Web Application Architecture

Bernard Nongpoh

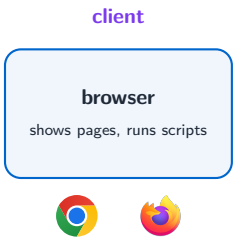
Assistant Professor, Department of CSE



ভাৰতীয় প্ৰযুক্তিবিদ্যা প্ৰতিষ্ঠান গুৱাহাটী
भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

Web Application (1/3)

What is running when you use a web application?



CERN: A Short History of the Web; Stanford CS253, L01 What is Web Security?

Web Application (2/3)

What is running when you use a web application?

client

browser

shows pages, runs scripts



server

web server

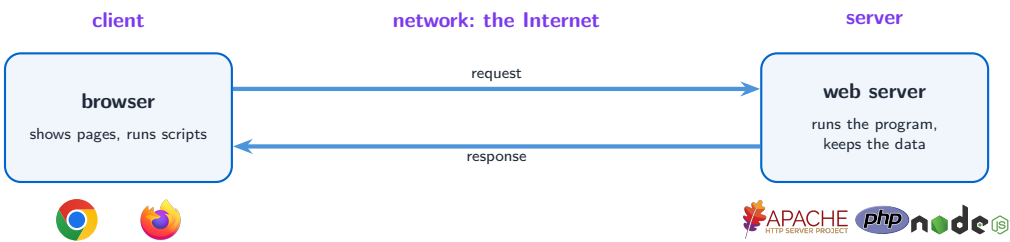
runs the program,
keeps the data



CERN: A Short History of the Web; Stanford CS253, L01 What is Web Security?

Web Application (3/3)

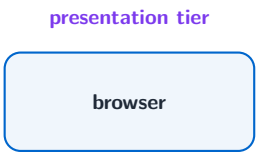
What is running when you use a web application?



CERN: A Short History of the Web; Stanford CS253, L01 What is Web Security?

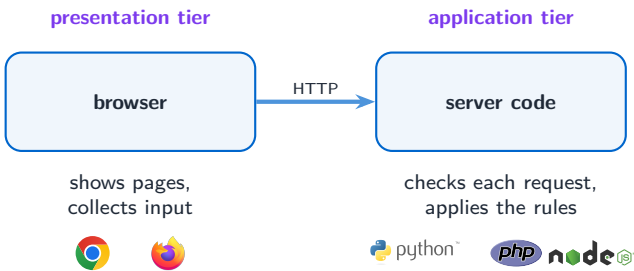
Three-Tier Architecture (1/4)

Where does the program keep its data?



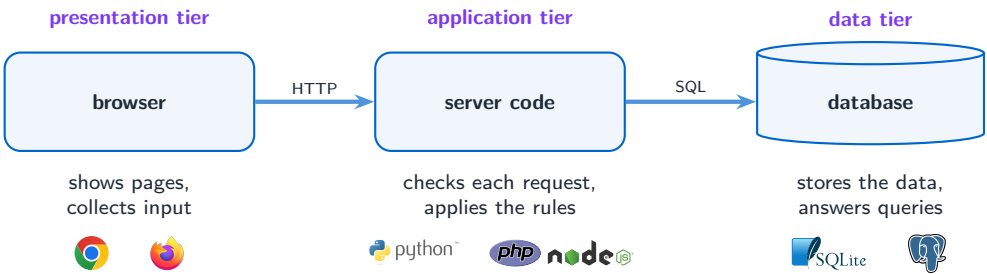
Three-Tier Architecture (2/4)

Where does the program keep its data?



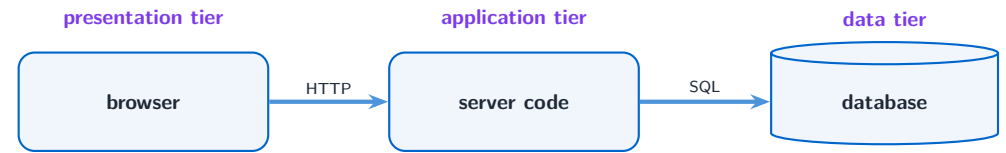
Three-Tier Architecture (3/4)

Where does the program keep its data?



Three-Tier Architecture (4/4)

Where does the program keep its data?



in Juice Shop

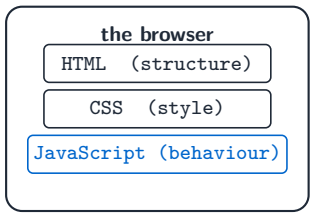
`Angular single-page app`

`Node.js + Express`

`SQLite: Users, Products, Baskets`

Client-Side Scripting (1/3)

The server sends it, but *your browser* runs it.

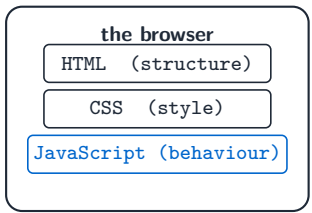


runs on the user's machine

MDN: JavaScript

Client-Side Scripting (2/3)

The server sends it, but *your browser* runs it.



runs on the user's machine

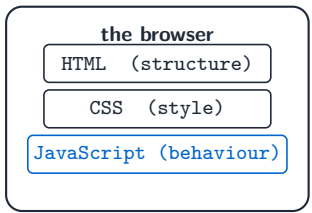
What it does

- change the page without reloading
- check a form before sending
- call APIs in the background (AJAX)
- animations, maps, editors

MDN: JavaScript

Client-Side Scripting (3/3)

The server sends it, but *your browser* runs it.



runs on the user's machine

What it does

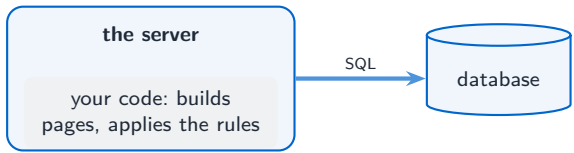
- change the page without reloading
- check a form before sending
- call APIs in the background (AJAX)
- animations, maps, editors

Because it runs on the client, the user can **read and change** it. Never trust it.

