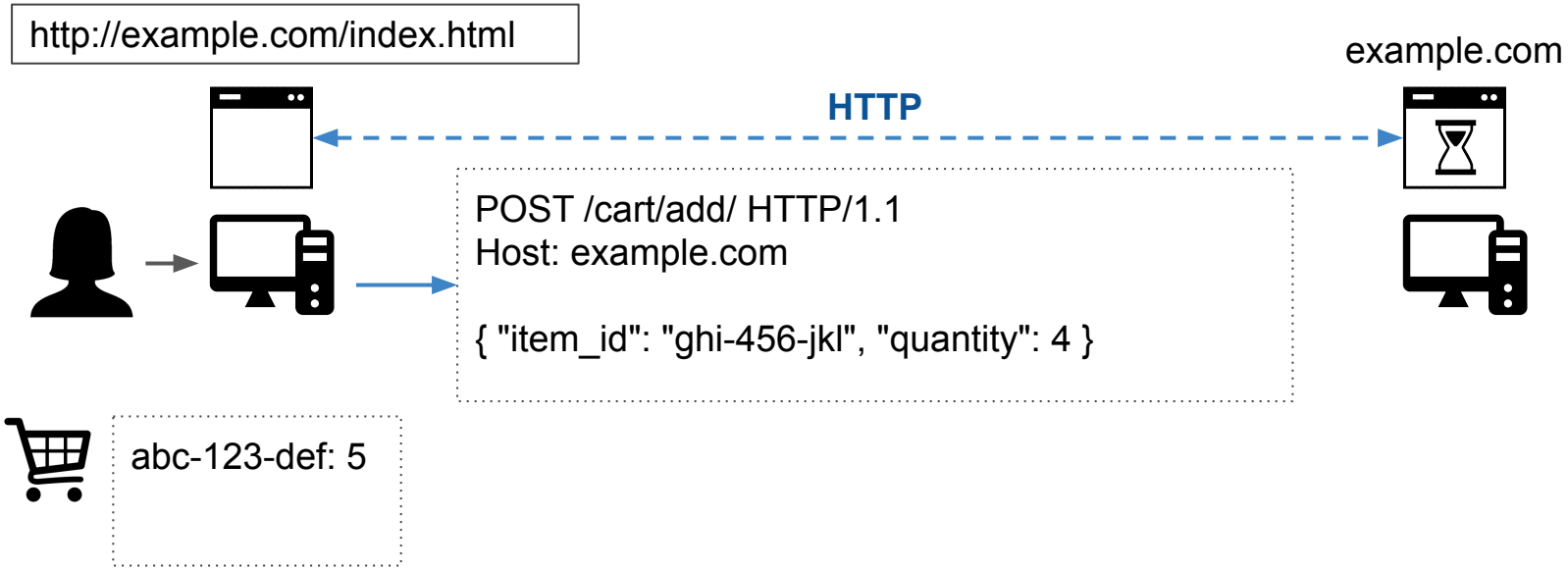
# Session: Why is it relevant to Web Applications?



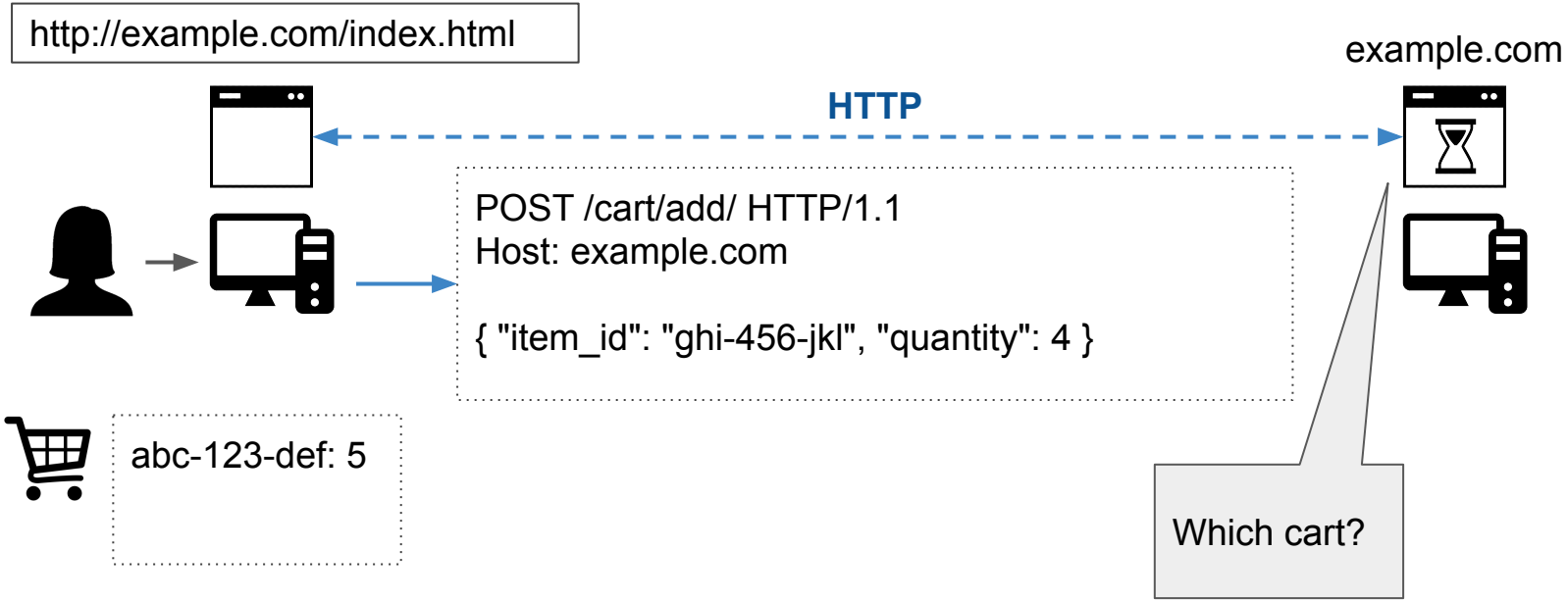
# Session: Why is it relevant to Web Applications?



# Session: Why is it relevant to Web Applications?



# Session: Why is it relevant to Web Applications?



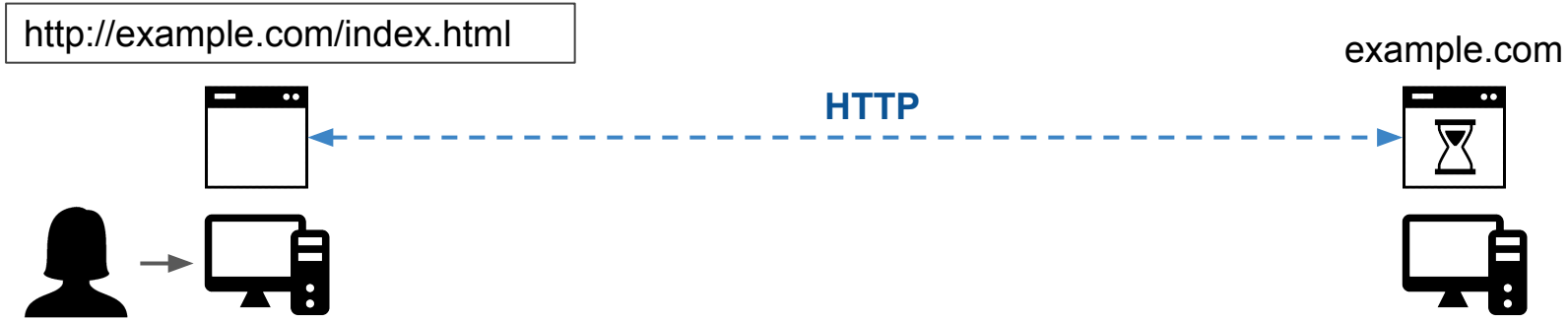
## Cookie

1. Session
2. **Cookie**
3. Web Security

## Cookie: What is it?

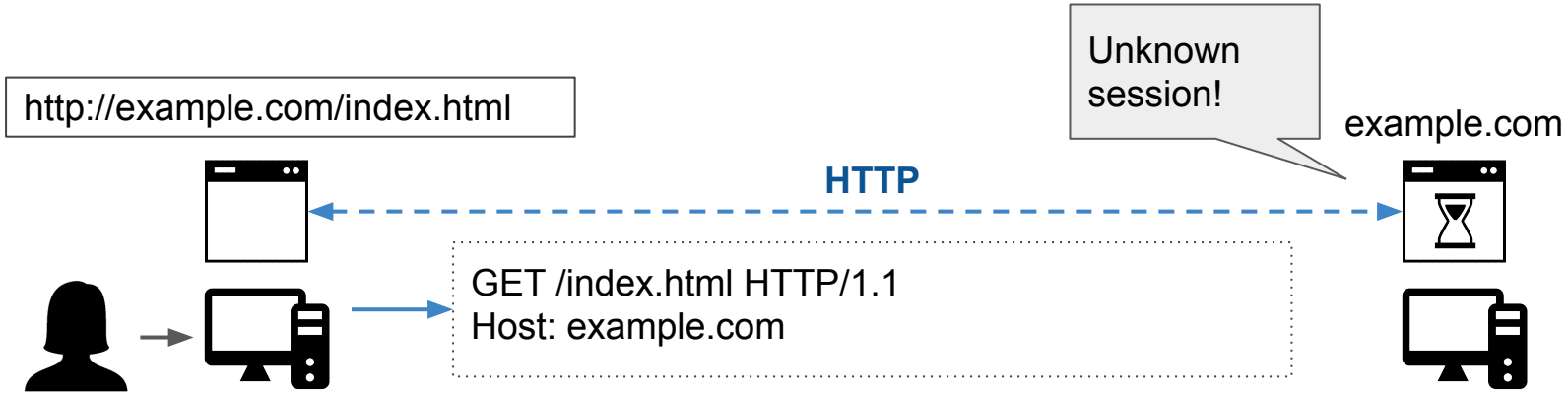
- Cookie is a piece of data that is always passed between the server and the client in consecutive HTTP messages
- At the minimum, a cookie can store a session ID to relate multiple HTTP requests and responses
- Mainly used for:
  - Session management
  - Personalization
  - Tracking User Behaviour

# Cookie: What is it?



# Cookie: What is it?

# Cookie: What is it?

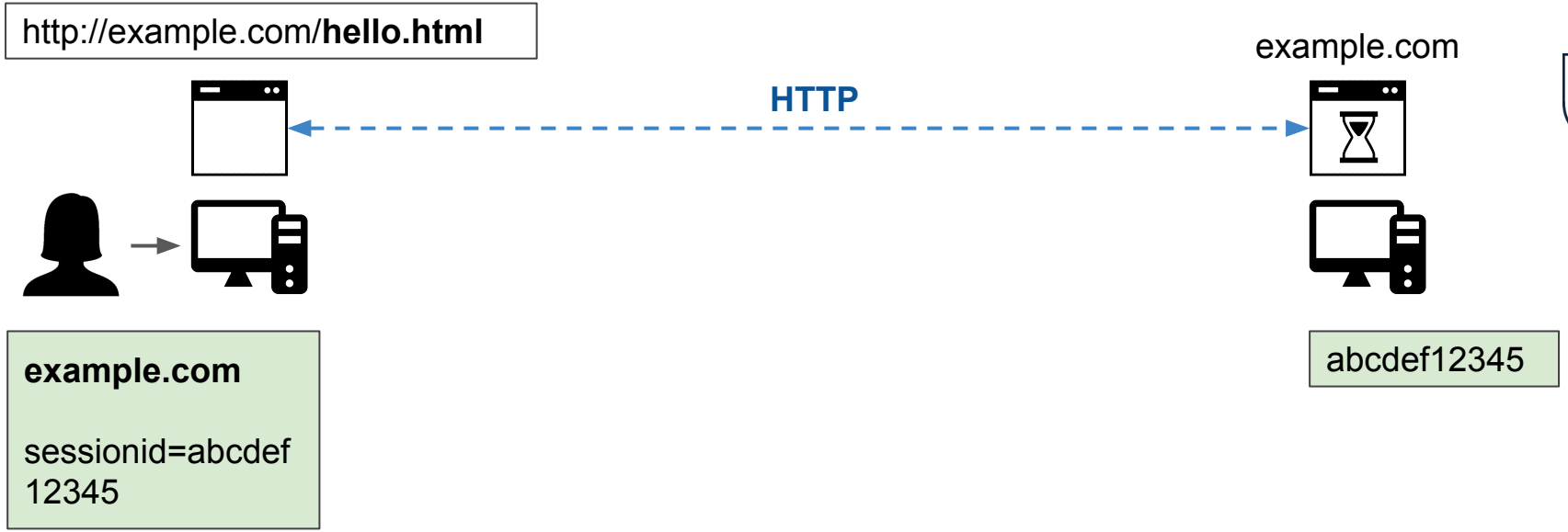


# Cookie: What is it?

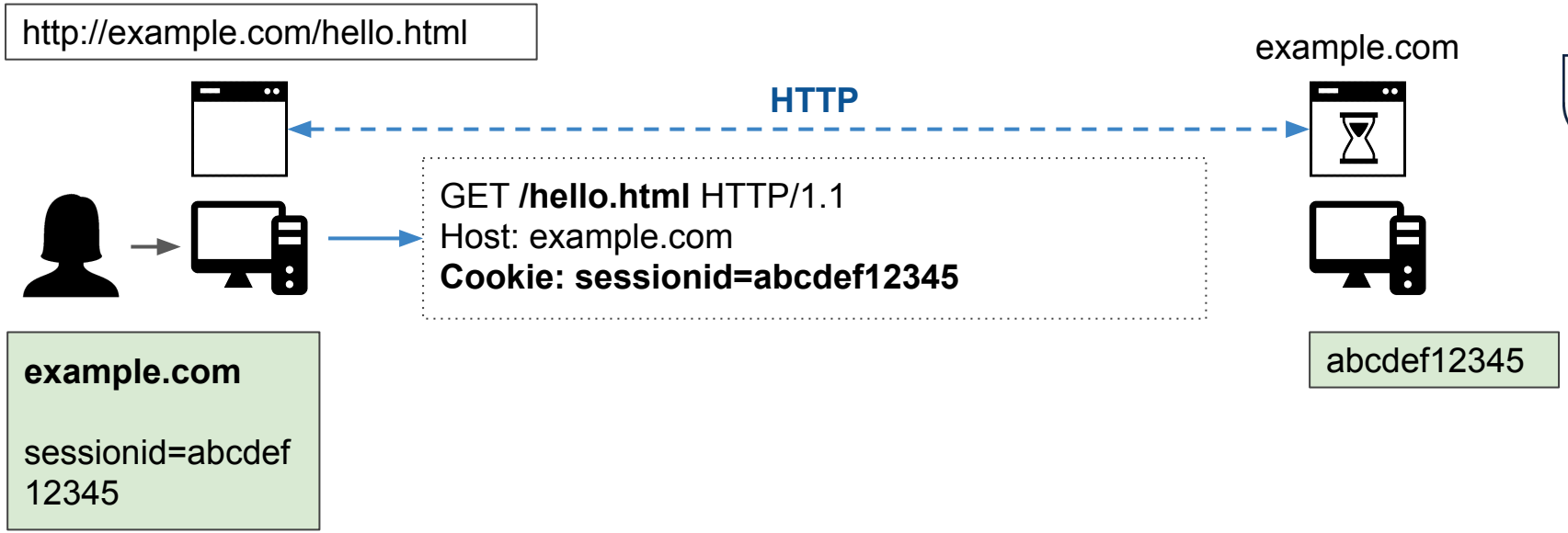
# Cookie: What is it?

