Software Security 10

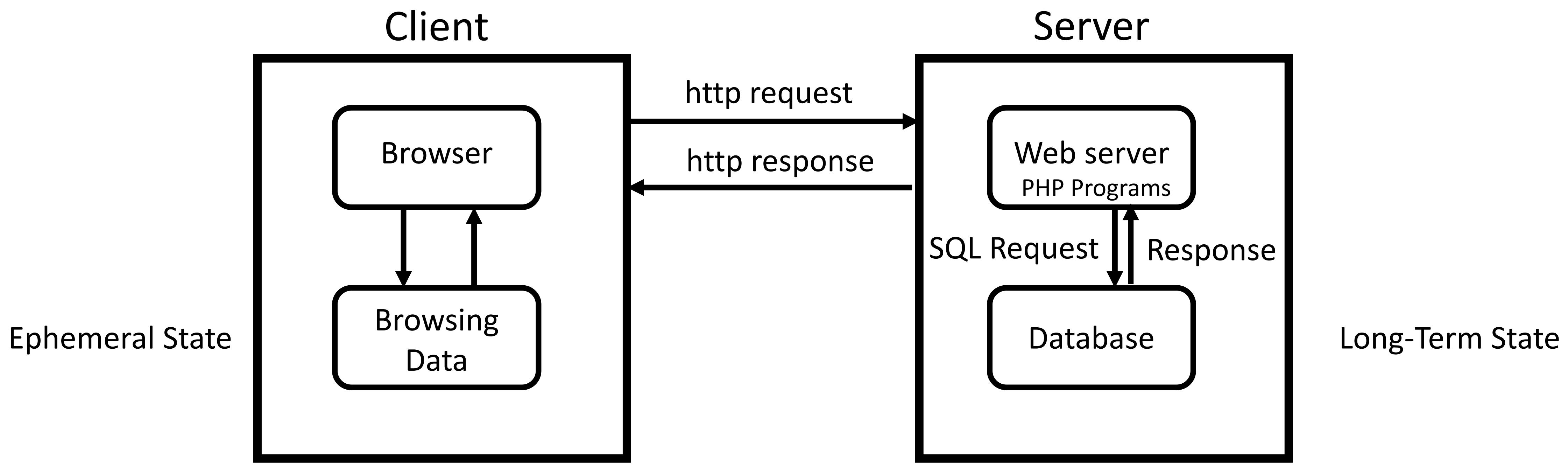
Sven Schäge

Learning Goals

- Understand why and how HTTP is augmented to save state information in the websurfing scenario.
- Be able to describe and compare the two mechanisms to realize ephemeral states: hidden from fields and cookies. Be able to reflect on their benefits and disadvantages.
- Understand and critically reflect on the usage of cookies in ad networks.
- Be able to develop why stealing cookies is a serious security risk although they are not cryptographic keys in a classical sense. Be able to argue about the effectiveness of general damage control mechanisms.
- Understand the concept of CSRF attacks, its technical details, and practical significance.
- Be able to give effective countermeasures.
- Be able explain and critically reflect on the Same-Origin Policy.
- Understand the concept of XSS attacks, its technical details, andpractical significance.
- Differentiate between different types of XSS attacks and provide fitting countermeasures.