MDN: JavaScript

Server-Side Scripting (1/3)

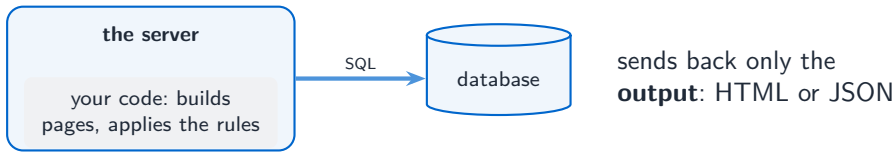
Code that runs on the server, out of the user's reach.



MDN: Server-side website programming

Server-Side Scripting (2/3)

Code that runs on the server, out of the user's reach.

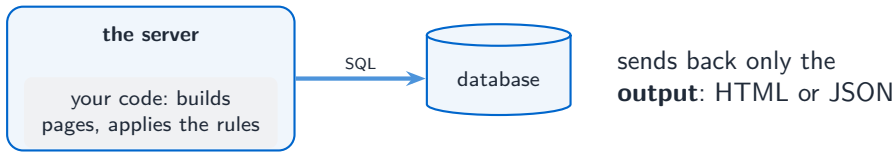


Languages: PHP, Python, Node.js, Java, Ruby. The user sees the **output**, never the code.

MDN: Server-side website programming

Server-Side Scripting (3/3)

Code that runs on the server, out of the user's reach.



Languages: PHP, Python, Node.js, Java, Ruby. The user sees the **output**, never the code. So this is where real checks live: the server decides who may do what.

MDN: Server-side website programming

Servers and Frameworks (1/6)

You do not build all this from scratch. What is available?

Web servers	Apache · Nginx · IIS

MDN: Web frameworks

Servers and Frameworks (2/6)

You do not build all this from scratch. What is available?

Web servers	Apache · Nginx · IIS
Server languages	PHP · Python · Node.js · Java · Ruby

MDN: Web frameworks

Servers and Frameworks (3/6)

You do not build all this from scratch. What is available?

Web servers	Apache · Nginx · IIS
Server languages	PHP · Python · Node.js · Java · Ruby
Server frameworks	Laravel · Django, Flask · Express · Spring · Rails

MDN: Web frameworks

Servers and Frameworks (4/6)

You do not build all this from scratch. What is available?

Web servers	Apache · Nginx · IIS
Server languages	PHP · Python · Node.js · Java · Ruby
Server frameworks	Laravel · Django, Flask · Express · Spring · Rails
Client frameworks	React · Angular · Vue

MDN: Web frameworks

Servers and Frameworks (5/6)

You do not build all this from scratch. What is available?

Web servers	Apache · Nginx · IIS
Server languages	PHP · Python · Node.js · Java · Ruby
Server frameworks	Laravel · Django, Flask · Express · Spring · Rails
Client frameworks	React · Angular · Vue
Databases	MySQL · PostgreSQL · SQLite · MongoDB

MDN: Web frameworks

Servers and Frameworks (6/6)

You do not build all this from scratch. What is available?

Web servers	Apache · Nginx · IIS
Server languages	PHP · Python · Node.js · Java · Ruby
Server frameworks	Laravel · Django, Flask · Express · Spring · Rails
Client frameworks	React · Angular · Vue
Databases	MySQL · PostgreSQL · SQLite · MongoDB













Juice Shop, our target today, is **Node.js + Express + Angular + SQLite**.

MDN: Web frameworks

What Is Web Security? (1/5)

Where in this picture can things go wrong?



Stanford CS253, L01 What is Web Security?; MDN: Same-origin policy

What Is Web Security? (2/5)

Where in this picture can things go wrong?

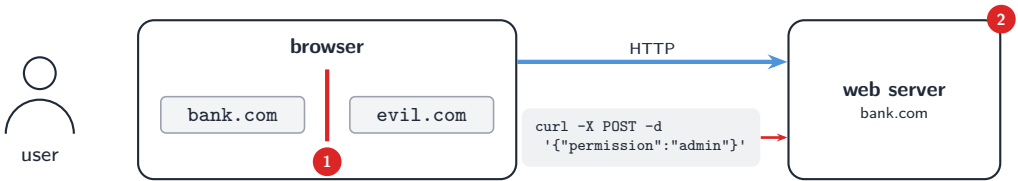


1. Browser security
sites in one browser stay apart (same-origin policy)

Stanford CS253, L01 What is Web Security?; MDN: Same-origin policy

What Is Web Security? (3/5)

Where in this picture can things go wrong?



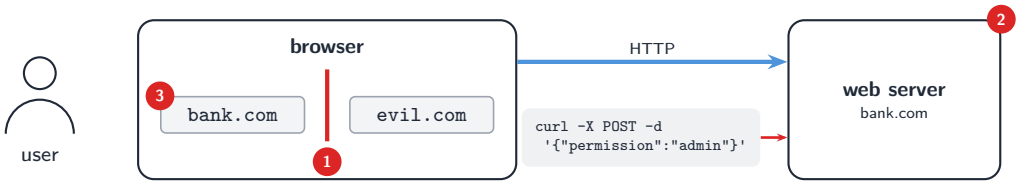
1. Browser security
sites in one browser stay apart (same-origin policy)

2. Server-side security
anyone can send the server any request

Stanford CS253, L01 What is Web Security?; MDN: Same-origin policy

What Is Web Security? (4/5)

Where in this picture can things go wrong?



1. Browser security
sites in one browser stay apart (same-origin policy)

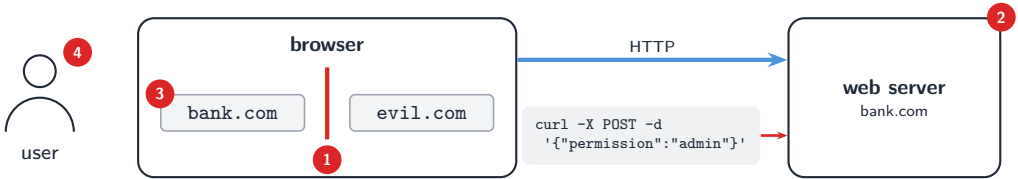
2. Server-side security
anyone can send the server any request

3. Client-side security
the user is attacked inside the app's own page

Stanford CS253, L01 What is Web Security?; MDN: Same-origin policy

What Is Web Security? (5/5)

Where in this picture can things go wrong?