# Cookie: What is it?

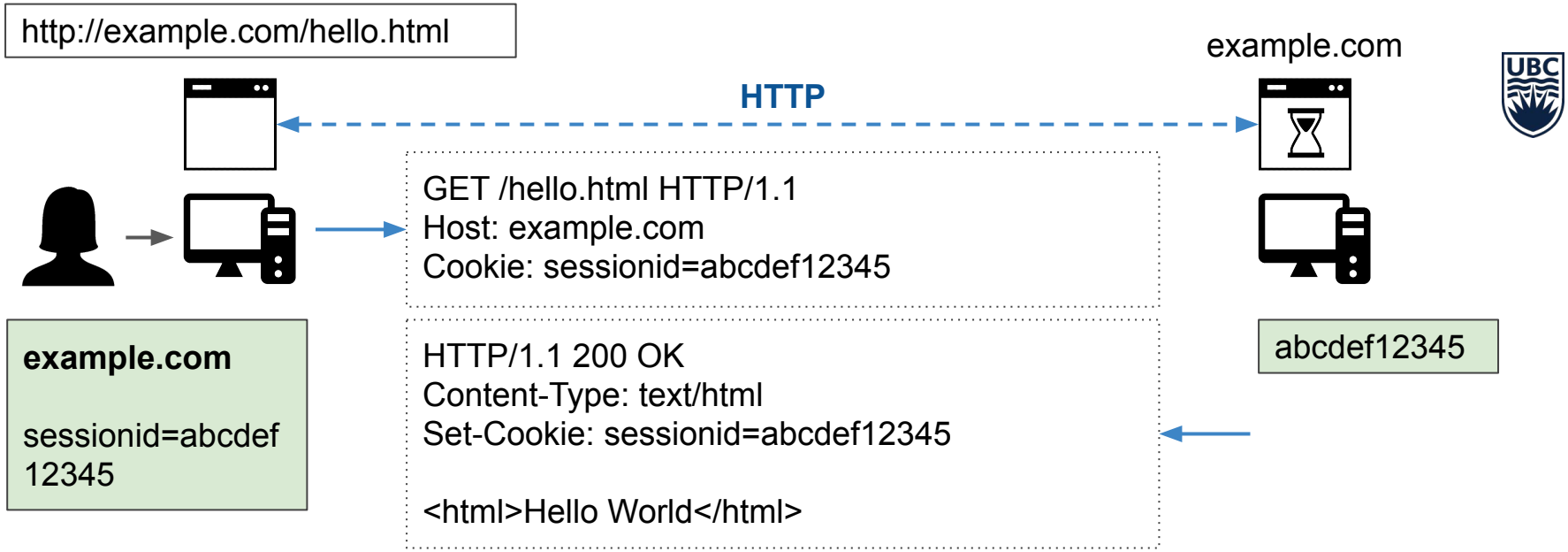
# Cookie: What is it?



# Cookie: What is it?



# Cookie: What is it?



## Cookie: Format

- Name: indicates the type of information
- Value: the data representing the information
- Attributes: set by server only
  - Domain: specifies the scope of the cookie
  - Path: which path the cookie is allowed to be sent to
  - Expires: when the cookie should expire
  - Max-Age: the maximum age for the cookie
  - 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: theme=default
Set-Cookie: language=en
Set-Cookie: currency=cad

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

# Cookie: Format

- Example: Client Request

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

## Web Security

1. Session
2. Cookie
3. **Web Security**

## Web Security

- Same-Origin Policy
- Cross-site Scripting (XSS)
  - Cookie Stealing and Session Hijacking
- Cross-site Request Forgery (XRF or CSRF)

## Web Security: Same-Origin Policy

- Same-Origin Policy says only scripts loaded from the same origin can be executed in the page
  - Enforced by all browsers
- Intent: Two different web domains should not be able to tamper with each other's contents
- Easy to state, but many exceptions in practice
  - Visual display is shared
  - Timing and DOM events are shared
  - Cookies can be shared
  - Send/receive messages for Cross-Origin Requests

## Web Security: Same-Origin Policy

- Assign an origin for each resource in a web page (e.g., cookies, DOM sub-tree, network)
  - A script can only access elements belonging to the same origin as itself
- Definition of an origin (URI scheme, Hostname, port)
  - URI Scheme: Protocol (typically http or https)
  - Hostname: domain name (e.g., **example.com**:8080)
  - Port: example.com:**8080** (if unspecified, defaults to 80 for http and 443 for https))

## Web Security: Same-Origin Policy

- Each frame gets the origin of its URL
- Scripts executed by a frame execute with the authority of the HTML file's origin
  - True for both inline scripts and those pulled from external domains
- Passive content (e.g., CSS, Images) can't run code and is hence given zero authority

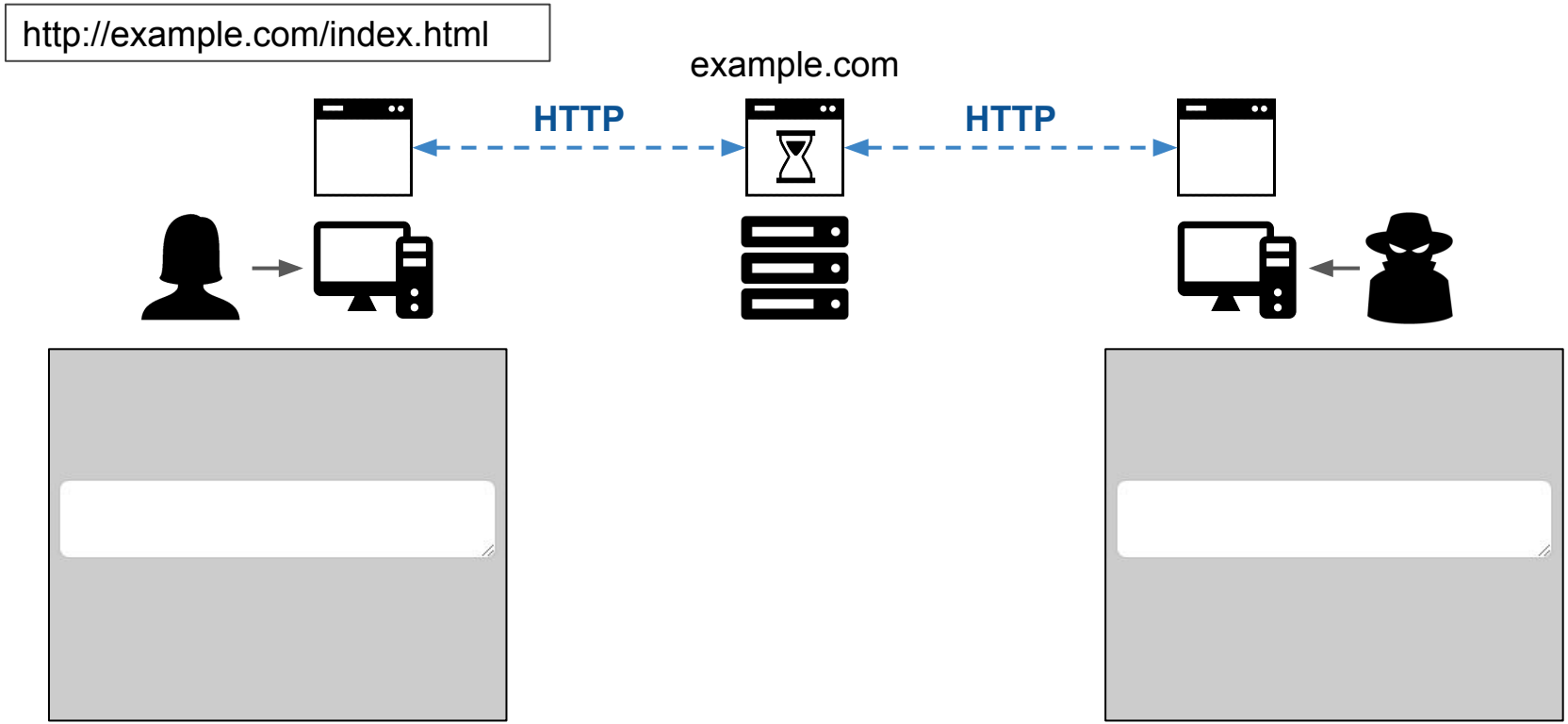
## Web Security: Same-Origin Policy

- A Frame is a self-contained entity in a webpage which has scripts and content
- A frame's origin is set to the domain it comes from, but if and only if it explicitly sets the property domain="xyz.com"
- Subframes can set their "domain=" property to only their parent domain(s) or themselves
  - Example: "ece.ubc.ca" can set its domain property to "ubc.ca", but not "utoronto.ca" (for example)

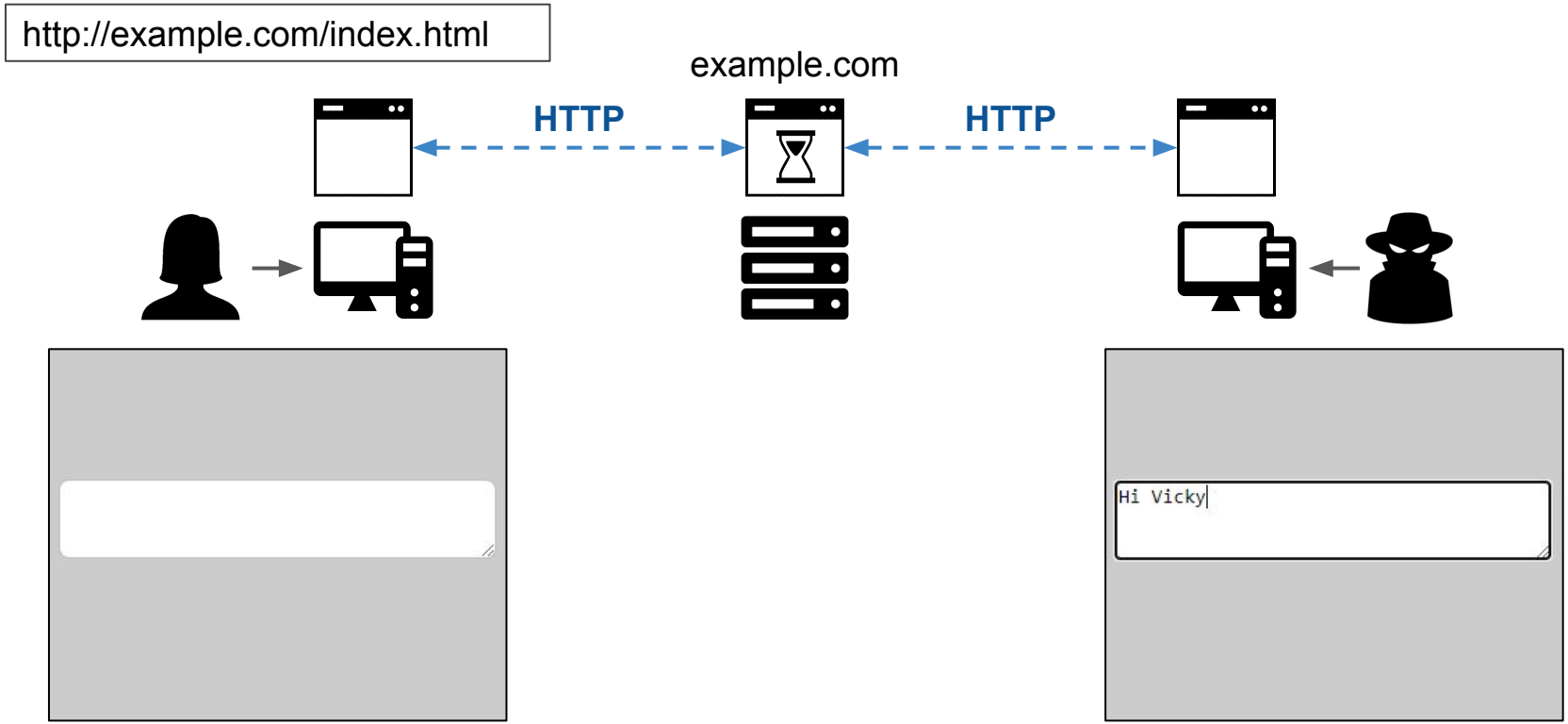
## Web Security: Cross-site Scripting

- Cross-site Scripting is executing a foreign (and malicious) piece of code as if it was 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

