**WEB security vulnerabilities (XSS, CSRF, session fixation, SQL injection, man-in-the-middle, buffer overflow)**

**XXS**

*"XSS" redirects here. For other uses, see XSS (disambiguation).*

*Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications. XSS enables attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy. Cross-site scripting carried out on websites accounted for roughly 84% of all security vulnerabilities documented by Symantec as of 2007.[1] Bug bounty company HackerOne in 2017 reported that XSS is still a major threat vector.[2] XSS effects vary in range from petty nuisance to significant security risk, depending on the sensitivity of the data handled by the vulnerable site and the nature of any security mitigation implemented by the site's owner.*

**CSRF**

*Cross-site request forgery, also known as one-click attack or session riding and abbreviated as CSRF (sometimes pronounced sea-surf[1]) or XSRF, is a type of malicious exploit of a website where unauthorized commands are transmitted from a user that the web application trusts.[2] There are many ways in which a malicious website can transmit such commands; specially-crafted image tags, hidden forms, and JavaScript XMLHttpRequests, for example, can all work without the user's interaction or even knowledge. Unlike cross-site scripting (XSS), which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser.*

**Session Flaxing**

*In computer network security, session fixation attacks attempt to exploit the vulnerability of a system that allows one person to fixate (find or set) another person's session identifier. Most session fixation attacks are web based, and most rely on session identifiers being accepted from URLs (query string) or POST data.*

***SQL Injection***

*SQL injection is a code injection technique, used to attack data-driven applications, in which nefarious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).[1] SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.*

*SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.*

*In a 2012 study, it was observed that the average web application received 4 attack campaigns per month, and retailers received twice as many attacks as other industries.[2]*

***Buffer Overflow***

*In information security and programming, a buffer overflow, or buffer overrun, is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations.*

*Buffers are areas of memory set aside to hold data, often while moving it from one section of a program to another, or between programs. Buffer overflows can often be triggered by malformed inputs; if one assumes all inputs will be smaller than a certain size and the buffer is created to be that size, then an anomalous transaction that produces more data could cause it to write past the end of the buffer. If this overwrites adjacent data or executable code, this may result in erratic program behavior, including memory access errors, incorrect results, and crashes.*

***What is JSONP, CORS? (A communication technique used in JavaScript programs running in web browsers to request data from a server in a different domain, something prohibited by typical web browsers because of the same-origin policy)***

*This is a pretty broad question, and could warrant a wiki unto itself. There is also quite a bit on google regarding the two, but I think I can hit a few key points.*

*If you need a read-only ajax interface to your servers and you need to support IE<=9, Opera<12, or Firefox<3.5 or various other older or obscure browsers, CORS is out, use JSONP. IE8 and IE9 sorta support CORS but have problems, see the link in the first comment below.*

*On the other hand, if your web API is read/write (e.g. full REST or just POST/GET) instead of just read (i.e. GET), JSONP is out. Use CORS. JSONP is inherently read-only.*

*If neither of these are a concern, I would just go with whatever is easiest or most familiar to you. If its a tossup, try CORS, since it is the more "modern" solution and JSONP is more of a hack, turning data into scripts to bypass cross-domain restrictions. CORS does however, typically require more server-side configuration.*

*If you're using jQuery, I'm not sure where you're coming up with the idea that CORS is "much more friendly to the client and easier to implement." See https://gist.github.com/3131951 . jQuery abstracts the details of JsonP, and CORS can actually be somewhat tricky to implment on your server-side depending on what technology you're using.*

*I recently developed a web app, using jquery and backbone.js, which reads from various cross-domain web services that we control, and ended up using Json-P instead of CORS because we need to support IE7 and it was a bit simpler on the server side (we run Django w/ DjangoRestFramework), and virtually the same with jquery on the client side.*

**HTTPS negotiation steps.**

*HTTPS is defined in RFC 2818. In short, the browser first establishes an SSL/TLS connection to the server and then sends HTTP requests/responses within this connection.*