1. Browser security
sites in one browser stay apart (same-origin policy)

2. Server-side security
anyone can send the server any request

3. Client-side security
the user is attacked inside the app's own page

4. Protect the user
from tricks (social engineering) and from trackers

Stanford CS253, L01 What is Web Security?; MDN: Same-origin policy

Why Is Web Security Hard? (1/3)

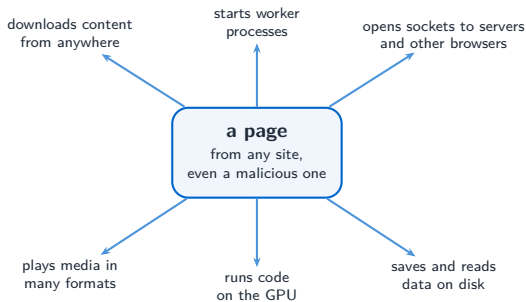
Why do web applications have so many security bugs?

- Goal: run **untrusted code** safely
- Many sites mixed in one tab
- Low-level features, hardware access
- It must also be fast
- APIs grew; never designed as a whole
- Old pages must work: "don't break the web"

Why Is Web Security Hard? (2/3)

Why do web applications have so many security bugs?

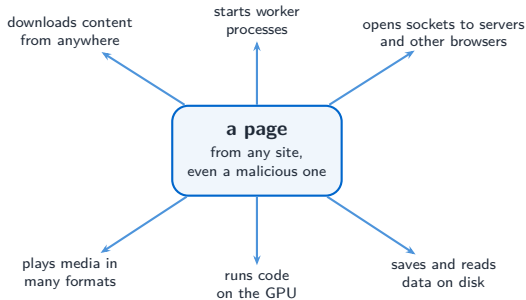
- Goal: run **untrusted code** safely
- Many sites mixed in one tab
- Low-level features, hardware access
- It must also be fast
- APIs grew; never designed as a whole
- Old pages must work: "don't break the web"



Why Is Web Security Hard? (3/3)

Why do web applications have so many security bugs?

- Goal: run **untrusted code** safely
- Many sites mixed in one tab
- Low-level features, hardware access
- It must also be fast
- APIs grew; never designed as a whole
- Old pages must work: "don't break the web"



The browser must allow all of this, and still keep **each site and the user** safe.

URL (1/7)

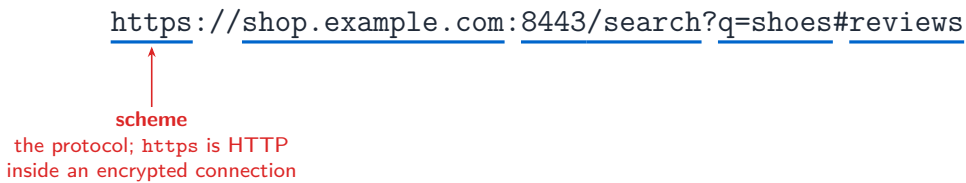
What does each part of a web address mean?

`https://shop.example.com:8443/search?q=shoes#reviews`

RFC 3986: URI Generic Syntax; Stanford CS253, L01 What is Web Security?

URL (2/7)

What does each part of a web address mean?



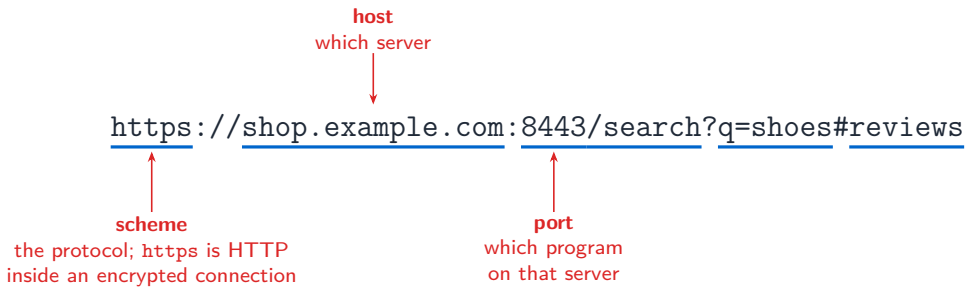
URL (3/7)

What does each part of a web address mean?



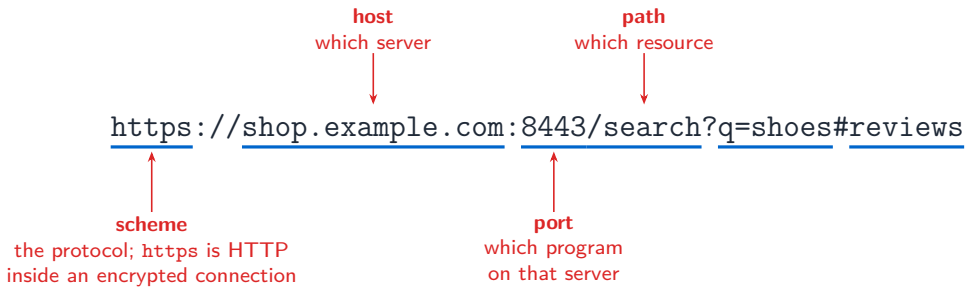
URL (4/7)

What does each part of a web address mean?