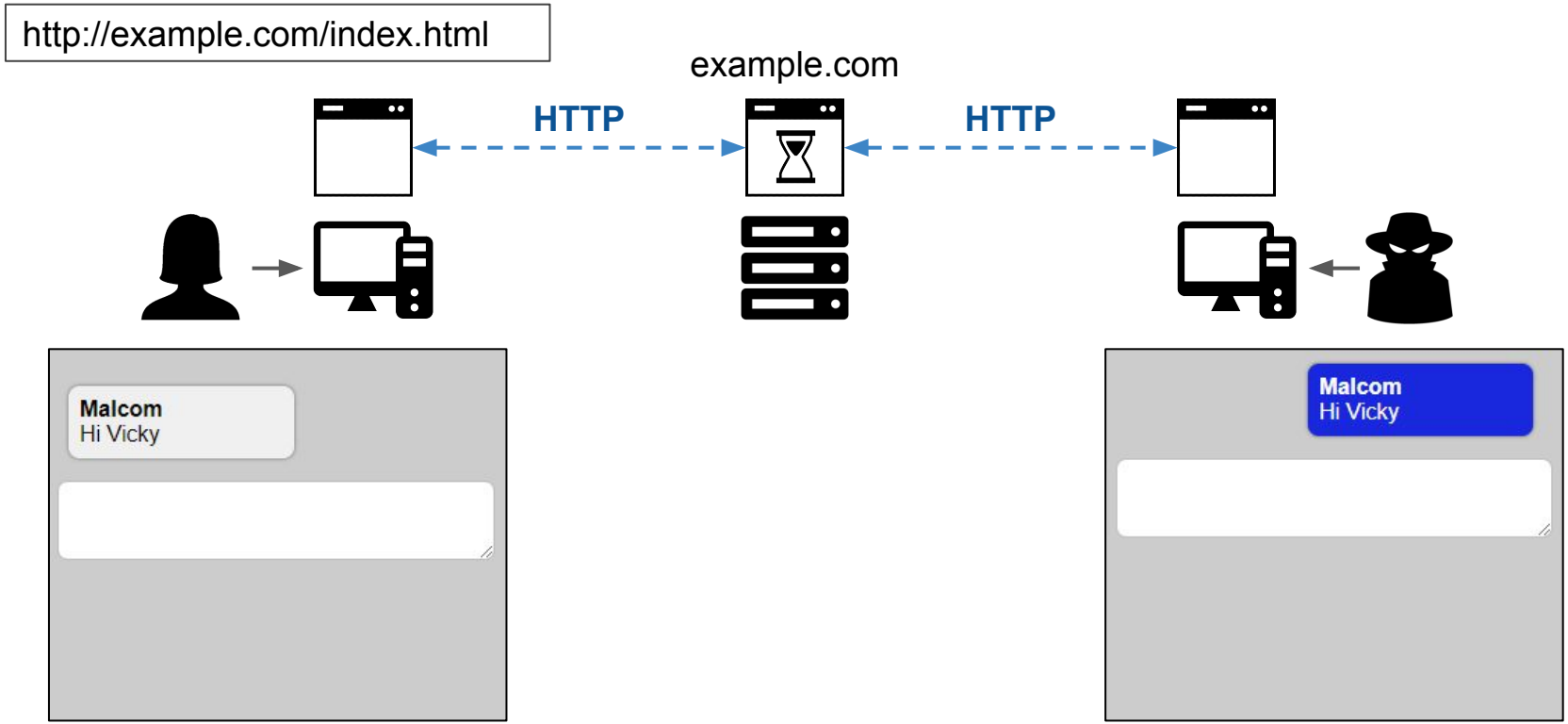
# Web Security: Cross-site Scripting



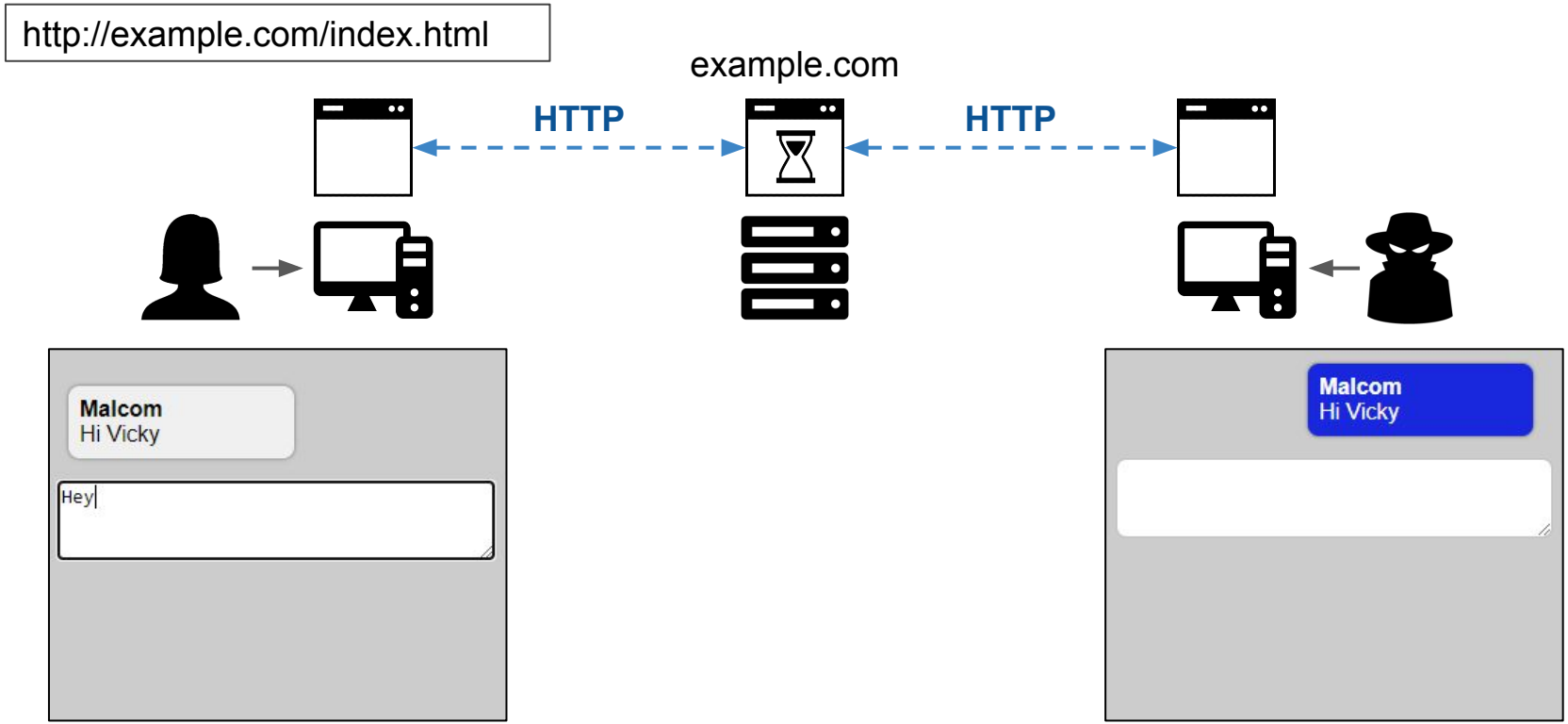
# Web Security: Cross-site Scripting



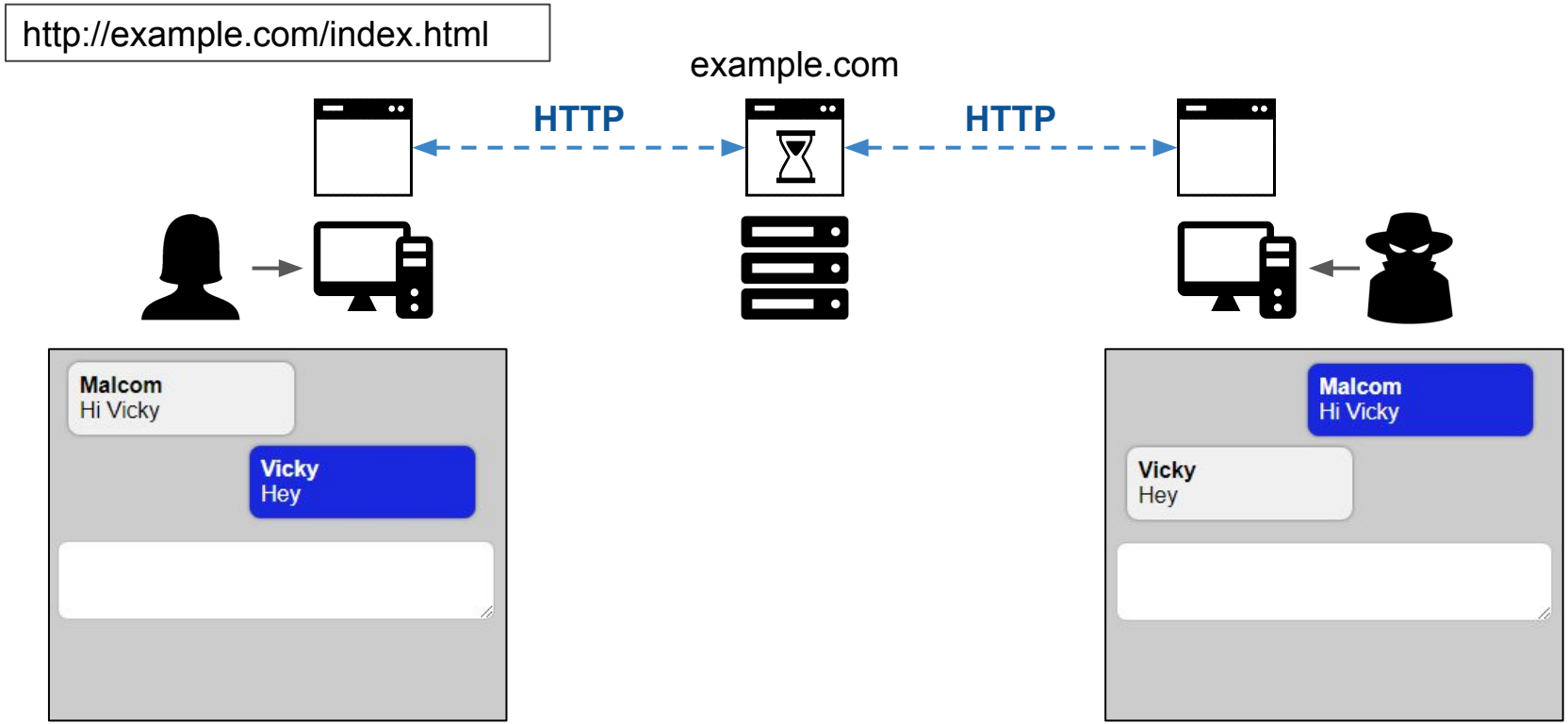
# Web Security: Cross-site Scripting



# Web Security: Cross-site Scripting

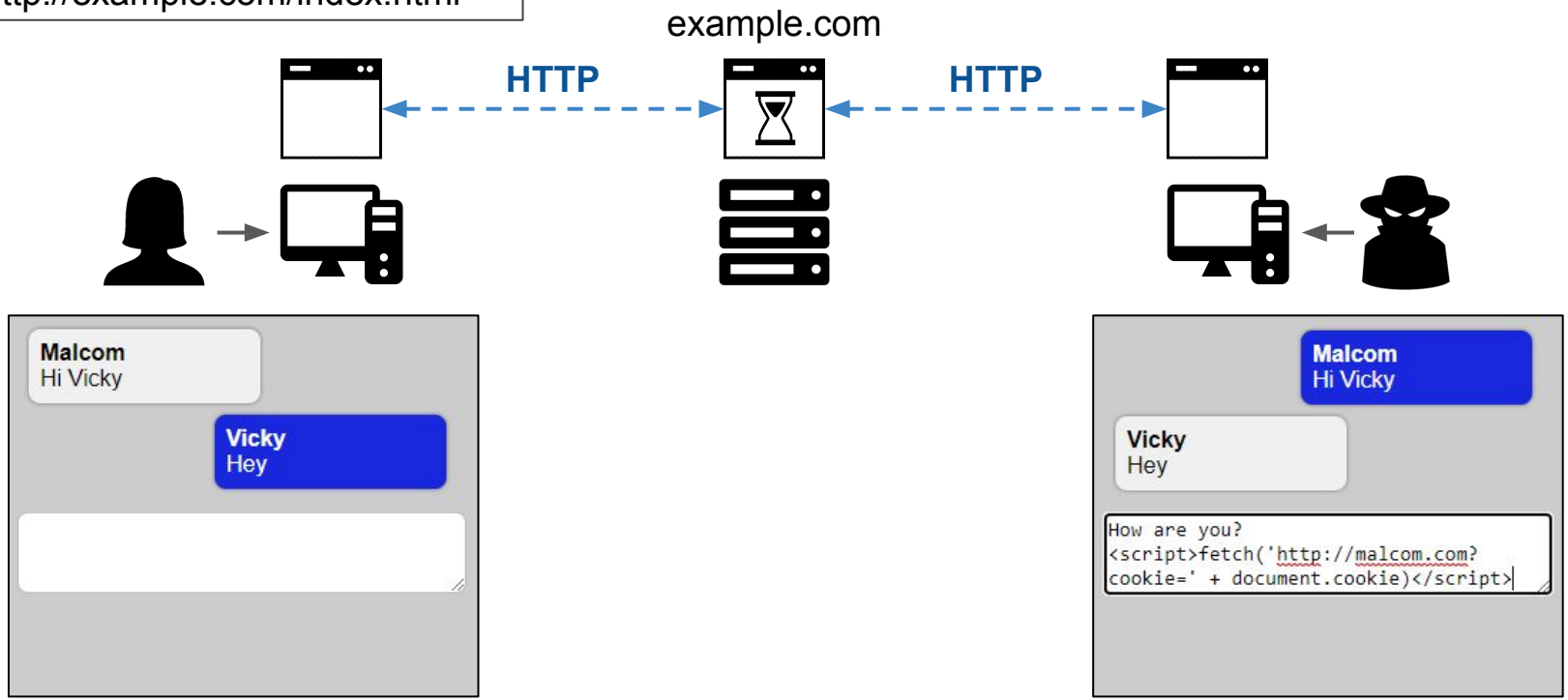


# Web Security: Cross-site Scripting



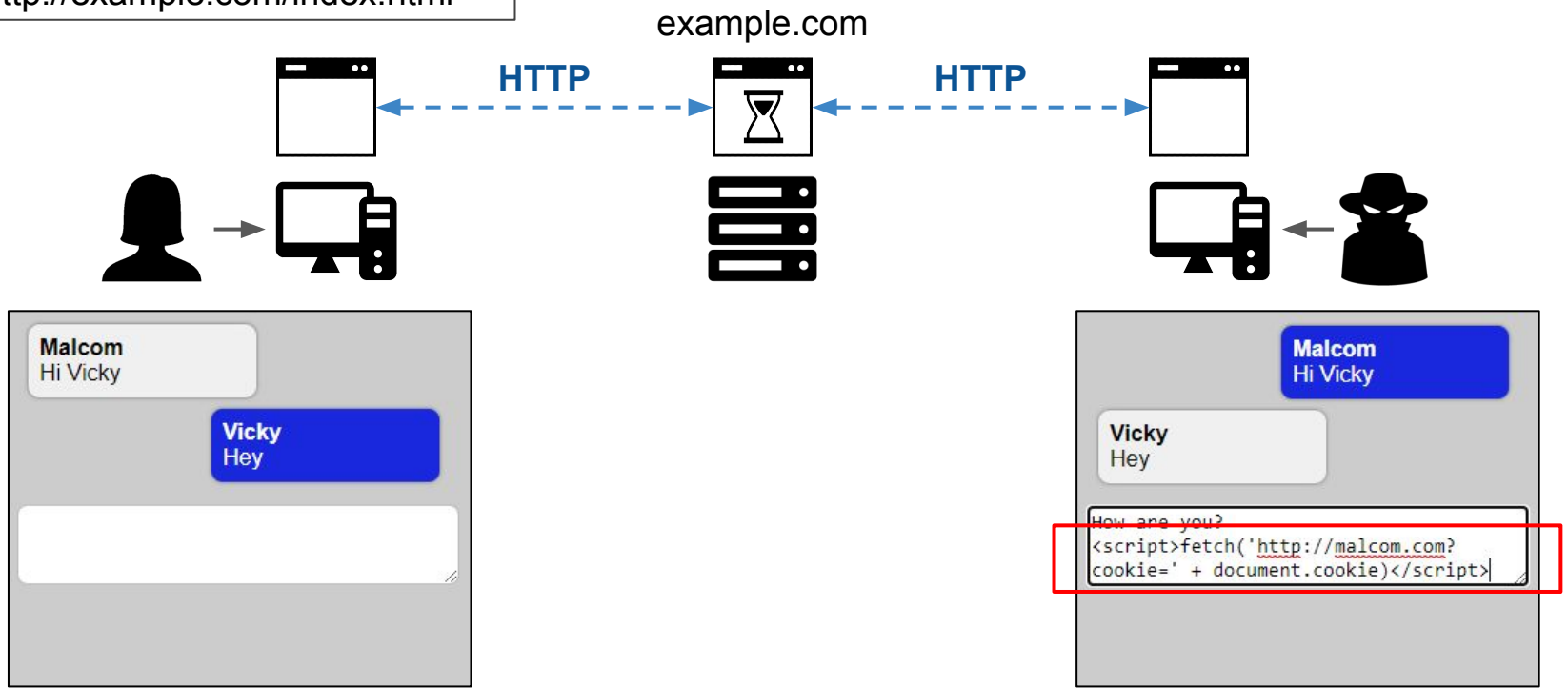
# Web Security: Cross-site Scripting

http://example.com/index.html



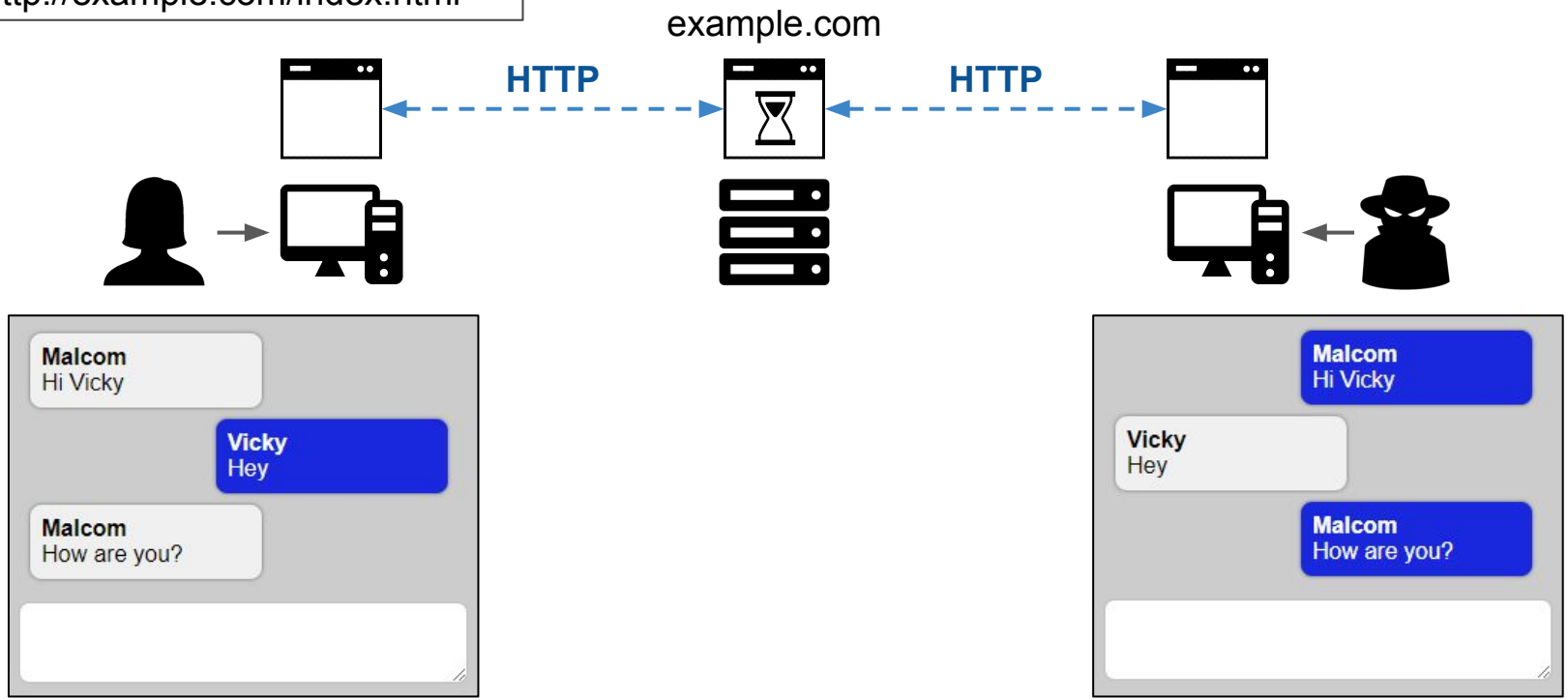
# Web Security: Cross-site Scripting

http://example.com/index.html



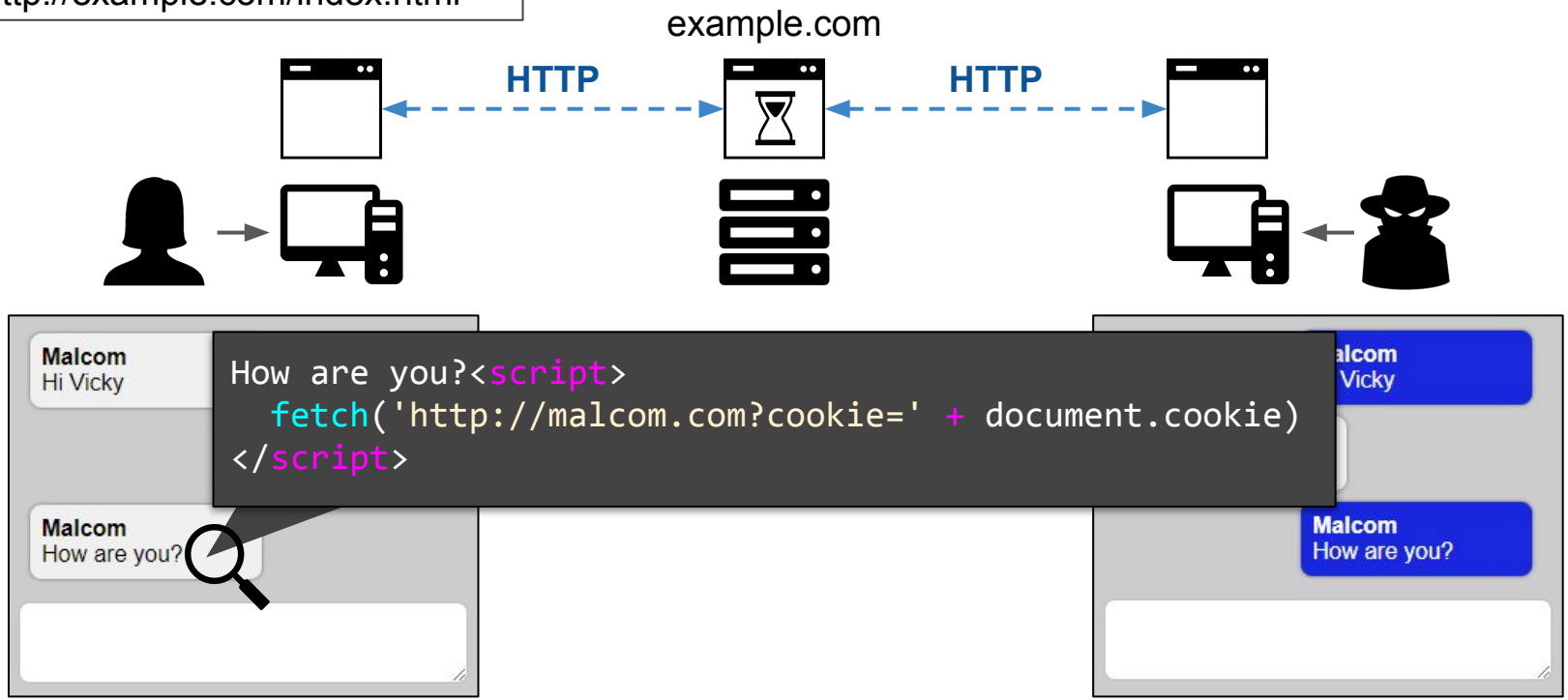
# Web Security: Cross-site Scripting

http://example.com/index.html



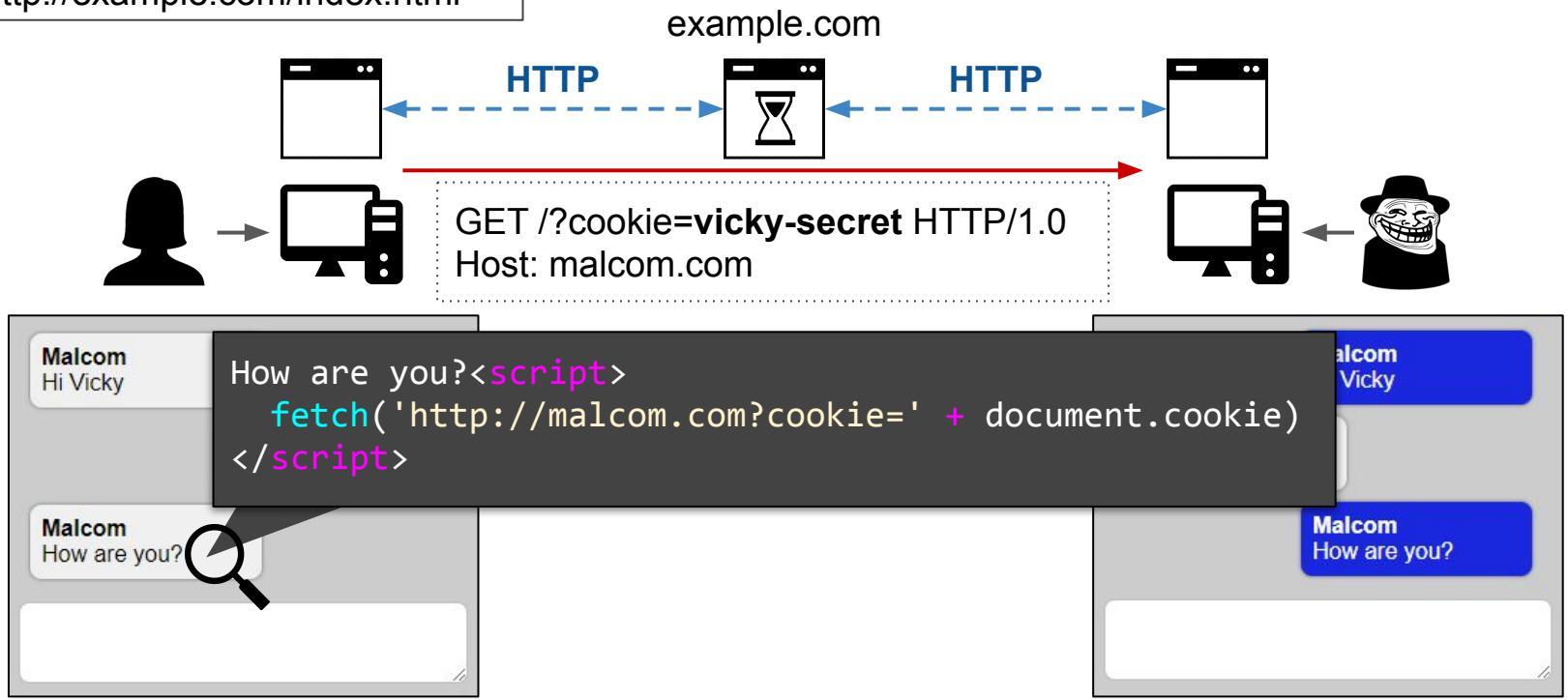