HTTP and States

Websurfing

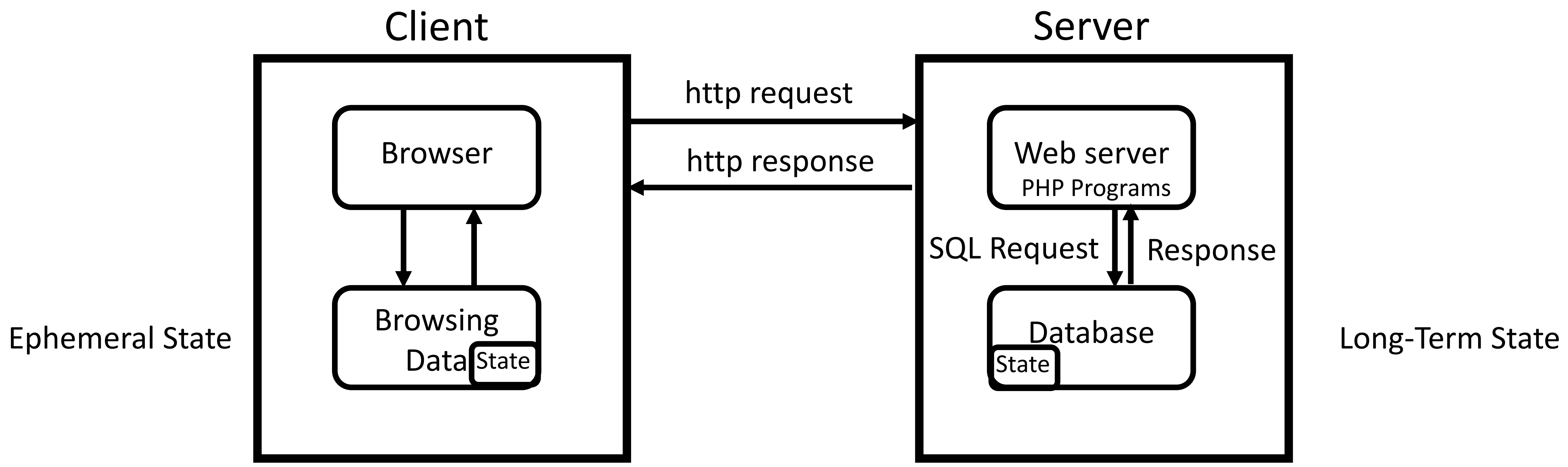

Client
Server
Browser
Browsing Data
http request
http response
Web server
PHP Programs
SQL Request
Response
Database
Ephemeral State
Long-Term State

HTTP is Stateless

- HTTP is a stateless protocol
- It essentially consists of an http request of the browser to the webserver, which in turn responds with an HTTP response

- In general, having no state can be beneficial to avoid denial-of-service attacks
- However, there is per se no means of enabling multi-move communication with selected partners
    - In theory, when relying on authentication and while using only basic HTTP, the client has to authenticate for every HTTP request anew

How Does HTTP Keep Track of User Identities?



Ephemeral State

- Ephemeral states have to be sent to the server every time the client sends an HTTP request
- They can be thought of as the last working state that the server had in the communication with that client
- The server just stores its state at the client and can so maintain lightweight memory usage
    - At the cost of more communication
- Ephemeral state helps server to keep track of client
- In general: ephemeral state can be record of entire history of past communication between client and server
- Example: all state information that was generated after client successfully passed login and so authenticated to server => no second login required
- Two realizations
    - Hidden fields in HTML document
    - Cookies

Hidden Form Fields

- Hidden form fields are elements in a HTML file that are not rendered by the browser
- Format:
  - <input type=“hidden” name=“current ephemeral state” value =“…”>
- To use them as ways to maintain ephemeral states, the server embeds these fields into each HTML file it returns
- When the client reacts to these HTML files, e.g. by clicking a button, the field value is also returned to the server
- Concrete example in online shop:
  - <input type=“hidden” name=“price” value =“10”>

Basic Hidden Form Fields

- Security Problem: Client can modify hidden form fields!
  - In example: modify price to be paid!
- Solution
  - Server maintains critical information
  - Client only stores handle to that information: often called capability
  - Handle is large random number => unlikely to guess a real capability
  - <input type=“hidden” name=“id” value =“23999873”>
  - Server maintains table that relates ids to real ephemeral state information (= price information in our example)

Problems with Hidden Form Fields

- Per page administration is expensive
- After browser closes, all information lost

Cookies

- Cookies are small files that represent a handle to a trusted state stored at the server
- Cookies are generated by servers and sent to clients as ephemeral state
- Clients additionally return the cookie whenever they make a request to the server
- Since cookies are stored on the hard drive, the are available even after the browser has closed and reopened
- Stored cookies can have options:
  - a domain it should be returned to
  - expiry date
  - Specifiers of subdirectories of the domain that may access the cookie

Cookies

- Stores Personal Information
  - Can be used for tracking
- Stores Authentication Information
  - Session Identifier
- Can also hold information that is bound to a domain not a session.
  - Can store information that relates to several communication sessions over a timespan!