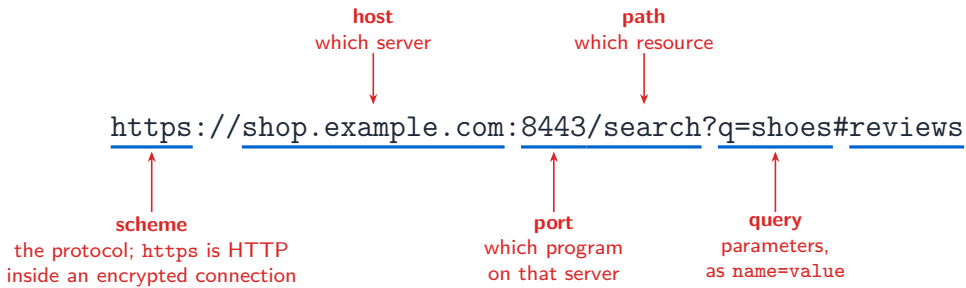
URL (5/7)

What does each part of a web address mean?



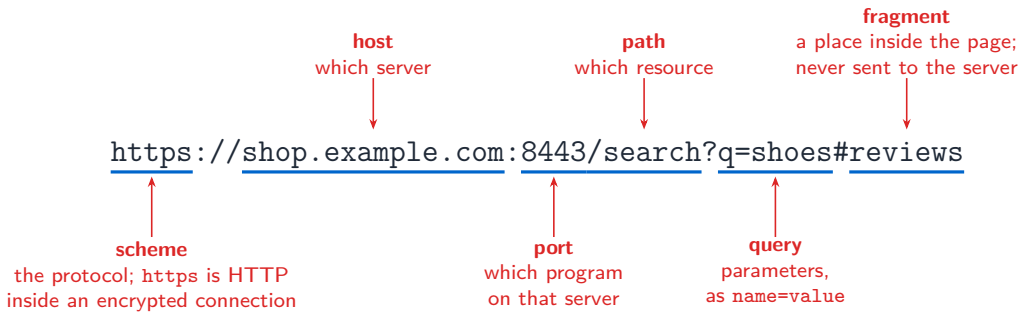
URL (6/7)

What does each part of a web address mean?



URL (7/7)

What does each part of a web address mean?



HTTP: Request and Response (1/6)

Request: browser to server

```
GET /index.html HTTP/1.1
Host: 127.0.0.1:8000
User-Agent: curl/8.7.1
Accept: */*
```

RFC 9110: HTTP Semantics; Stanford CS253, L02 DNS, HTTP

HTTP: Request and Response (2/6)

Request: browser to server

```
GET /index.html HTTP/1.1
Host: 127.0.0.1:8000
User-Agent: curl/8.7.1
Accept: */*
```

← method, path, version

RFC 9110: HTTP Semantics; Stanford CS253, L02 DNS, HTTP

HTTP: Request and Response (3/6)

Request: browser to server

```
GET /index.html HTTP/1.1
Host: 127.0.0.1:8000
User-Agent: curl/8.7.1
Accept: */*
```

← method, path, version

← headers: `name: value`

RFC 9110: HTTP Semantics; Stanford CS253, L02 DNS, HTTP

HTTP: Request and Response (4/6)

Request: browser to server

```
GET /index.html HTTP/1.1
Host: 127.0.0.1:8000
User-Agent: curl/8.7.1
Accept: */*
```

← method, path, version

← headers: name: value

Response: server to browser

```
HTTP/1.0 200 OK
Content-type: text/html
Content-Length: 93

<html>
  <body>
    <h1>Hello, Web</h1>
    ...
```

← status code

RFC 9110: HTTP Semantics; Stanford CS253, L02 DNS, HTTP

HTTP: Request and Response (5/6)

Request: browser to server

```
GET /index.html HTTP/1.1
Host: 127.0.0.1:8000
User-Agent: curl/8.7.1
Accept: */*
```

← method, path, version

← headers: name: value

Response: server to browser

```
HTTP/1.0 200 OK
Content-type: text/html
Content-Length: 93

<html>
  <body>
    <h1>Hello, Web</h1>
    ...
```

← status code

← headers (2 lines)

← empty line

← body: the page

RFC 9110: HTTP Semantics; Stanford CS253, L02 DNS, HTTP

HTTP: Request and Response (6/6)

Request: browser to server

```
GET /index.html HTTP/1.1
Host: 127.0.0.1:8000
User-Agent: curl/8.7.1
Accept: */*
```

← method, path, version

← headers: name: value

Response: server to browser

```
HTTP/1.0 200 OK
Content-type: text/html
Content-Length: 93

<html>
  <body>
    <h1>Hello, Web</h1>
    ...
```

← status code

← headers (2 lines)

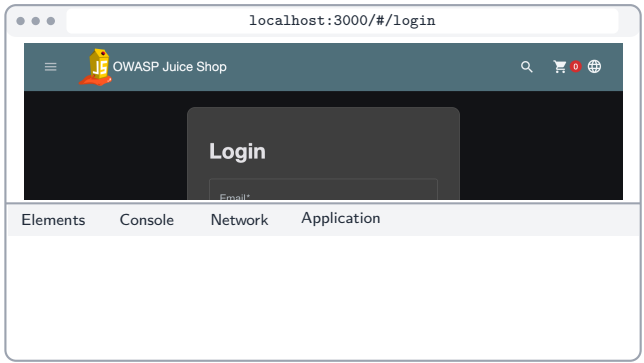
← empty line

← body: the page

Status codes: **2xx** success, **3xx** go elsewhere, **4xx** client error, **5xx** server error.

Browser Developer Tools (1/5)

Can we see this HTTP traffic in a normal browser?

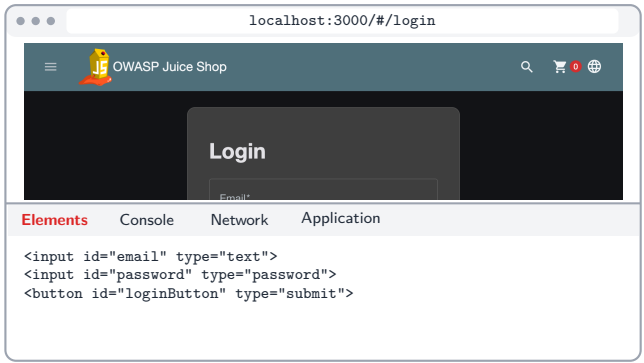


Chrome, Firefox: same tools

Chrome DevTools: Open; Stanford CS253, L02 DNS, HTTP

Browser Developer Tools (2/5)

Can we see this HTTP traffic in a normal browser?