# Web Security: Cross-site Scripting

http://example.com/index.html



# Web Security: Cross-site Scripting

http://example.com/index.html



# Web Security: Cross-site Scripting

## Defense

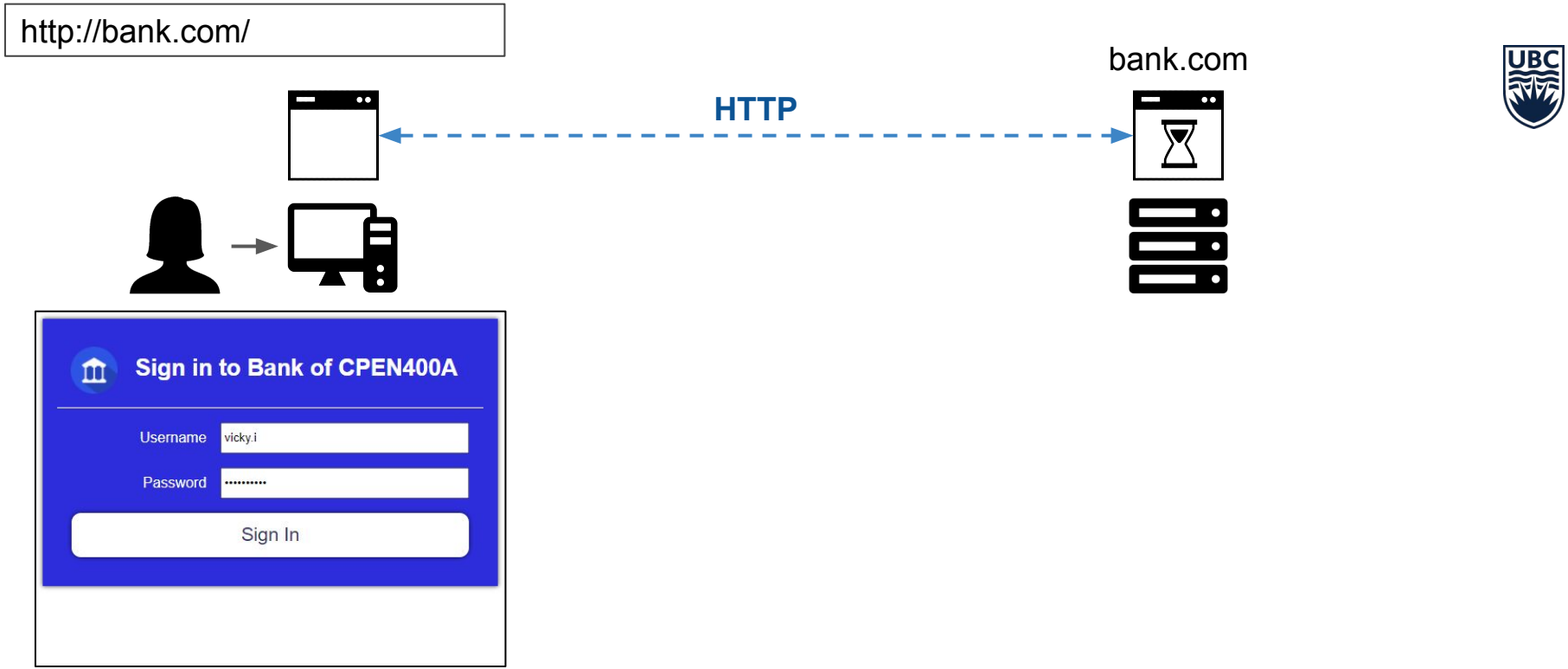
- Sanitizing user input by checking for JS
  - Hard to do as JS code can be concealed in many ways (e.g., by escaping within HTML or CSS tags)
  - Performance overhead on the server for parsing inputs
- Lighter-weight but incomplete methods
  - Tying cookies to the IP address of the user logged in (works 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 (allow servers to specify approved origins of content for web browsers) – not yet implemented in all browsers