Ad Networks and Third Party Cookies

- Ad networks store cookies on user machines that essentially map to a list of all visited sites
- Web-services embed ad networks on their websites and so allow the ad network to read and update their cookie
- The web-service is incentivized since it obtains money based on how many times the ad network has been loaded over its site

Stealing Cookies

- In a security context, cookies are often used to keep track of users that have already successfully authenticated themselves
- Cookies that act as session identifier decide on whether a client is deemed as authenticated or not
- Naturally, attackers are interested in stealing these cookies
- Attackers could try to
    - steal cookies on either client or server
    - create valid new cookies like the server (can be made hard)
    - observe or manipulate network to obtain cookie (solution: use TLS)

Damage Control

- Expiry Date (make cookies depend on time data)
- Deletion of cookies after session ends

Cross-Site Request Forgery (CSRF)

CSRF

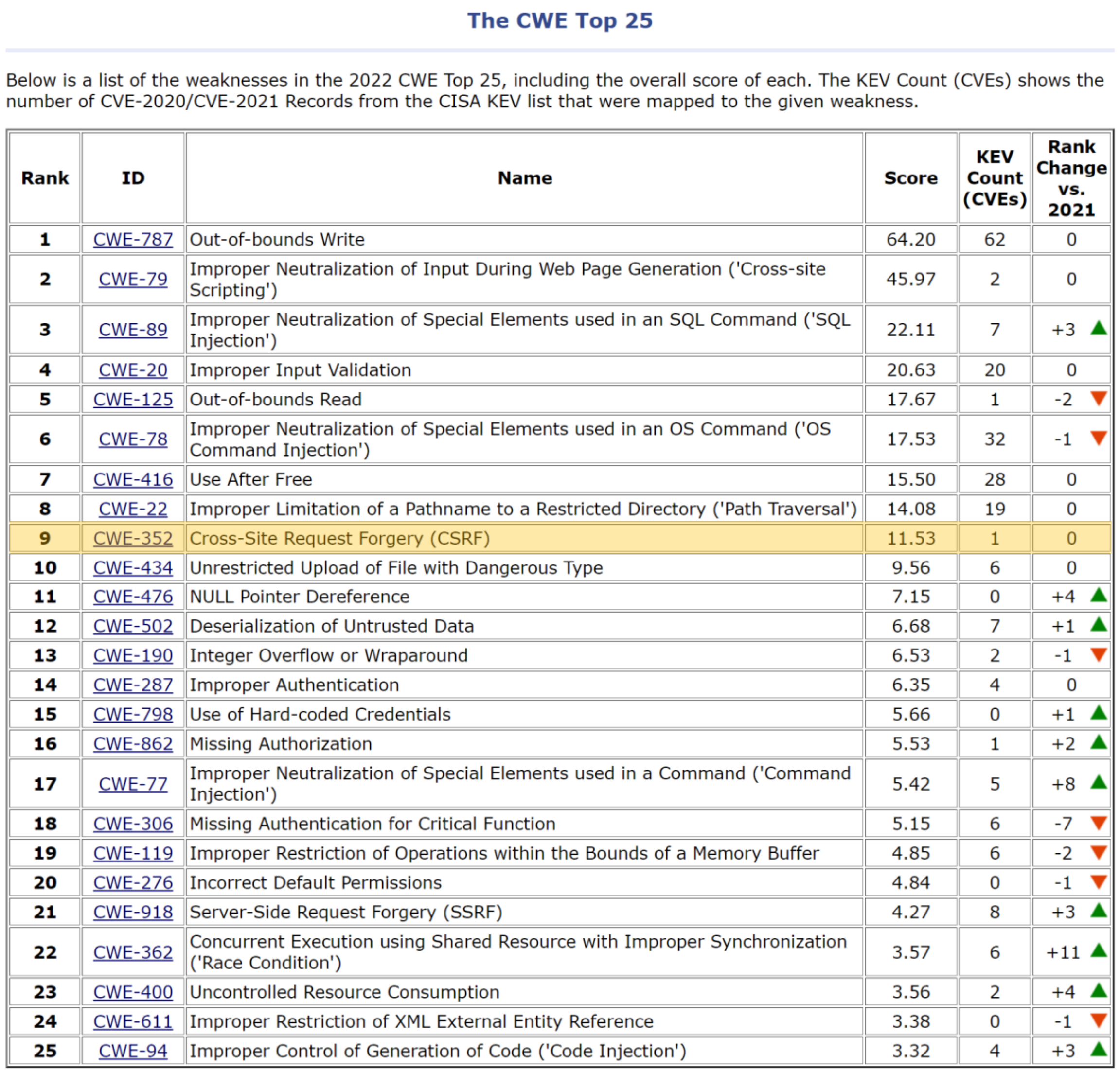
- Attacker makes client make HTTP request to other web-service
  - Client does not notice - or too late
  - Web-service believes client has willingly made request (believes in browser)
  - Attacker can launch attack by making client do what attacker would not be allowed to
  - Attack only works if just visiting a website will result in some action that changes security-critical (long-term) state information