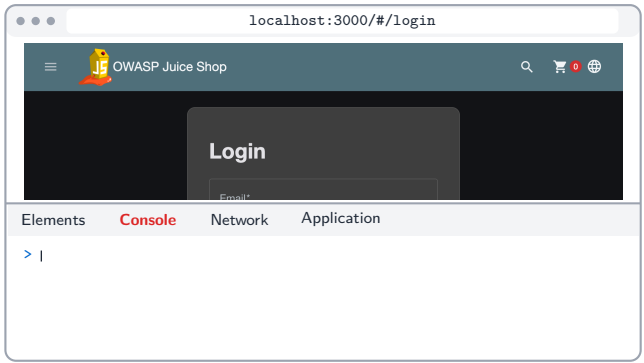
Elements: the page as a tree (DOM)

Chrome, Firefox: same tools

Chrome DevTools: Open; Stanford CS253, L02 DNS, HTTP

Browser Developer Tools (3/5)

Can we see this HTTP traffic in a normal browser?



Elements: the page as a tree (DOM)

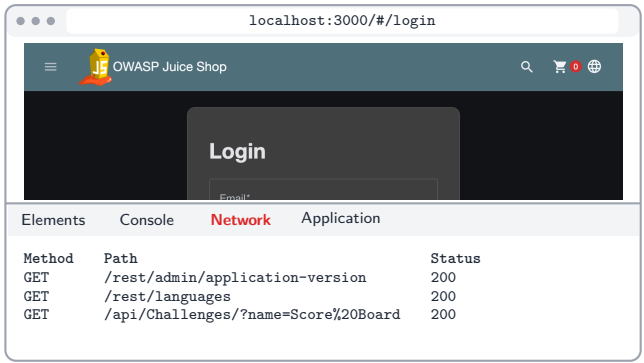
Console: run JavaScript in the page

Chrome, Firefox: same tools

Chrome DevTools: Open; Stanford CS253, L02 DNS, HTTP

Browser Developer Tools (4/5)

Can we see this HTTP traffic in a normal browser?



Elements: the page as a tree (DOM)

Console: run JavaScript in the page

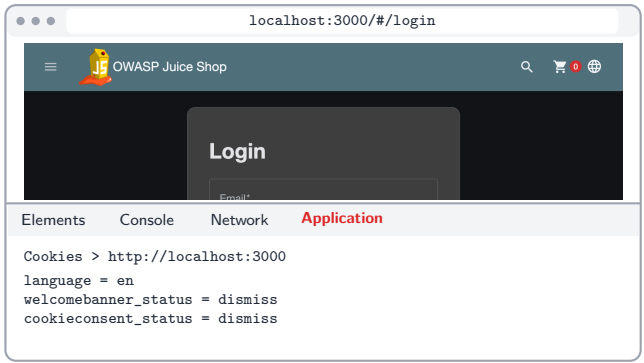
Network: every request and response

Chrome, Firefox: same tools

Chrome DevTools: Open; Stanford CS253, L02 DNS, HTTP

Browser Developer Tools (5/5)

Can we see this HTTP traffic in a normal browser?



Elements: the page as a tree (DOM)

Console: run JavaScript in the page

Network: every request and response

Application: cookies and storage

Chrome, Firefox: same tools

Chrome DevTools: Open; Stanford CS253, L02 DNS, HTTP

HTTP Is Stateless (1/5)

Each request stands alone. How does a site remember who you are?



RFC 6265: HTTP State Management Mechanism; Stanford CS253, L04 Cookies and Sessions

HTTP Is Stateless (2/5)

Each request stands alone. How does a site remember who you are?



RFC 6265: HTTP State Management Mechanism; Stanford CS253, L04 Cookies and Sessions

HTTP Is Stateless (3/5)

Each request stands alone. How does a site remember who you are?



HTTP Is Stateless (4/5)

Each request stands alone. How does a site remember who you are?



HTTP Is Stateless (5/5)

Each request stands alone. How does a site remember who you are?



Whoever sends this session id **is** alice for the server.

Cookies: Uses and Attributes (1/5)

🍪 A cookie: **small data** the browser keeps and sends back.

```
Set-Cookie: sid=a3fWa; Max-Age=2592000; Secure; HttpOnly; SameSite=Strict
```

MDN: Using HTTP cookies; RFC 6265: HTTP State Management Mechanism; Stanford CS253, L04 Cookies and Sessions

Cookies: Uses and Attributes (2/5)

A cookie: **small data** the browser keeps and sends back.

```
Set-Cookie: sid=a3fWa; Max-Age=2592000; Secure; HttpOnly; SameSite=Strict
```

Max-Age



kept **30 days**
without it: gone when the browser closes

MDN: Using HTTP cookies; RFC 6265: HTTP State Management Mechanism; Stanford CS253, L04 Cookies and Sessions

Cookies: Uses and Attributes (3/5)

A cookie: **small data** the browser keeps and sends back.

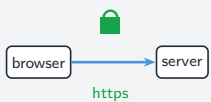
```
Set-Cookie: sid=a3fWa; Max-Age=2592000; Secure; HttpOnly; SameSite=Strict
```

Max-Age



kept **30 days**
without it: gone when
the browser closes

Secure



sent **only over HTTPS**
nobody on the
network reads it

MDN: Using HTTP cookies; RFC 6265: HTTP State Management Mechanism; Stanford CS253, L04 Cookies and Sessions

Cookies: Uses and Attributes (4/5)

A cookie: **small data** the browser keeps and sends back.