*To establish the SSL/TLS channel, the client initiates a handshake, during which the server send its X.509 certificate. Besides the SSL/TLS handshake mechanisms, the browser verifies the certificate against a list of trust anchors it has (the trusted certificates) and against the name it's trying to access (the host name in the URL must match the certificate, as defined in RFC 2818 Section 3.1). Most of this is usually implemented within the SSL/TLS stacks, but some browsers can let you bypass this by adding exceptions (sometimes, permanent exceptions), so browsers also have a fallback mechanism in case the SSL/TLS stack fails to accept the certificate successfully.*

*The SSL/TLS protocol (and its handshake) are defined in the SSLv3, TLS 1.0, 1.1 and 1.2.*

*Using the server public key in the server certificate, the client and server is able to perform an authenticated key exchange, after which they have a pre master secret in common. The master secret and then the shared symmetric keys used for encrypting the application data are derived from this pre master secret.*

*The mechanisms used for this authenticated key exchange depend on the cipher suite. There are more details in the TLS specification itself.*

*If you want to learn by example, it's worth looking at the Wireshark sample data (as shown in this answer).*

***What is HTTP Strict Transport Security (HSTS)? (Prevents Man in the Middle attacks)***

*HTTP Strict Transport Security (HSTS) is an opt-in security enhancement that is specified by a web application through the use of a special response header. Once a supported browser receives this header that browser will prevent any communications from being sent over HTTP to the specified domain and will instead send all communications over HTTPS. It also prevents HTTPS click through prompts on browsers.*

*The specification has been released and published end of 2012 as RFC 6797 (HTTP Strict Transport Security (HSTS)) by the IETF. (Reference see in the links at the bottom.)*

[*https://www.owasp.org/index.php/HTTP\_Strict\_Transport\_Security\_Cheat\_Sheet*](https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet)

***Browser-server communication methods: WebSocket, EventSource, Comet(Polling, Long-Polling, Streaming)***

***Big Answer with picture***

[*https://stackoverflow.com/questions/11077857/what-are-long-polling-websockets-server-sent-events-sse-and-comet*](https://stackoverflow.com/questions/11077857/what-are-long-polling-websockets-server-sent-events-sse-and-comet)

***What is role-based access control and access control list?***

*In computer systems security, role-based access control (RBAC)[1][2] is an approach to restricting system access to authorized users. It is used by the majority of enterprises with more than 500 employees,[3] and can implement mandatory access control (MAC) or discretionary access control (DAC). RBAC is sometimes referred to as role-based security.*

*Role-based-access-control (RBAC) is a policy neutral access control mechanism defined around roles and privileges. The components of RBAC such as role-permissions, user-role and role-role relationships make it simple to perform user assignments. A study by NIST has demonstrated that RBAC addresses many needs of commercial and government organizations[citation needed]. RBAC can be used to facilitate administration of security in large organizations with hundreds of users and thousands of permissions. Although RBAC is different from MAC and DAC access control frameworks, it can enforce these policies without any complication.*

***Access Control list***

*An access control list (ACL), with respect to a computer file system, is a list of permissions attached to an object. An ACL specifies which users or system processes are granted access to objects, as well as what operations are allowed on given objects.[1] Each entry in a typical ACL specifies a subject and an operation. For instance, if a file object has an ACL that contains (Alice: read,write; Bob: read), this would give Alice permission to read and write the file and Bob to only read it.*

***What is session and persistent cookies, sessionStorage and localStorage?***

[*https://stackoverflow.com/questions/19867599/what-is-the-difference-between-localstorage-sessionstorage-session-and-cookies*](https://stackoverflow.com/questions/19867599/what-is-the-difference-between-localstorage-sessionstorage-session-and-cookies)

***How to implement remember-me? (***[***http://jaspan.com/improved\_persistent\_login\_cookie\_best\_practice***](http://jaspan.com/improved_persistent_login_cookie_best_practice)***)***

***Authentication using cookies, JWT (JSON Web Tokens).***

*https://ponyfoo.com/articles/json-web-tokens-vs-session-cookies*