- Recall: GET HTTP request method in general considered to be read-only!
- Bad design if otherwise! Assume, as bad example, online bank where bank transaction can be made by appropriate URL only (with parameters) using GET
  - Example: attacker makes client Alice send transaction request to her bank-site (that Alice is currently logged) in to transfer money to attacker
  - Attacker could use such an URL in a HTML tag field for an image
  - Images are loaded automatically

Additional Countermeasures

- Add additional authentication data into requests
- Require renewed authentication for any state-modifying requests.

Significance



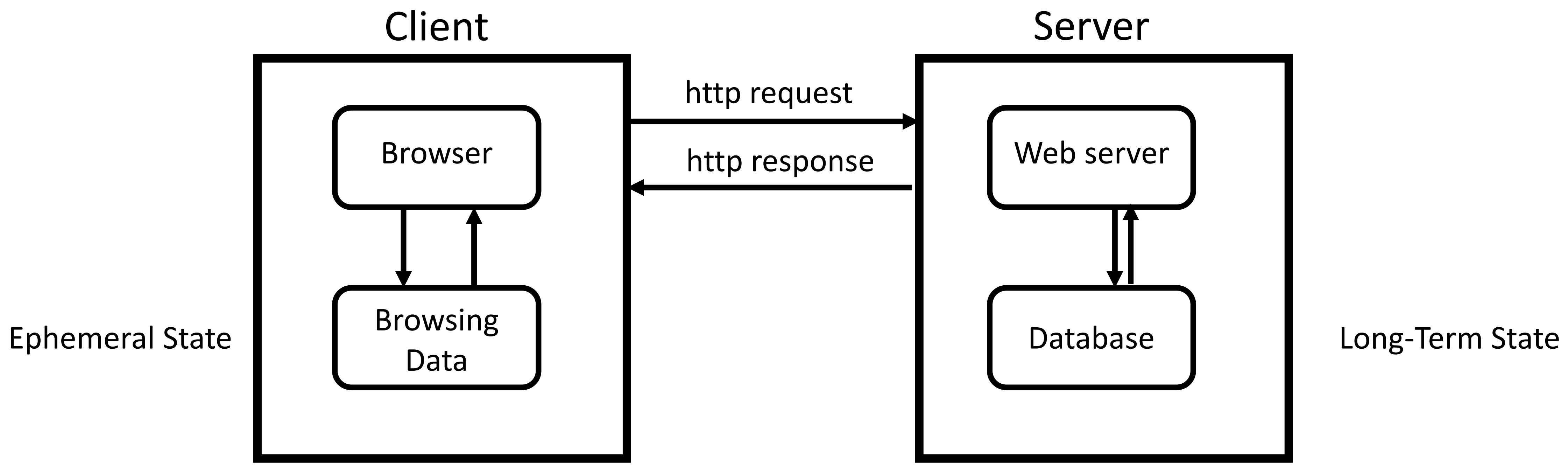
The CWE Top 25

Below is a list of the weaknesses in the 2022 CWE Top 25, including the overall score of each. The KEV Count (CVEs) shows the number of CVE-2020/CVE-2021 Records from the CISA KEV list that were mapped to the given weakness.

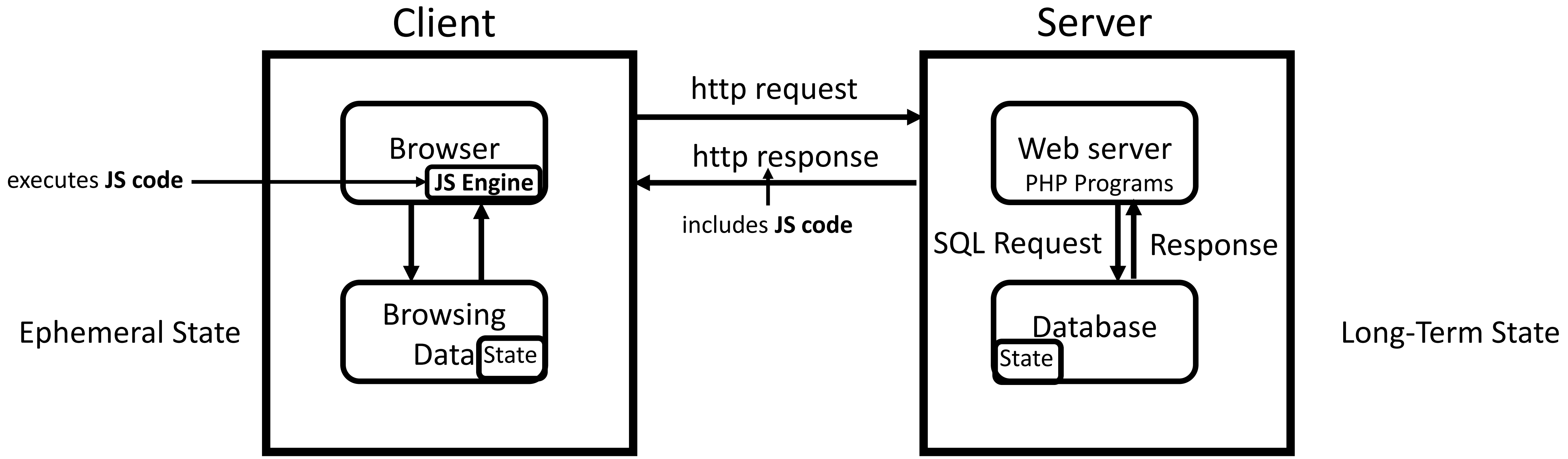
Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

JavaScript (JS) and the Same Origin Policy

Websurfing


Client
Server
Browser
Browsing Data
http request
http response
Web server
Database
Ephemeral State
Long-Term State

JavaScript: Code and Execution



JavaScript vs PHP

- JavaScript is another way, besides static and dynamic webpages, to create webpages
- Like dynamic webpages that use the combination of PHP and databases on the server to create HTML pages, JS can also be used to create HTML pages where certain parts are only decided on at runtime
- However, in contrast to HTML pages that are generated dynamically on the server, JS pages are generated on the client, usually by the browser
- Client-side scripting is considerably faster than server-side scripting since there are no transfer times

JS Code

- JS code can be embedded in a HTML file

```
<html>
 <body>
  Some HTML code
  <p>
  <script>
        var x = 3;
        document.write("First Integer: ", x, "<br>");
        var y = 5;
        document.write("Second Integer: ", y, "<br>");
        document.write("Product: ", x*y, "</p>" );
  </script>
  Some additional HTML code
 </body>
</html>
```

```
Some HTML code

First Integer: 3
Second Integer: 5
Product: 15

Some additional HTML code
```

- JS code can also be located in a separate (possibly) remote file

```
…
<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.11/lodash.js">
</script>
…
```

Executing unknown code on my computer?