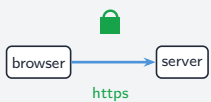
```
Set-Cookie: sid=a3fWa; Max-Age=2592000; Secure; HttpOnly; SameSite=Strict
```

Max-Age



kept **30 days**
without it: gone when the browser closes

Secure



sent **only over HTTPS**
nobody on the network reads it

HttpOnly



JavaScript cannot read it
so XSS cannot steal it

MDN: Using HTTP cookies; RFC 6265: HTTP State Management Mechanism; Stanford CS253, L04 Cookies and Sessions

Cookies: Uses and Attributes (5/5)

A cookie: **small data** the browser keeps and sends back.

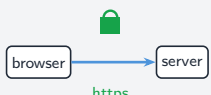
```
Set-Cookie: sid=a3fWa; Max-Age=2592000; Secure; HttpOnly; SameSite=Strict
```

Max-Age



kept **30 days**
without it: gone when the browser closes

Secure



sent **only over HTTPS**
nobody on the network reads it

HttpOnly



JavaScript cannot read it
so XSS cannot steal it

SameSite=Strict



not sent when the request comes from another site

MDN: Using HTTP cookies; RFC 6265: HTTP State Management Mechanism; Stanford CS253, L04 Cookies and Sessions

Quiz: Requests and Sessions (1/4)

Fill in each answer.

1. Which part of `http://a.com/p?x=1#top` does the server never see?

2. A server answers `404`. Who made the mistake?

3. After login, what tells the server who you are?

RFC 9110: HTTP Semantics

Quiz: Requests and Sessions (2/4)

Fill in each answer.

1. Which part of `http://a.com/p?x=1#top` does the server never see?

`#top`

2. A server answers `404`. Who made the mistake?

3. After login, what tells the server who you are?

RFC 9110: HTTP Semantics

Quiz: Requests and Sessions (3/4)

Fill in each answer.

1. Which part of `http://a.com/p?x=1#top` does the server never see? `#top`

2. A server answers `404`. Who made the mistake? `the client`

3. After login, what tells the server who you are? _____

RFC 9110: HTTP Semantics

Quiz: Requests and Sessions (4/4)

Fill in each answer.

1. Which part of `http://a.com/p?x=1#top` does the server never see? `#top`

2. A server answers `404`. Who made the mistake? `the client`

3. After login, what tells the server who you are? `the cookie`

RFC 9110: HTTP Semantics

JavaScript and the DOM (1/4)

What if a page carries its own program?

```
<h1 id="t">Hello</h1>
<script>
  let h = document.getElementById("t");
  h.textContent = "Hi, alice";
</script>
```

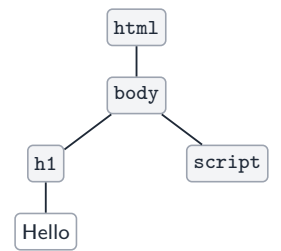
MDN: JavaScript; Stanford CS253, L01 What is Web Security?

JavaScript and the DOM (2/4)

What if a page carries its own program?

```
1 <h1 id="t">Hello</h1>
2 <script>
3   let h = document.getElementById("t");
4   h.textContent = "Hi, alice";
5 </script>
```

the DOM: the page as a tree

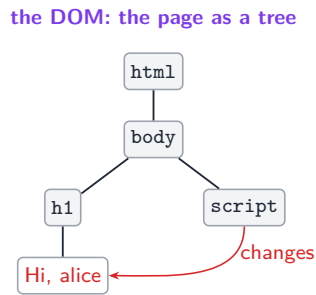


MDN: JavaScript; Stanford CS253, L01 What is Web Security?

JavaScript and the DOM (3/4)

What if a page carries its own program?

```
1 <h1 id="t">Hello</h1>
2 <script>
3   let h = document.getElementById("t");
4   h.textContent = "Hi, alice";
5 </script>
```

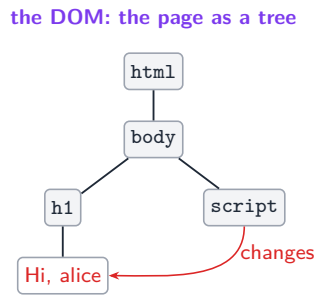


MDN: JavaScript; Stanford CS253, L01 What is Web Security?

JavaScript and the DOM (4/4)

What if a page carries its own program?

```
1 <h1 id="t">Hello</h1>
2 <script>
3   let h = document.getElementById("t");
4   h.textContent = "Hi, alice";
5 </script>
```



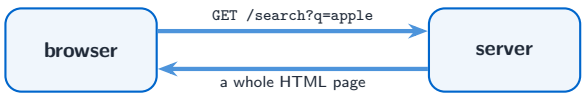
A script can read and change **everything in its page**, and read its cookies: `document.cookie`.

MDN: JavaScript; Stanford CS253, L01 What is Web Security?

AJAX and Single-Page Applications (1/3)

Every click loads a whole new page. Can a page ask for data only?

classic: one page per click



response body

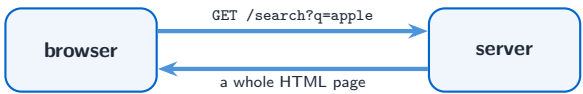
```
<h1>Results for apple</h1>
<li>Apple Juice (1000ml)</li>
```

Garrett, *Ajax: A New Approach to Web Applications*, 2005

AJAX and Single-Page Applications (2/3)

Every click loads a whole new page. Can a page ask for data only?

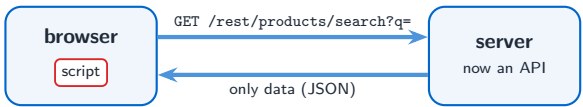
classic: one page per click



response body

```
<h1>Results for apple</h1>
<li>Apple Juice (1000ml)</li>
```

AJAX: the script asks in the background



response body

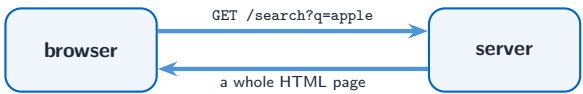
```
{"status":"success","data":[
 {"id":1,"name":"Apple Juice (1000ml)",
  ...}, ...]}
```

Garrett, *Ajax: A New Approach to Web Applications*, 2005

AJAX and Single-Page Applications (3/3)

Every click loads a whole new page. Can a page ask for data only?