## 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

# Web Security: Cross-site Request Forgery



# Web Security: Cross-site Request Forgery

http://bank.com/

bank.com

HTTP

POST /login HTTP/1.1
Host: bank.com

username=vicky.i
password=SecretPassword

# Web Security: Cross-site Request Forgery

http://bank.com/

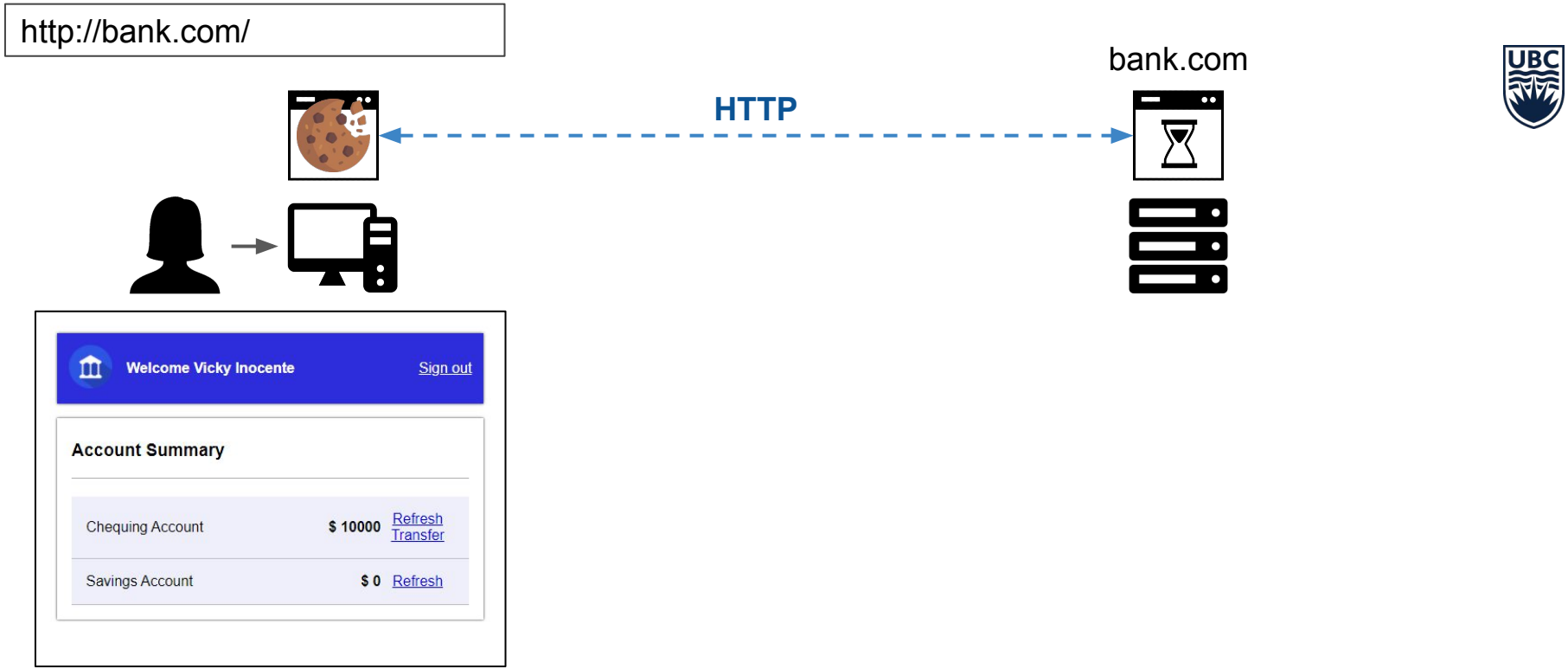
bank.com

HTTP

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

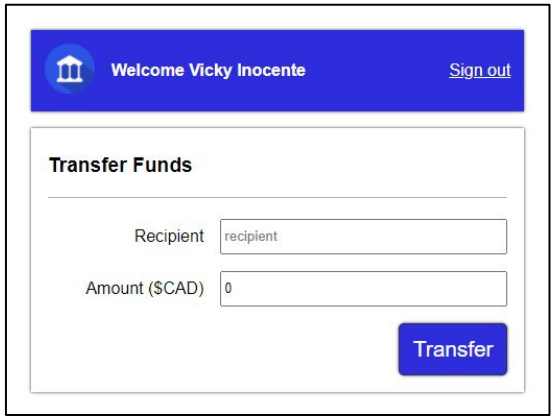
<html>Welcome </html>

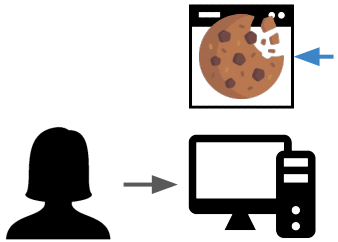
# Web Security: Cross-site Request Forgery



# Web Security: Cross-site Request Forgery

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

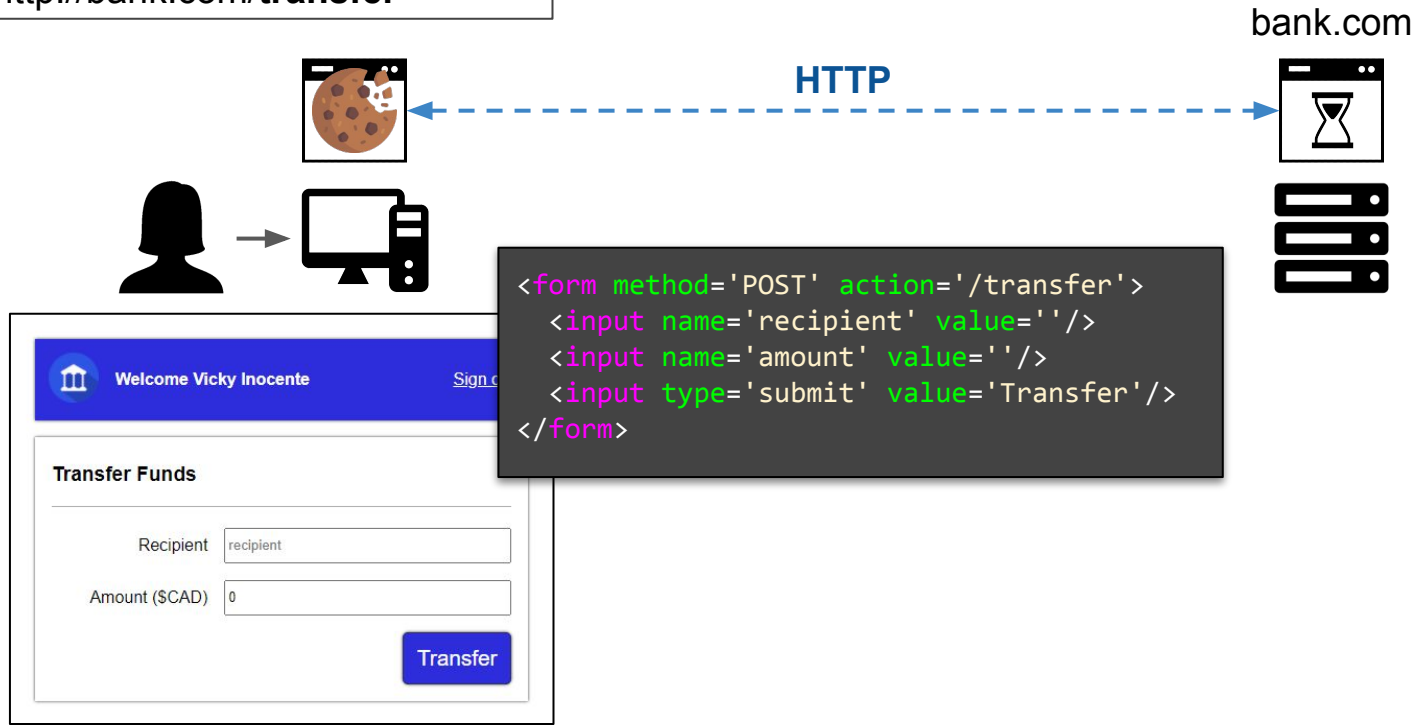




# Web Security: Cross-site Request Forgery

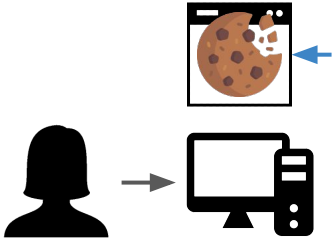
http://bank.com/**transfer**




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

# Web Security: Cross-site Request Forgery

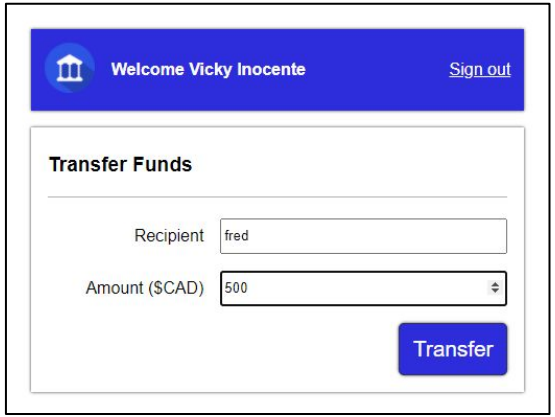
http://bank.com/**transfer**





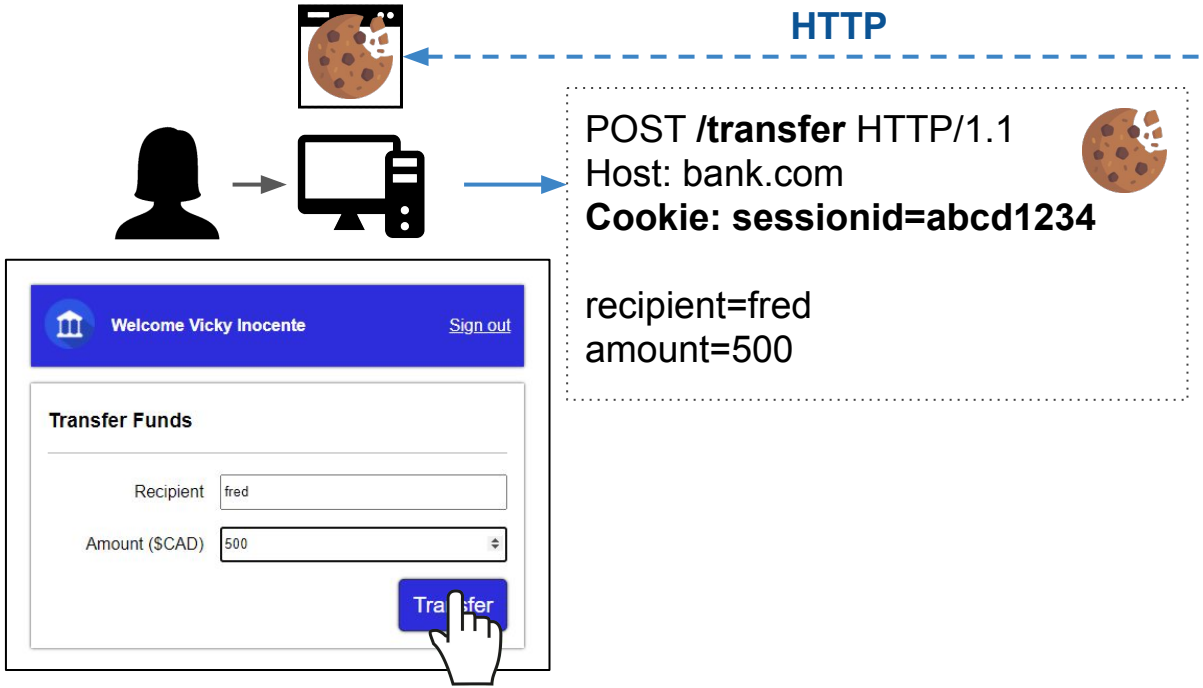
HTTP

bank.com



# Web Security: Cross-site Request Forgery

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



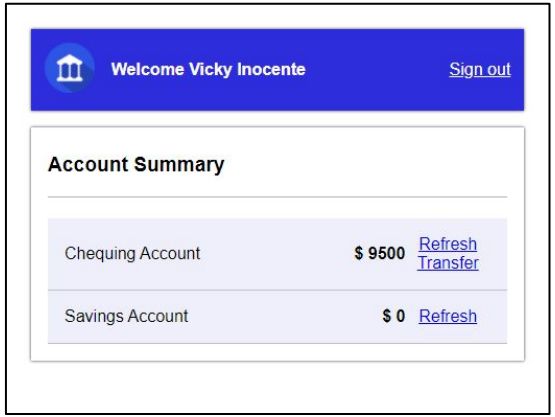
# Web Security: Cross-site Request Forgery