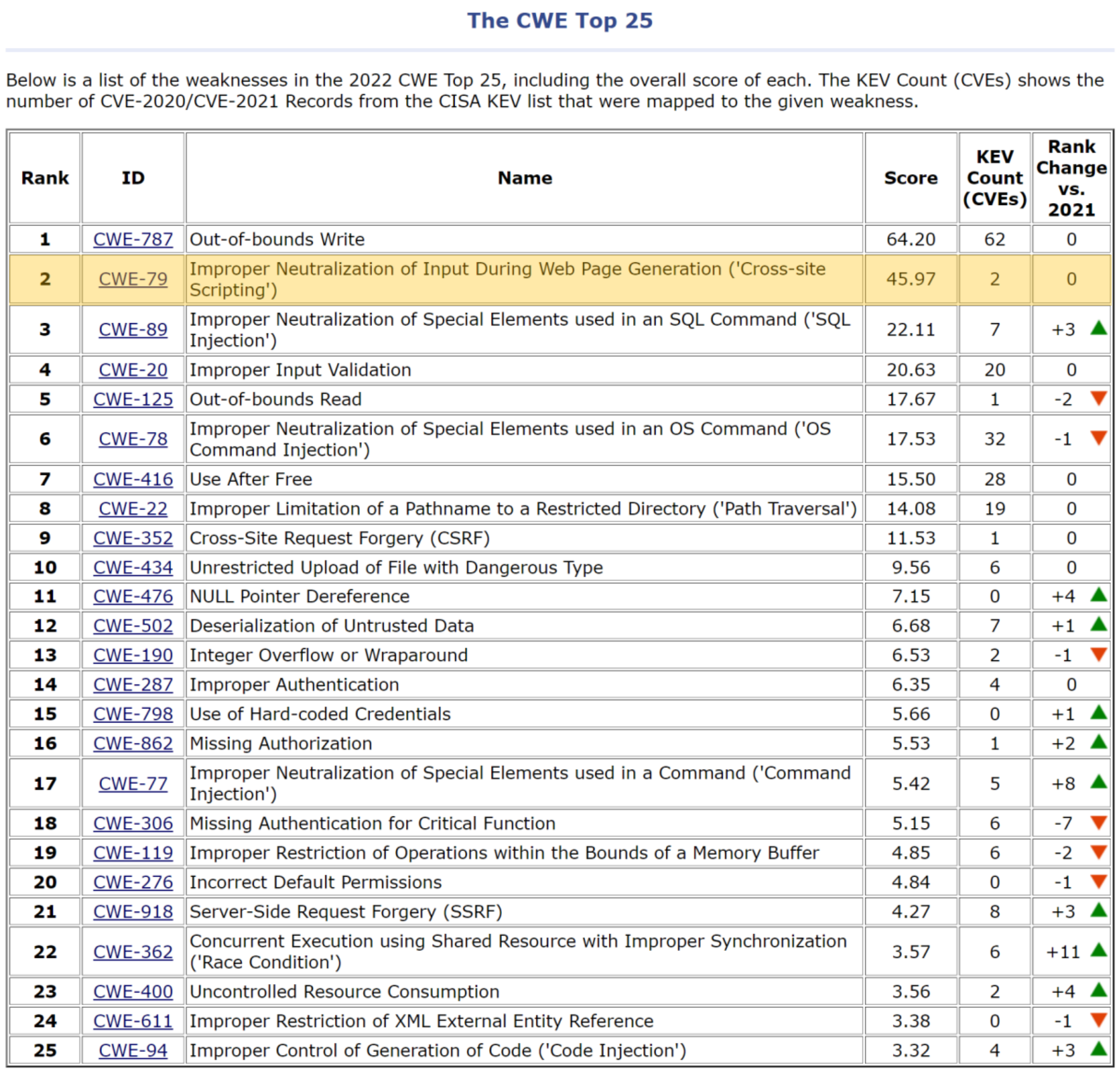
- JavaScript can be used to:
  - Create (parts of) HTML pages
  - Track events in the browser like mouse or keyboard usage
  - Communicate over http (in the background)
  - Store and read cookies (cookies become the only persistent storage)
- JavaScript has build-in mechanisms to improve security. Intuitively if we open a webpage JS we want:
  - No disk reading or writing
  - No access to other windows/tabs
  - No access to other domains -> Same Origin Policy