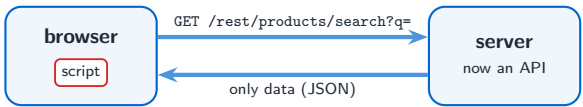
classic: one page per click



response body

```
<h1>Results for apple</h1>
<li>Apple Juice (1000ml)</li>
```

AJAX: the script asks in the background



response body

```
{"status":"success","data":[
 {"id":1,"name":"Apple Juice (1000ml)",
  ...}, ...]}
```

Single-page application: one HTML page, then only data from an API.

Garrett, *Ajax: A New Approach to Web Applications*, 2005

Trust Boundaries (1/5)

Which side of the picture can you trust?

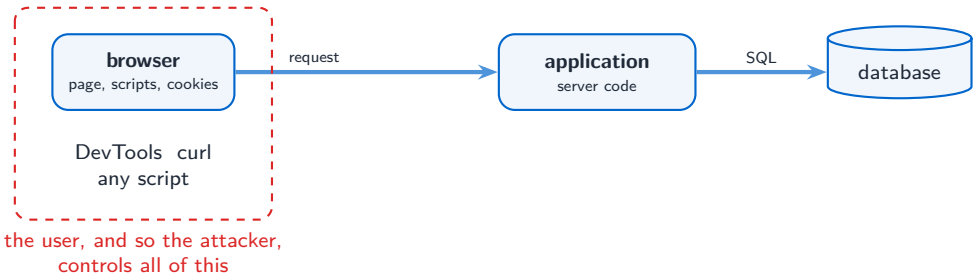


DevTools curl
any script

OWASP: Attack Surface Analysis Cheat Sheet; Stanford CS253, L01 What is Web Security?

Trust Boundaries (2/5)

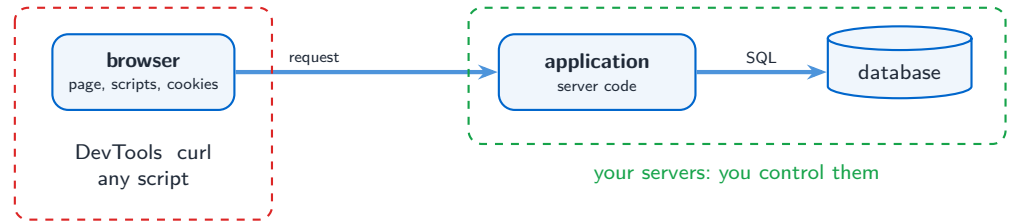
Which side of the picture can you trust?



OWASP: Attack Surface Analysis Cheat Sheet; Stanford CS253, L01 What is Web Security?

Trust Boundaries (3/5)

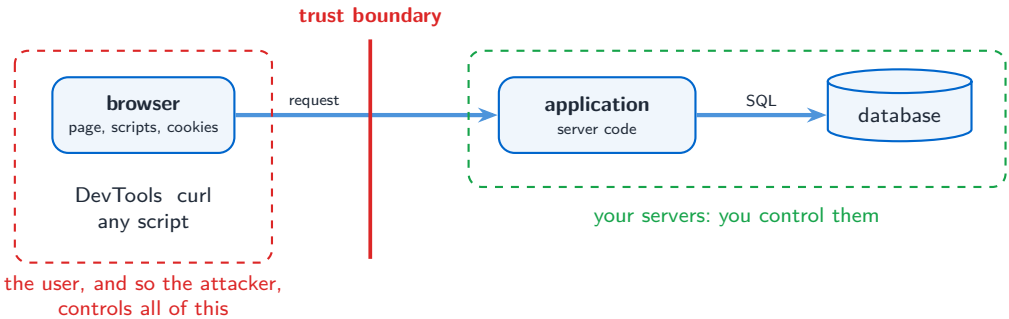
Which side of the picture can you trust?



OWASP: Attack Surface Analysis Cheat Sheet; Stanford CS253, L01 What is Web Security?

Trust Boundaries (4/5)

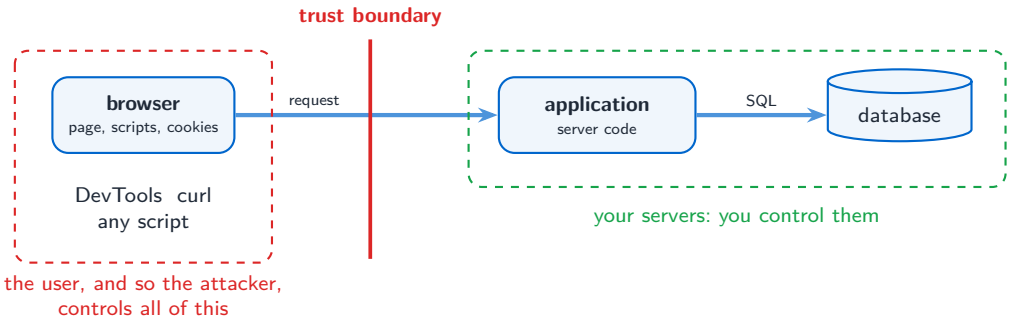
Which side of the picture can you trust?



OWASP: Attack Surface Analysis Cheat Sheet; Stanford CS253, L01 What is Web Security?

Trust Boundaries (5/5)

Which side of the picture can you trust?

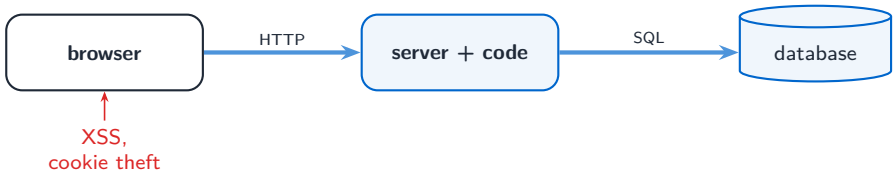


Rule: **check all data where it crosses the boundary**, on the server side.

OWASP: Attack Surface Analysis Cheat Sheet; Stanford CS253, L01 What is Web Security?