http://bank.com/

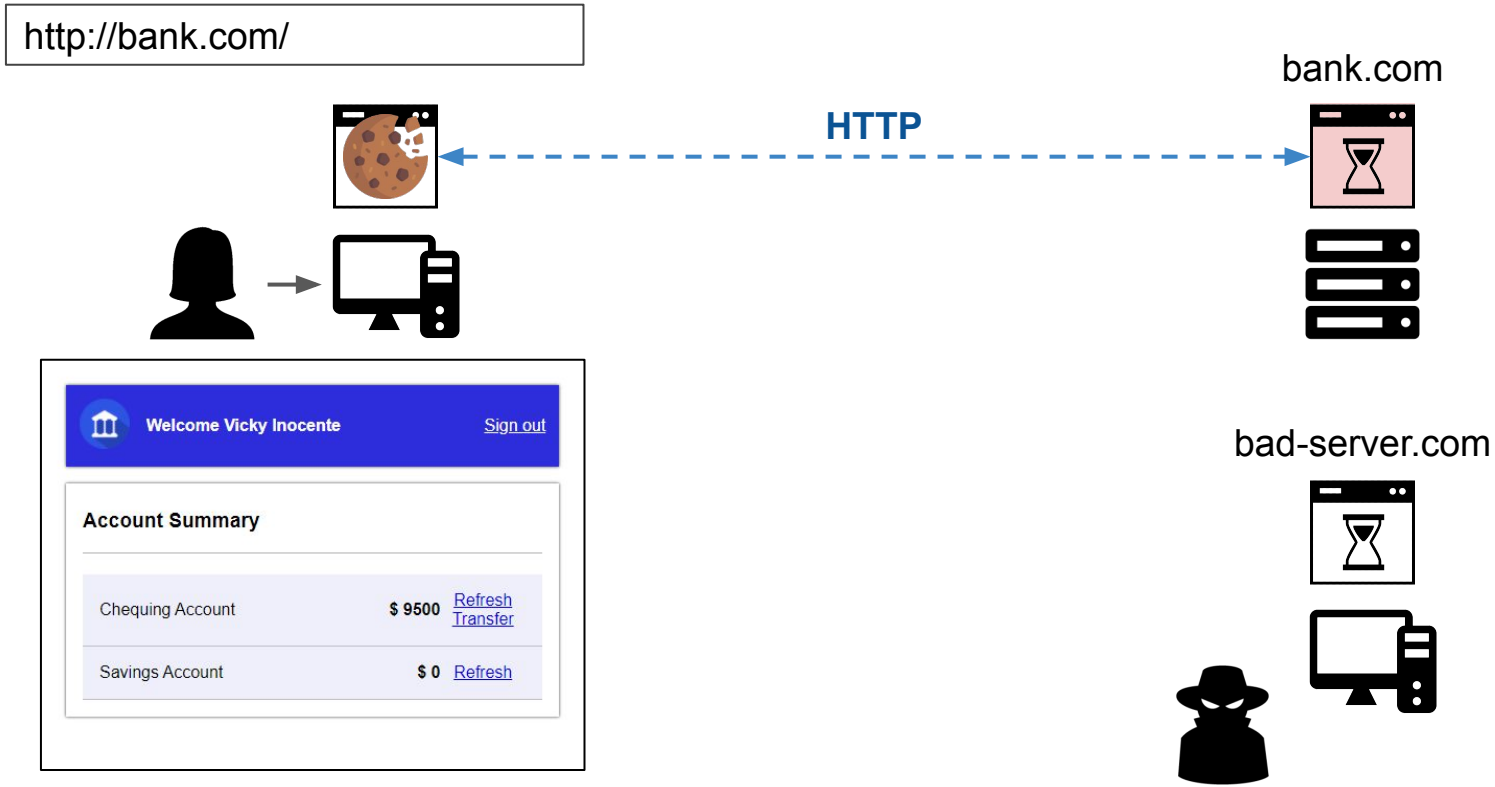
bank.com

HTTP

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

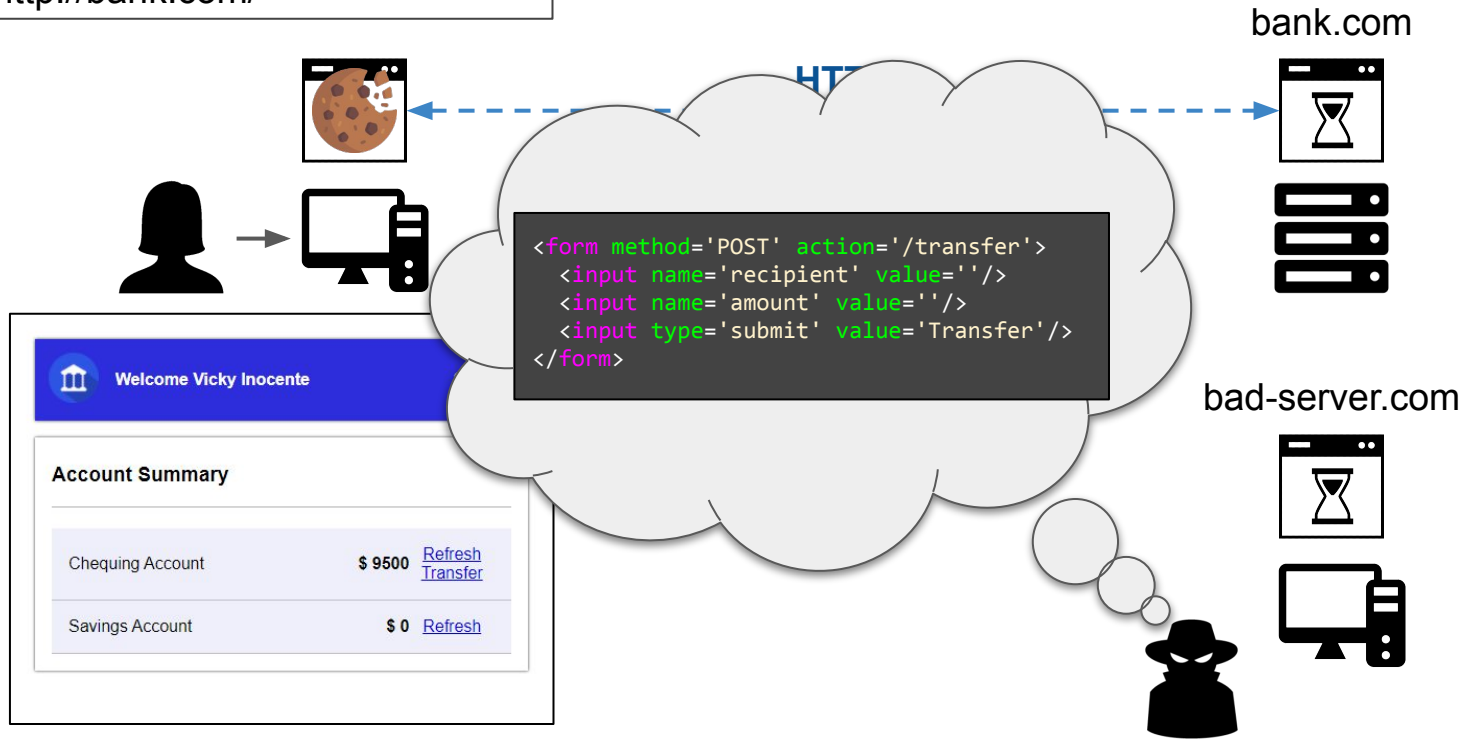


# Web Security: Cross-site Request Forgery

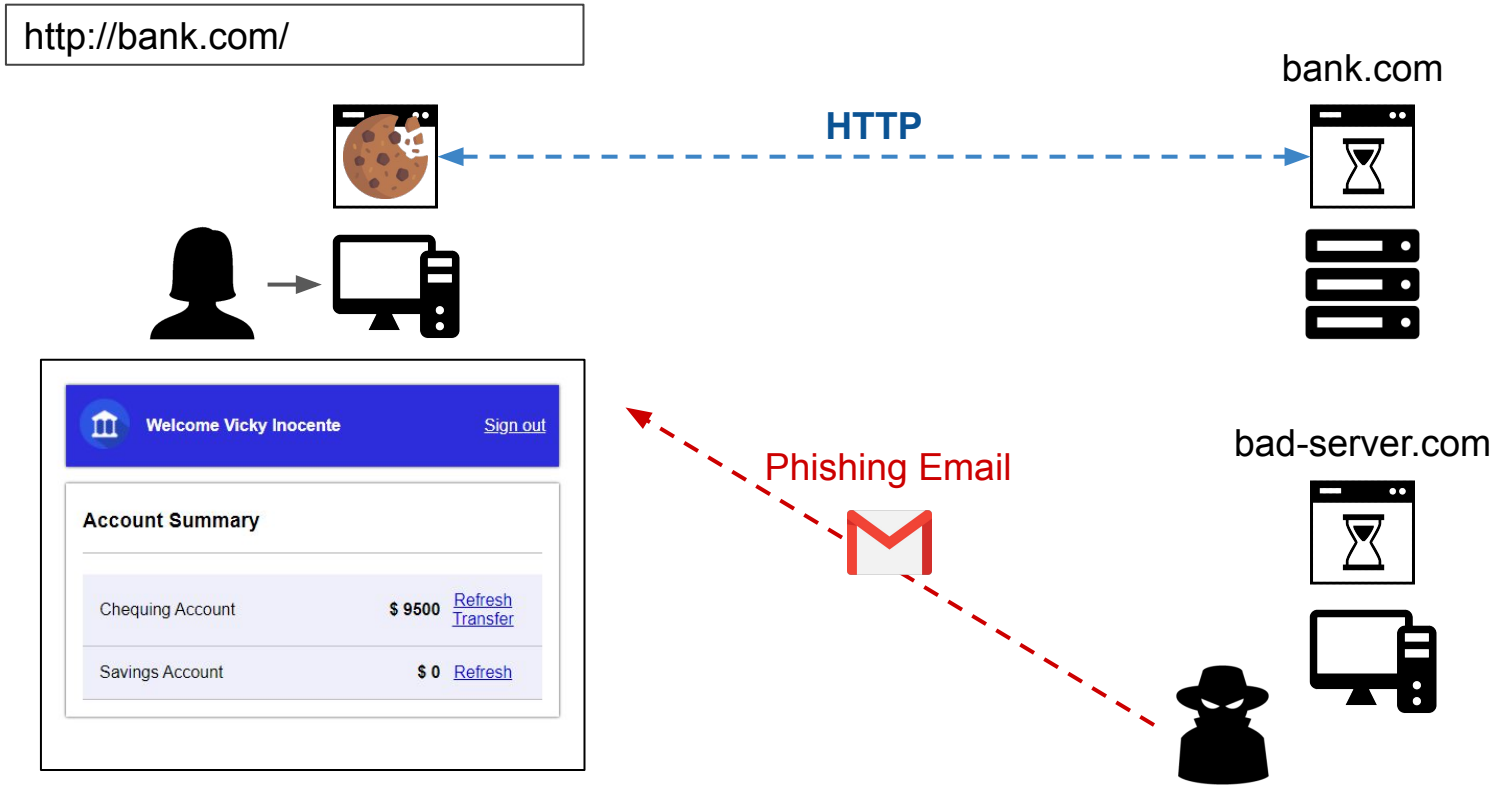


# Web Security: Cross-site Request Forgery

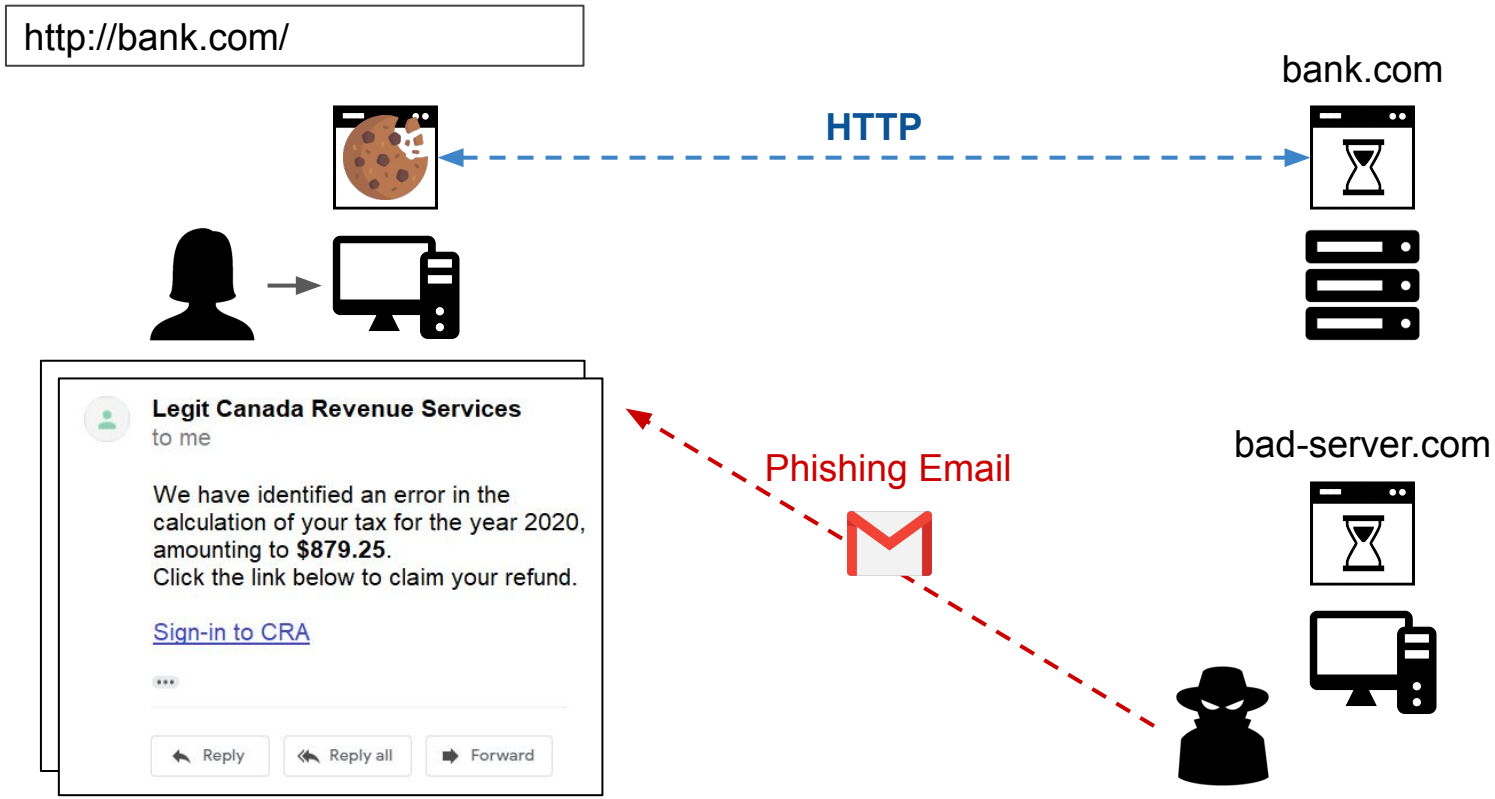
http://bank.com/



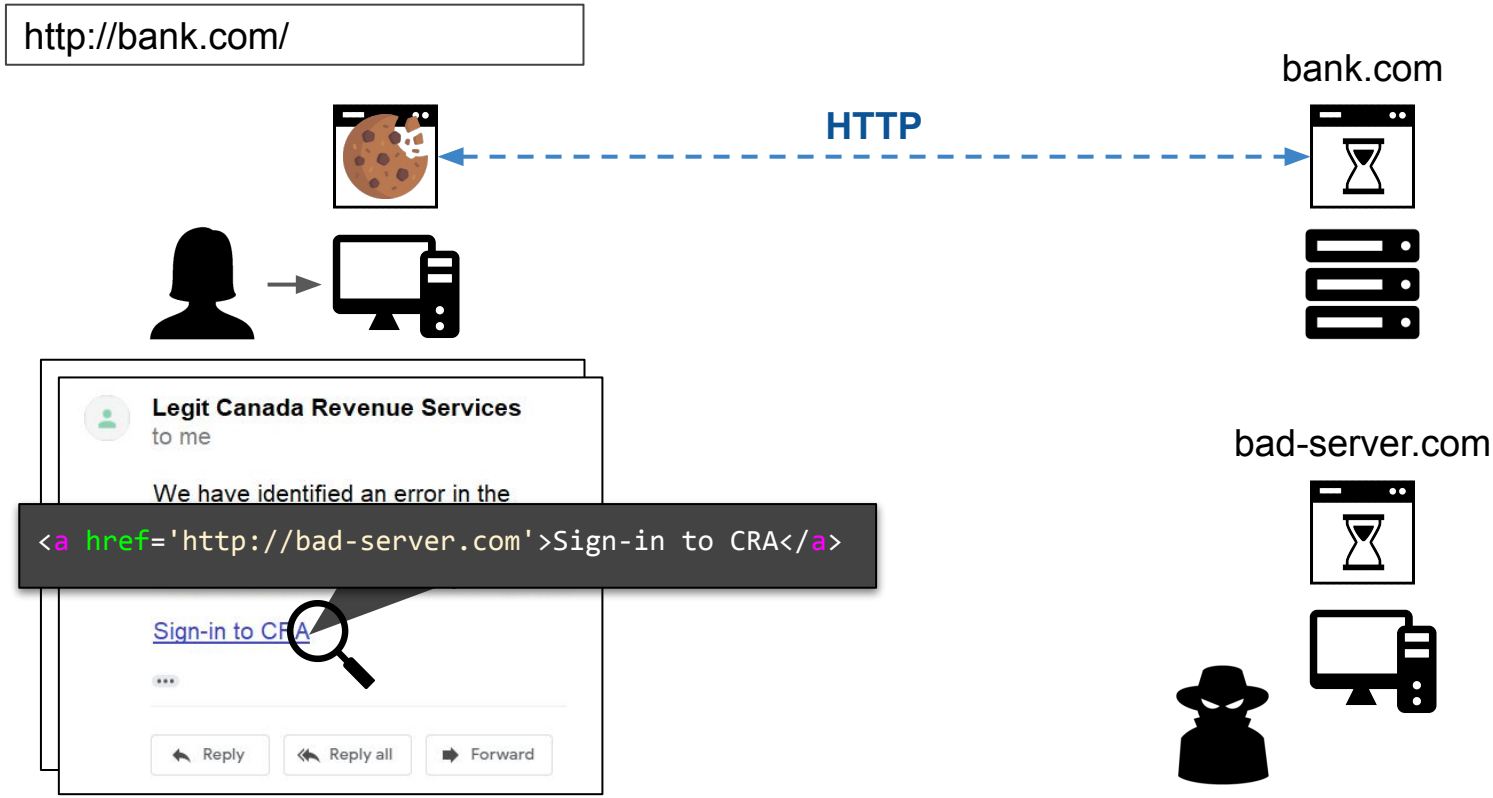
# Web Security: Cross-site Request Forgery