Same Origin Policy

- **Scripts** that are included into HTML page A can only access data in HTML page B if they have the same origin
- An origin is defined via the combination of **protocol, domain, and port**
- Some websites need additional sharing that is disallowed by the SOP
  - Consider a single-sign-on system where a central authentication service A realizes all logins of the user U to third parties T, e.g. an online-shop that the user logs in, say with its operating system account
  - Here the authenticator A has to deliver personal information of the user to the website P.

Cross-Site Scripting (XSS)

Significance

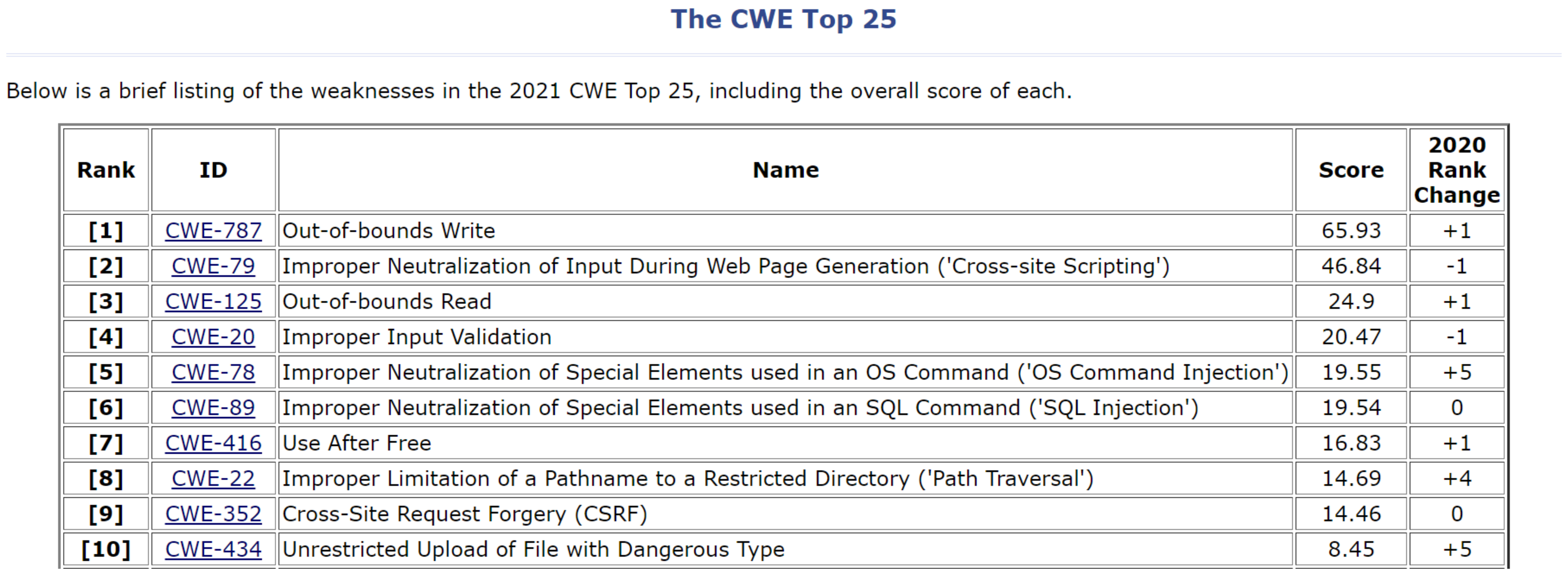


The CWE Top 25

Below is a list of the weaknesses in the 2022 CWE Top 25, including the overall score of each. The KEV Count (CVEs) shows the number of CVE-2020/CVE-2021 Records from the CISA KEV list that were mapped to the given weakness.

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

XSS is Widespread



The CWE Top 25

Below is a brief listing of the weaknesses in the 2021 CWE Top 25, including the overall score of each.

Rank	ID	Name	Score	2020 Rank Change
[1]	CWE-787	Out-of-bounds Write	65.93	+1
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.84	-1
[3]	CWE-125	Out-of-bounds Read	24.9	+1
[4]	CWE-20	Improper Input Validation	20.47	-1
[5]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	19.55	+5
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	19.54	0
[7]	CWE-416	Use After Free	16.83	+1
[8]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.69	+4
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	14.46	0
[10]	CWE-434	Unrestricted Upload of File with Dangerous Type	8.45	+5

https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html

XSS

- The idea of a cross-site scripting attack is to circumvent the Same Origin Policy
- To this end, the attacker crafts a script that would give the attacker access to critical information of the user with respect to some legitimate website (say an online bank service), by for example:
    - recording key strokes to obtain password information
    - reading cookie information to later be able to impersonate the user
- To make this attack work, the browser of the user must be tricked into believing that the attacker‘s script is from the legitimate site (the bank site)

XSS

- In general, XSS attacks often are used to escalate attacks from vulnerabilities of applications to security problems among several distributed parties
- At its core XSS attacks exploit trust relations among communication partners