Attack Surface: Where Each Part Breaks (1/5)

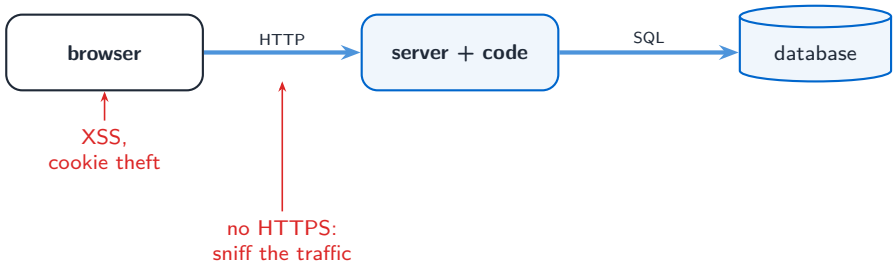
Every component we just saw is also a place to attack.



OWASP: Attack Surface Analysis

Attack Surface: Where Each Part Breaks (2/5)

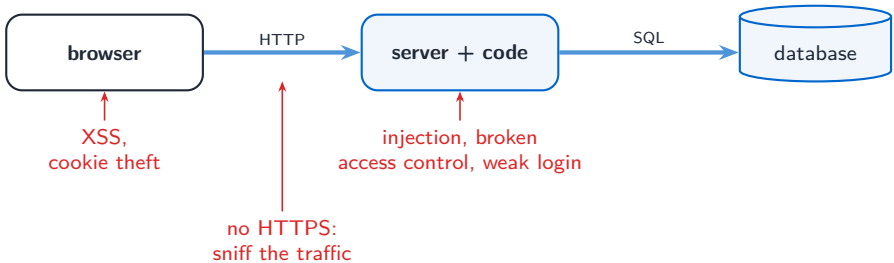
Every component we just saw is also a place to attack.



OWASP: Attack Surface Analysis

Attack Surface: Where Each Part Breaks (3/5)

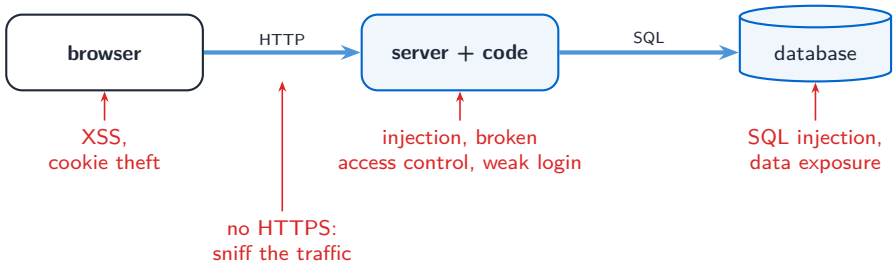
Every component we just saw is also a place to attack.



OWASP: Attack Surface Analysis

Attack Surface: Where Each Part Breaks (4/5)

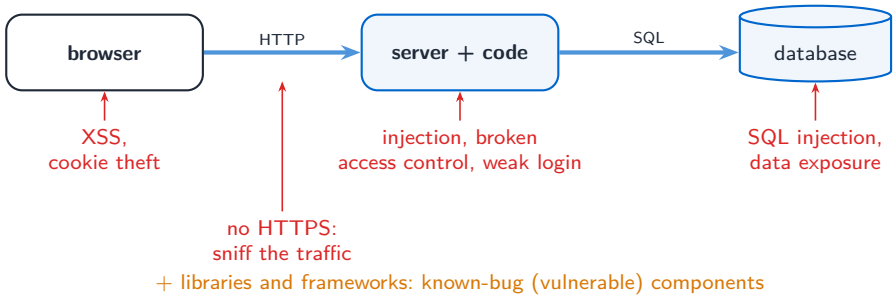
Every component we just saw is also a place to attack.



OWASP: Attack Surface Analysis

Attack Surface: Where Each Part Breaks (5/5)

Every component we just saw is also a place to attack.



The rest of today attacks each one: the server, the database, the browser, the session.

OWASP: Attack Surface Analysis

Quiz: Who Controls It? (1/4)

Can the attacker change it?

1. A hidden form field, `price=499`
2. The `User-Agent` header of a request
3. The session table on the server

OWASP: Attack Surface Analysis Cheat Sheet

Quiz: Who Controls It? (2/4)

Can the attacker change it?

1. A hidden form field, `price=499` **YES**
2. The `User-Agent` header of a request
3. The session table on the server

OWASP: Attack Surface Analysis Cheat Sheet

Quiz: Who Controls It? (3/4)

Can the attacker change it?

1. A hidden form field, `price=499` **YES**
2. The `User-Agent` header of a request **YES**
3. The session table on the server

OWASP: Attack Surface Analysis Cheat Sheet

Quiz: Who Controls It? (4/4)

Can the attacker change it?

1. A hidden form field, `price=499` **YES**
2. The `User-Agent` header of a request **YES**
3. The session table on the server **NO**

OWASP: Attack Surface Analysis Cheat Sheet

Responsible Disclosure (1/3)

You can now read and change any request. What may you test?

- Test only systems you **own**, or have permission to test
- Never destroy or change other people's data

Responsible Disclosure (2/3)

You can now read and change any request. What may you test?

- Test only systems you **own**, or have permission to test
- Never destroy or change other people's data
- Found a bug? Tell the **owner** privately, for example through its bug bounty program
- Give the owner time to fix it before you talk about it

Stanford CS253, L01 What is Web Security?; CERT-In: Responsible Vulnerability Disclosure and Coordination Policy

Responsible Disclosure (3/3)

You can now read and change any request. What may you test?

- Test only systems you **own**, or have permission to test
- Never destroy or change other people's data
- Found a bug? Tell the **owner** privately, for example through its bug bounty program
- Give the owner time to fix it before you talk about it



Stanford CS253, L01 What is Web Security?; CERT-In: Responsible Vulnerability Disclosure and Coordination Policy