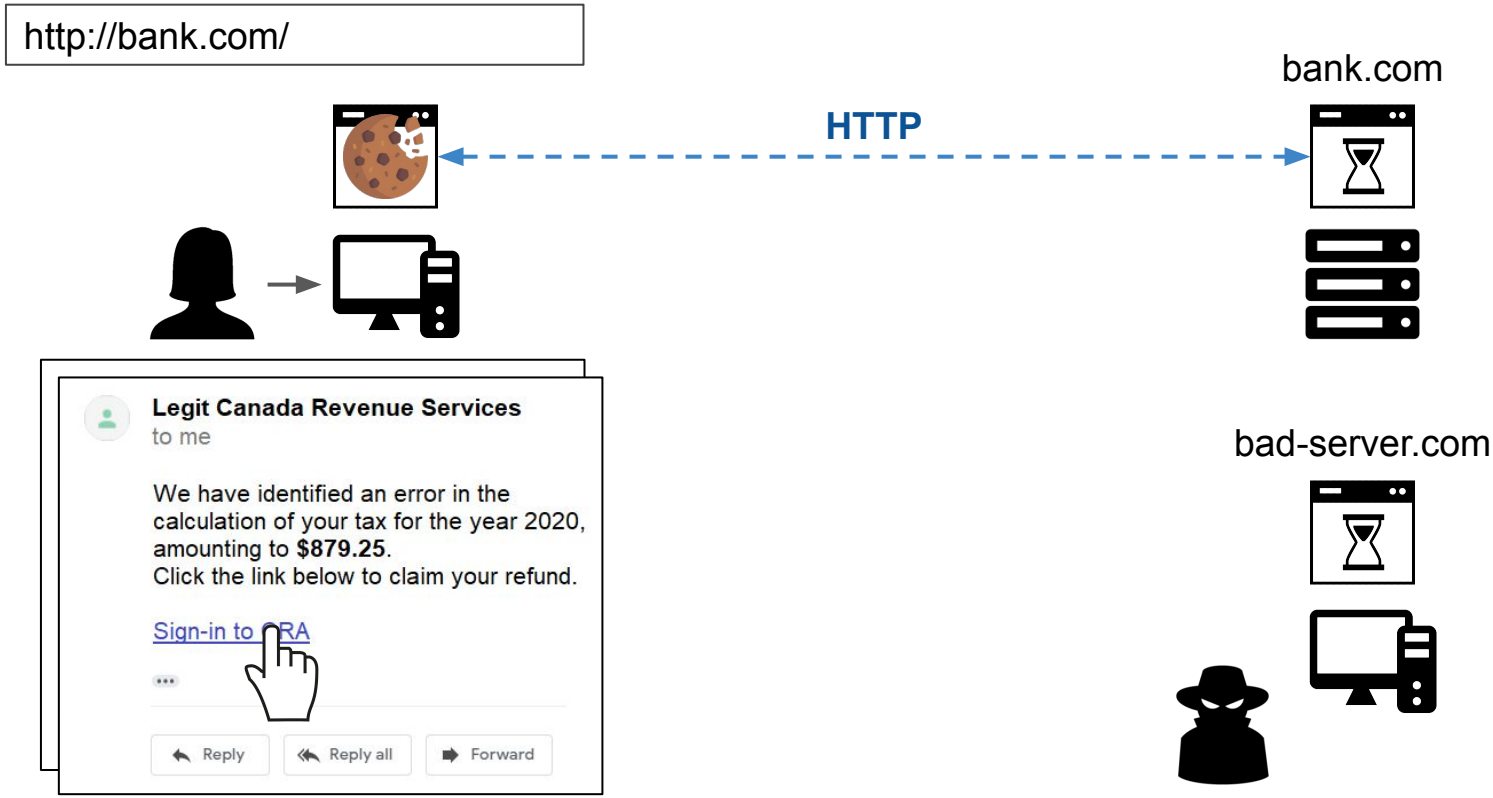
# Web Security: Cross-site Request Forgery



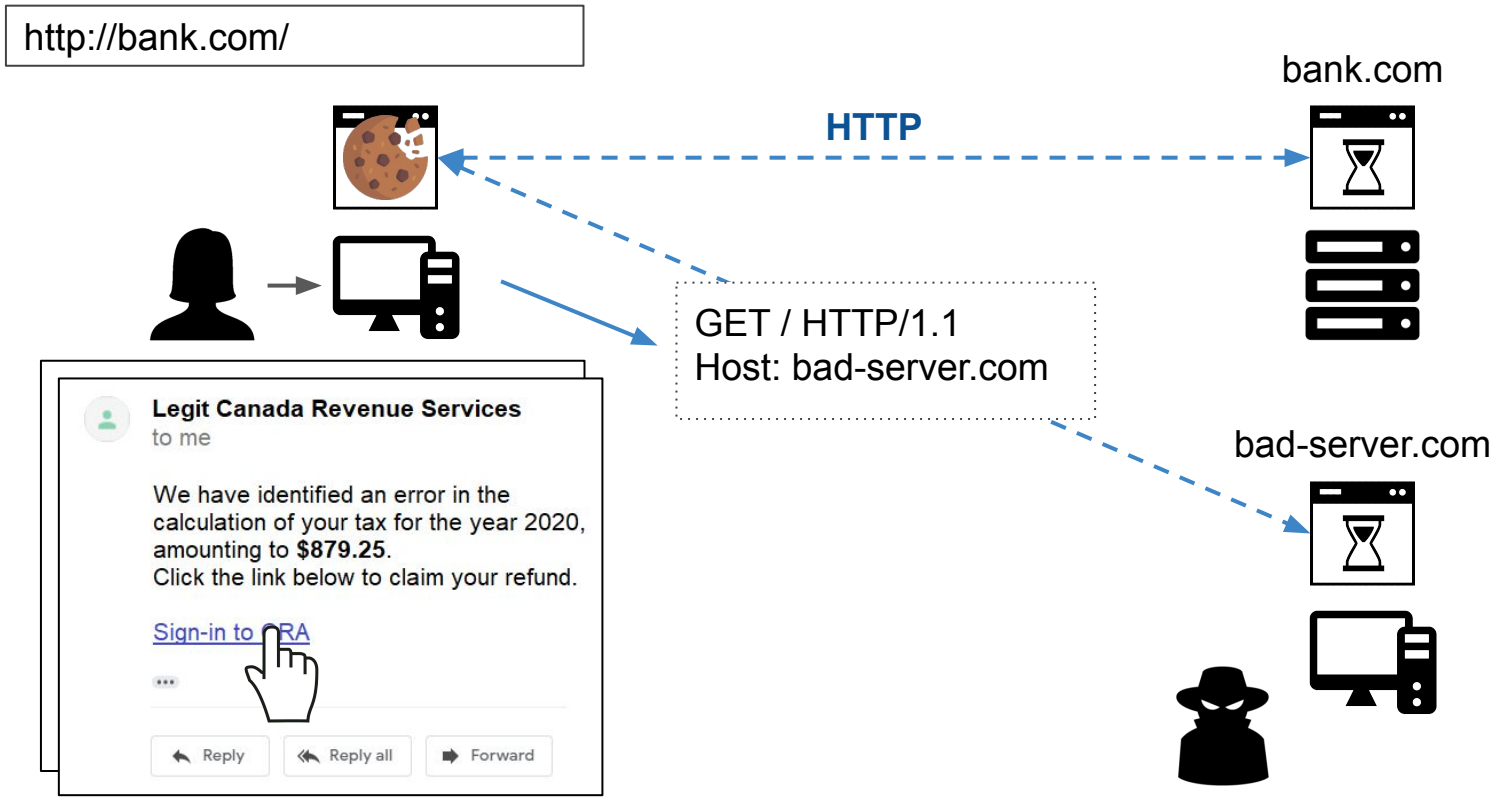
# Web Security: Cross-site Request Forgery



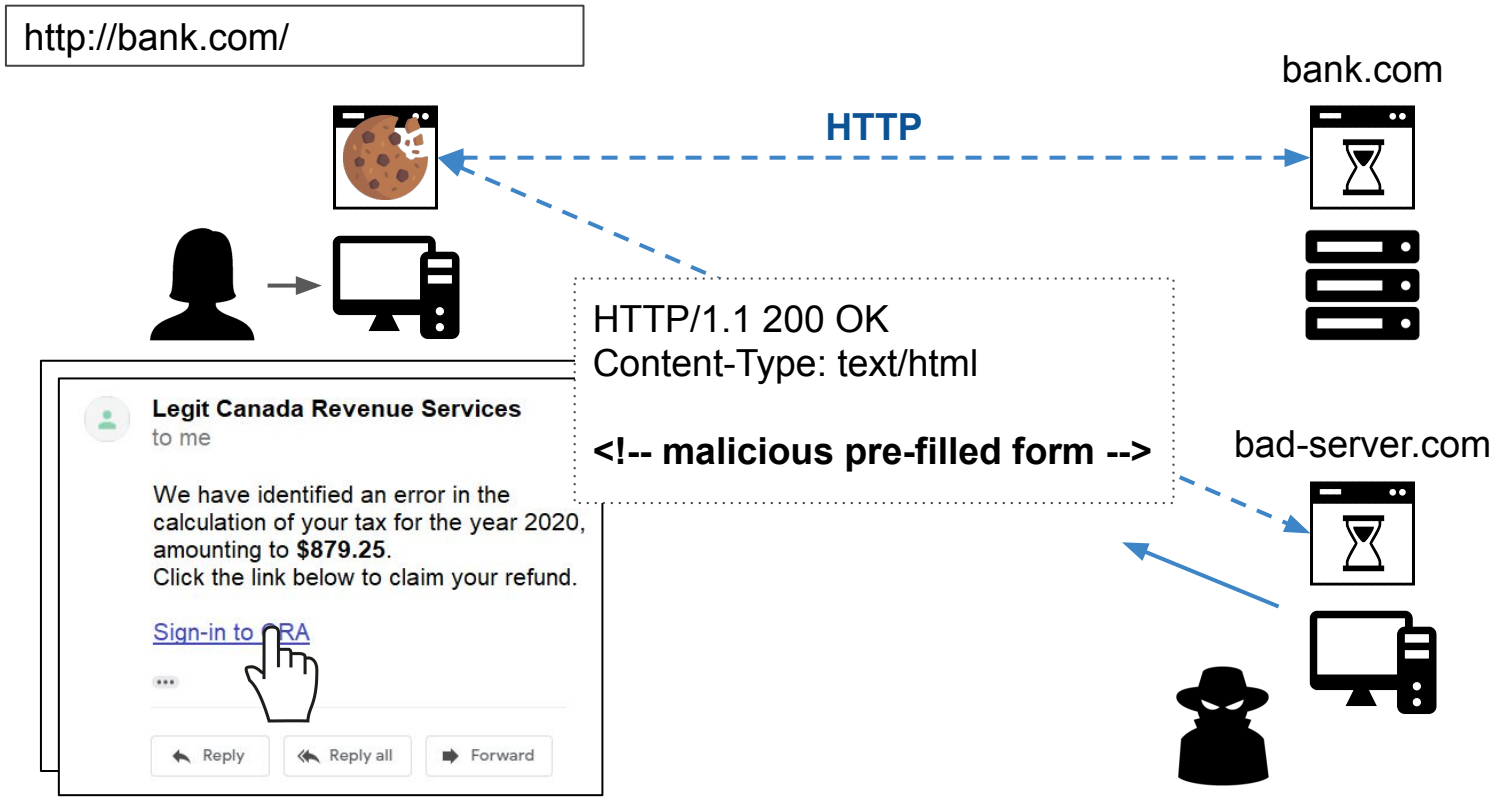
# Web Security: Cross-site Request Forgery