Cross-Site Scripting Attack

- In a XSS attack, the attacker makes the legitimate website (e.g. a bank) send the attacker‘s script to the user
- The user‘s browser will not reject the script since it is indeed coming from the correct origin
- The attackers script may exploit this in one or several ways
  - The attacker’s script makes the browser sent the cookie of the user – received after successful authentication on the webserver – to the attacker. The attacker can now impersonate the user.
  - The script may direct the browser to make any other request to the website. For example, on a poorly secured webserver of an online bank, the script may initiate a transaction.

Persistent XSS

- The attacker‘s script is stored on the legitimate website and
- The website will later send it to the user without being aware of the attack
- The script is executed by the user and for example delivers keystrokes to the attacker site which in turn uses them to **impersonate the user**

- **As an example think of a website where people can upload digital files/data**
    - **If an attacker uploads a file that contains malicious script code to record keystrokes and if the service is such that the code is executed by the browser when accessing the file, then the attacker may obtain security-critical information of users with respect to their accounts on that site**
- **More concretely, think of the comment section of a news site**

Countermeasures (implementable by webserver)

- Check uploaded files/data and reject if they contain script code
- Use additional authentication request when changing important information
- Set cookie with HttpOnly

Reflected XSS

- The attacker makes the user send the malicious script to the server
- The server‘s response contains the script that it just received, essentially echoing the script back to the user
- The browser executes the script believing it was crafted by the server while in reality it was created by the attacker

- **As an example, think of a website with a search function which echoes back the original search term when the result is returned**

Countermeasures (implementable by webserver)

- Avoid echoing
- Input validation to make sure it does not contain scripts

Server-Side XSS

- Attacker manipulates server to include malicious scripts in server responses
- Malicious code is sent out to all clients that the server communicates with
- Servers enjoy higher trust in many contexts. For example, they may be accessible by parties that otherwise have limited access to the Internet.