# Web Security: Cross-site Request Forgery



# Web Security: Cross-site Request Forgery



# 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 -->

bad-server.com

# Web Security: Cross-site Request Forgery

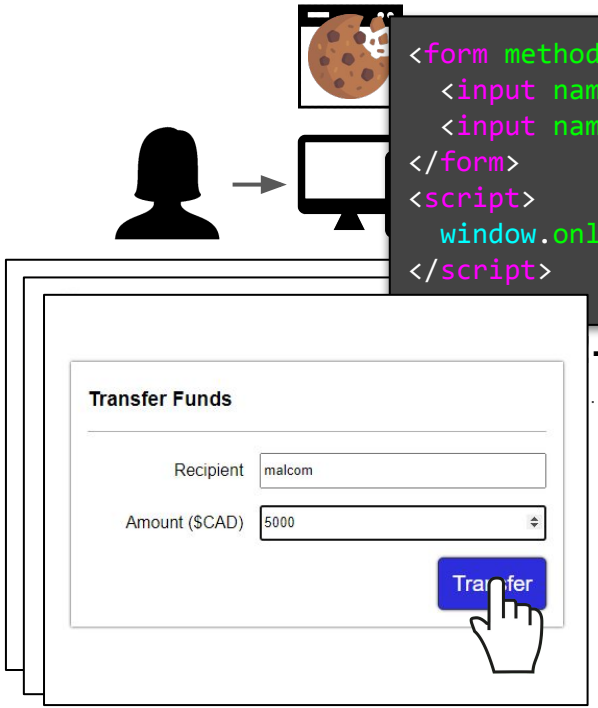
http://bank.com/

bank.com

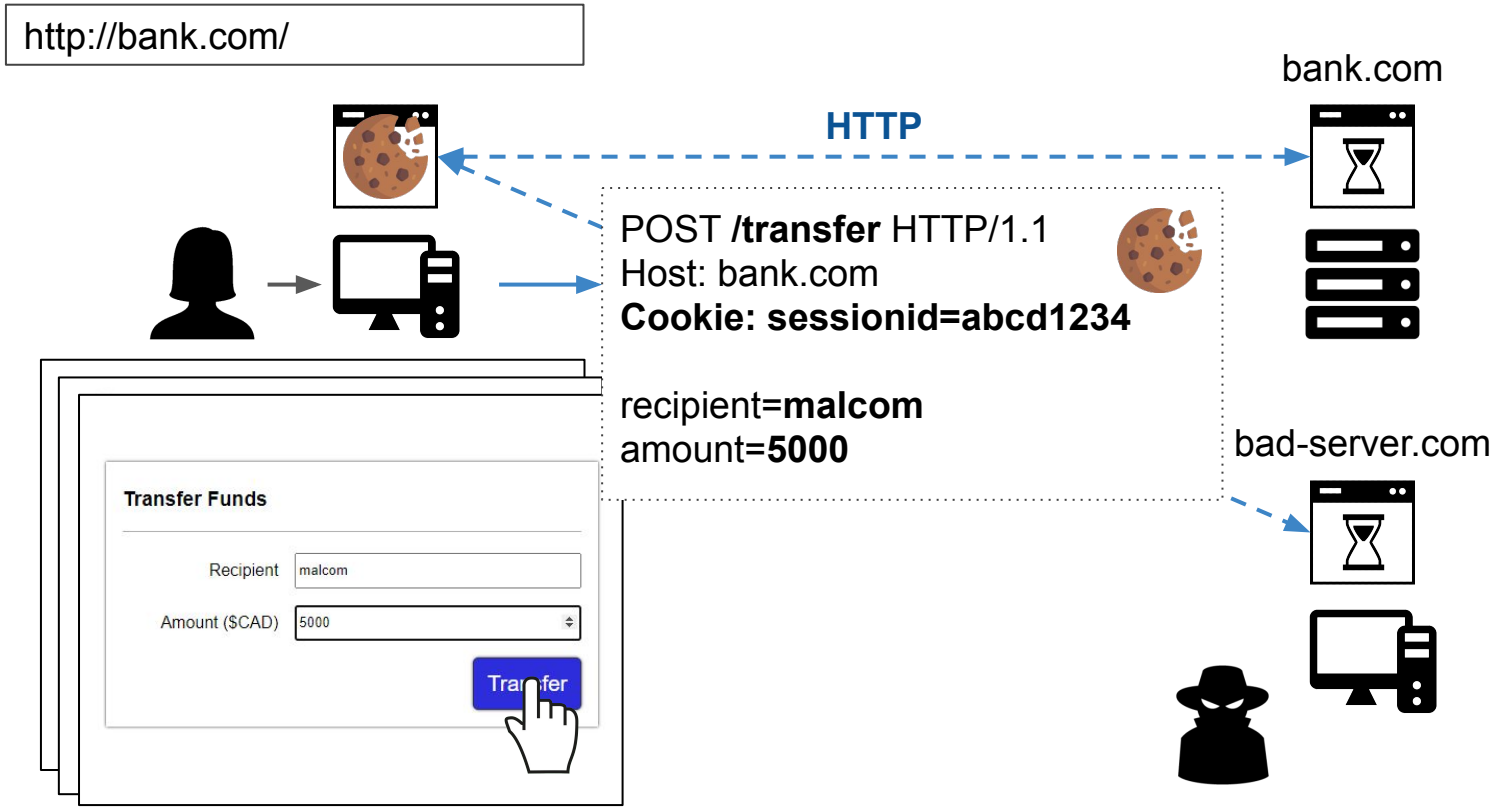
```
<form method='POST' action='http://bank.com/transfer'>
  <input name='recipient' value='malcom'/>
  <input name='amount' value='5000'/>
</form>
<script>
  window.onload = () => document.forms[0].submit()
</script>
```

-- malicious pre-filled form -->

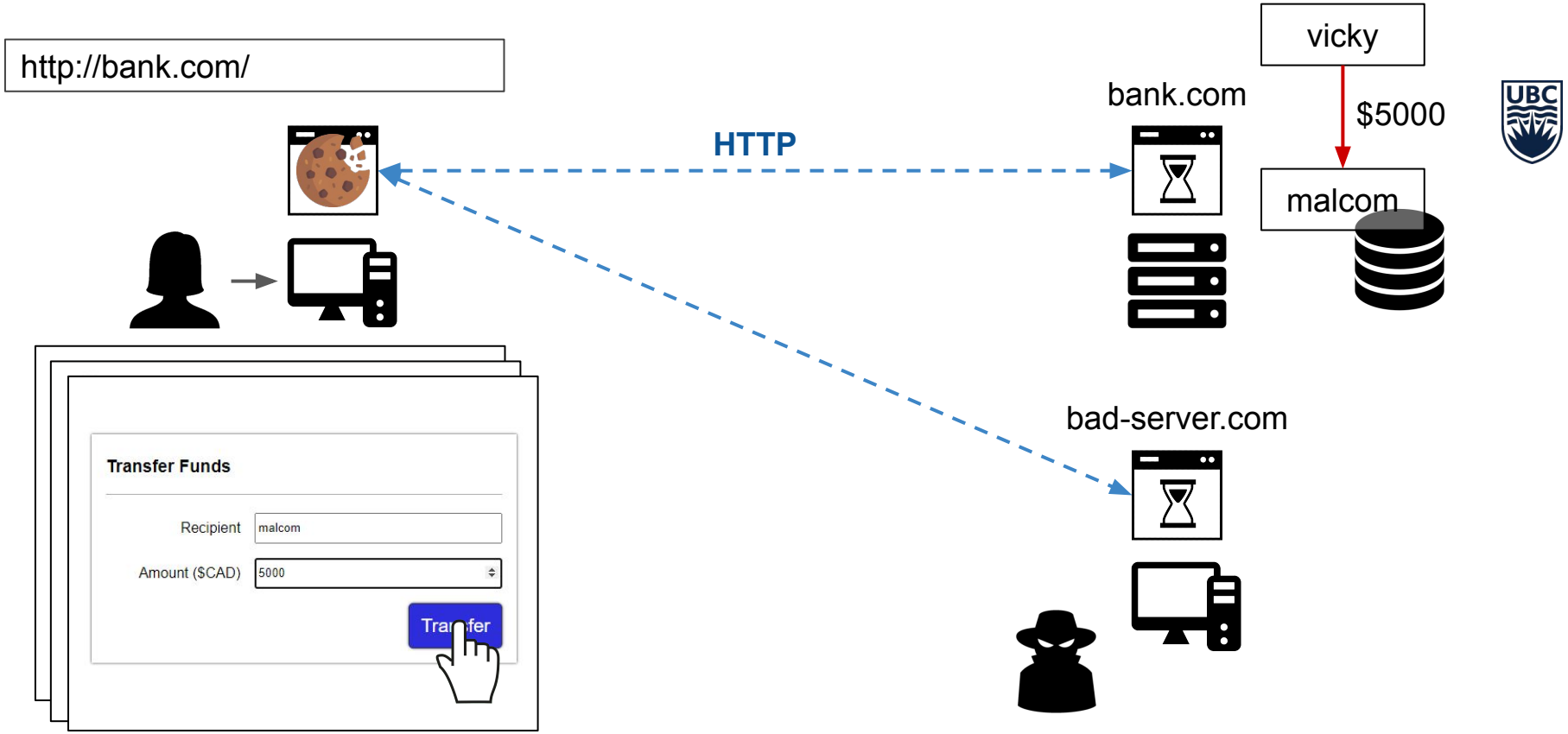
bad-server.com



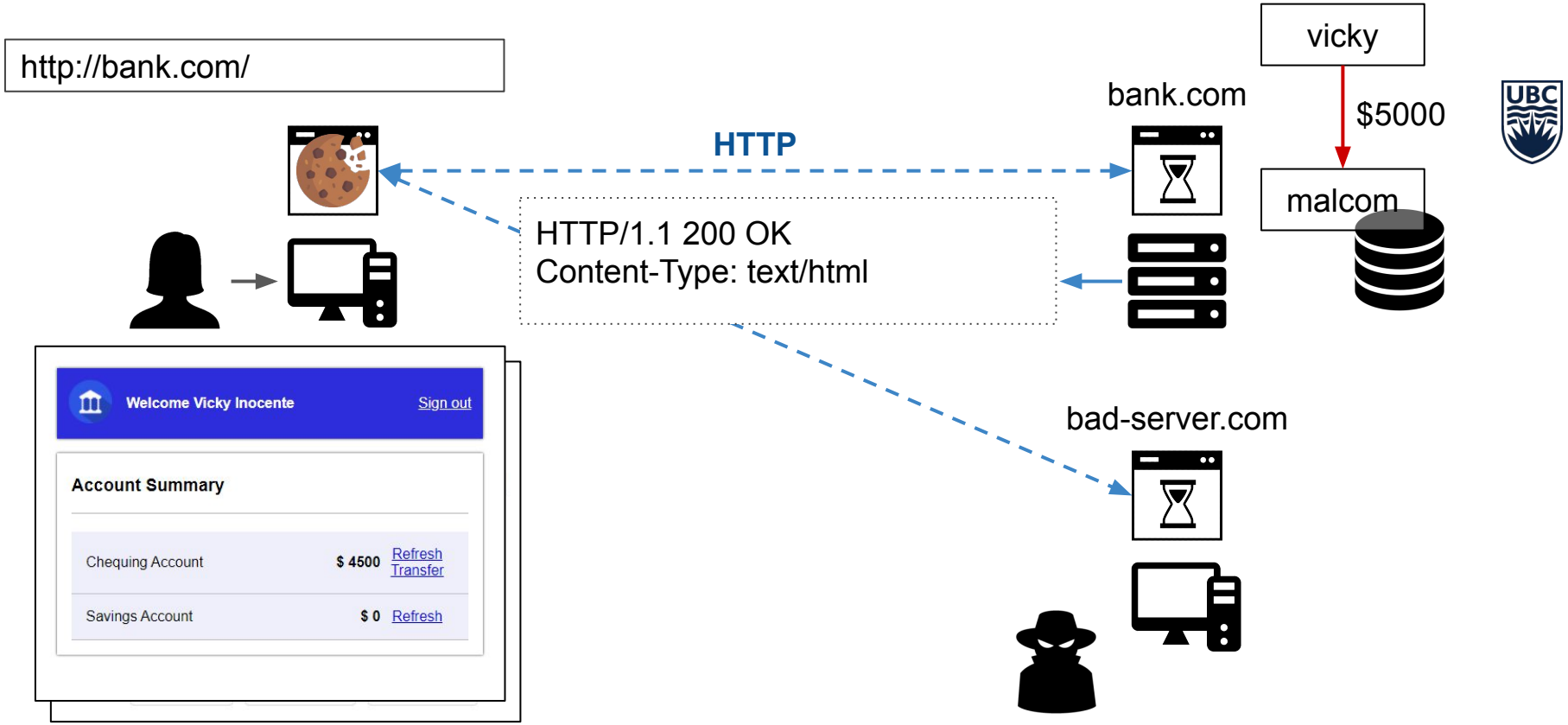
# Web Security: Cross-site Request Forgery



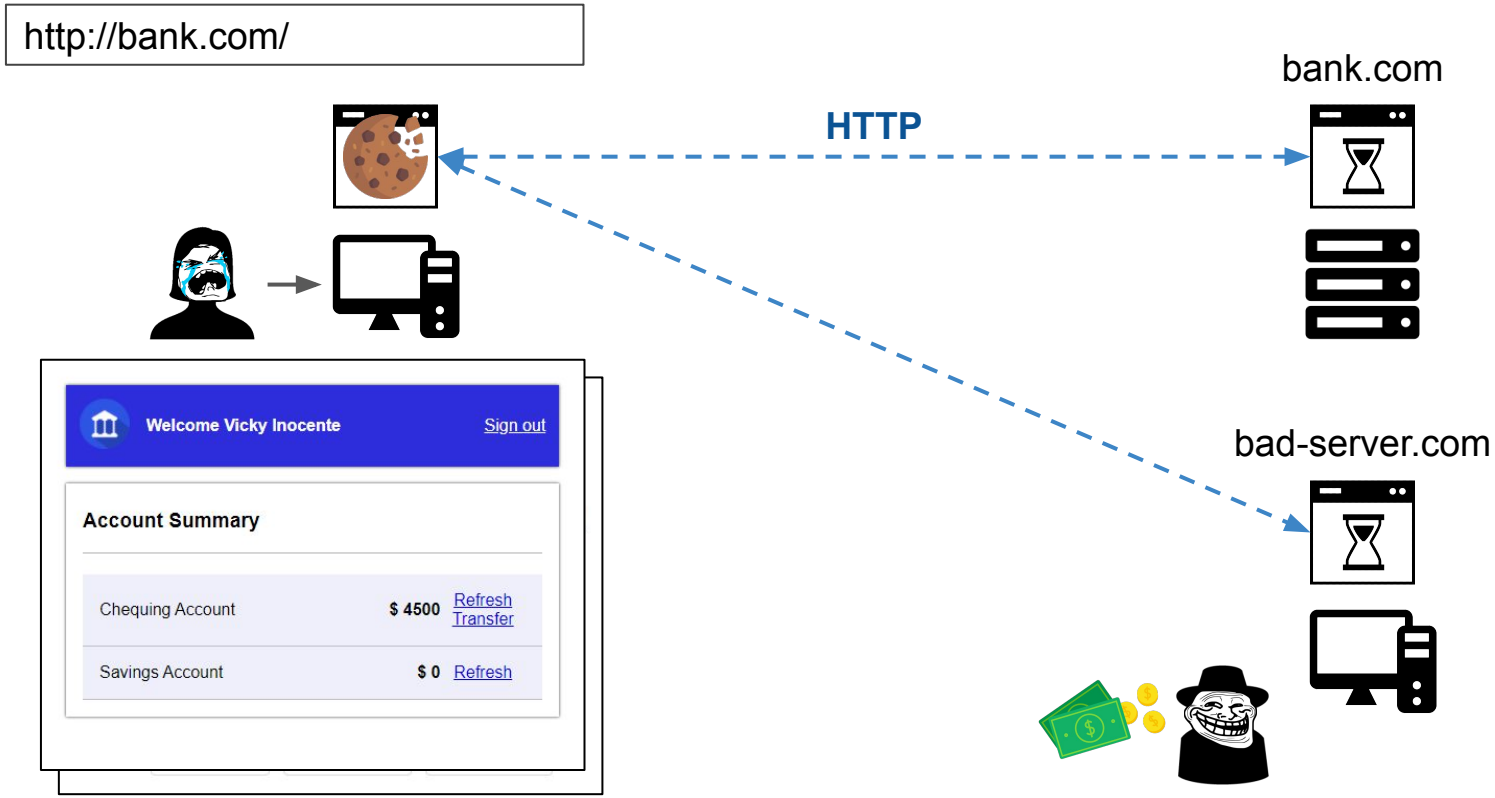
# Web Security: Cross-site Request Forgery



# Web Security: Cross-site Request Forgery



# Web Security: Cross-site Request Forgery



# Web Security: Cross-site Request Forgery

## Defense

- Make the URL hard to guess by attaching a random nonce or client-specific key to it
  - Works only if nonce/key is not leaked, and is complex
- Things that don’t work, but are often deployed
  - Using POST instead of GET requests (pointless)
  - Using multi-step transactions (makes it harder for the attacker, but they can still forge the sequence)
  - Using a secret cookie (all related cookies will be submitted with every request, even the secret ones)

## Web Security

1. Session
2. Cookie
3. Web Security