Interview Notes

Explain the types of injection attacks?

What is XSS and how it's can be exploited??

What are Different Types of XSS?

Explain DOM Based XSS and how it can be prevented?

What are the Different Types of SQL injection?

What type of SQL injection vulnerability is on the login page if we bypassed?

What is Parameter deception?

What is web cache poising and web cache deception?

What is HTTP Parameter Pollution?

What is Prototype Pollution?

What is Second Order SQL Injection?

How to do OTP Bypass and Rate limit bypass?

How to get shell using SQLi?

What are the security headers used in an application?

What is httpsOnly and secure flag is used for?

What is CSRF and how it can be exploited?

What is sast and dast and why it is performed?

How cache control is implemented?

What are the HTTP cookie attributes name and explain?

List all Security Headers and explain them?

Difference between hashing and encryption?

What is CryptoGraphy?

What is Oauth and SAML and to exploit or attack them and how to protect them?

What is Source and Sink?

What is Inner HTML?

What is SSRF, how to exploit and prevent SSRF?

What is JWT and how it is exploited and how can you prevent it?

What is the structure of JWT?

What are JWKs and JKUs?

What is TLS handshake and SSL handshake?

What is XXE and how to exploit it and how to prevent it?

What is parameterized query in SQL?

Application and API security questions?

Which Vulnerability You mastered on?

Your Methodologies

OWASP Top 10 Vulnerabilities

---

2. What two criteria must be met to exploit Session Fixation?

Session Fixation is not the same thing as Session Hijacking, but rather a type of Session Hijacking attack. The two criteria are:

1. Attacker must be able to forcibly set a (syntactically valid but otherwise inactive) session token in the victim's browser (e.g. using XSS / CRLF injection).
2. Once the victim authenticates, the application uses the session token already present and does not set a new one.

3. What are the differences between Base64 and Base64URL encoding?

In Base64URL encoding, a "-" is used instead of a "+", and a "_" instead of a "/". Padding with = is also optional, and is usually omitted. This is to provide more compatibility if the value needs to be used in a URL.

Did you know that padding is actually not required at all for decoding, even in regular Base64? This is because we can figure out how many bytes are left to decode based on the number of remaining Base64 characters:

2 characters = 1 more byte
3 characters = 2 more bytes

4. Name 5 (or more) types of Cross-Site Scripting.

This was quite controversial and I was generally pretty lenient on what constituted a "type". The 5 I had in mind while writing the question were: Reflected, Stored, DOM-based, CSTI, and Server-Side.

Other types suggested by people were: Self, XST, Universal, Blind, Mutation

7. How does the TE.TE variant of HTTP Request Smuggling work?

The TE.TE variant has two or more servers which always use the Transfer-Encoding header over the Content-Length header if both are present, which usually makes Request Smuggling impossible. However, by manipulating the Transfer-Encoding header, it is possible to cause one of the servers to not recognize it. This server will use the Content-Length header instead, allowing the Request Smuggling attack to work.

There are countless ways to manipulate the Transfer-Encoding header. Common ones are including whitespace before the colon, capitalization, or modifying the value "chunked" in the header itself.

15. What does the “unsafe-inline” value allow for if used in a script-src directive of a CSP?

"unsafe-inline" will allow all inline scripts (e.g <script>...</script> and "onevent" attributes) to be executed, but will not allow the loading of scripts from other files, nor will it allow the use of eval() and other methods which allow the execution of JavaScript from strings.

18. What are XML parameter entities and what limitations do they have in XXE Injection?

XML parameter entities are referenced using a % instead of &, but can only be referenced within a DTD, not the main XML document. This limitation means that parameter entities are often only useful with out-of-band XXE techniques.

20. What conditions must be met to *prevent* a browser from sending a CORS Preflight request?

1. Only GET, HEAD, or POST methods are allowed.

2. Only the following headers can be manually set: Accept, Accept-Language, Content-Language, Content-Type, Range.
3. If Content-Type is set, it must use one of the following: application/x-www-form-urlencoded, multipart/form-data, text/plain.
4. If XMLHttpRequest was used, no event listener must be registered on the XMLHttpRequest.upload property.
5. No ReadableStream object was used.

25. What is type juggling, and why does the JSON format help exploit these vulnerabilities?

Type juggling is a feature of certain programming languages where variables will be converted to a different type (e.g. string, integer, boolean) in certain operations, rather than throwing an exception. For example, when concatenating a string with an integer, the integer will be converted to a string.

This can however lead to vulnerabilities when preserving the type is important. The JSON format helps exploit these vulnerabilities as it supports a wide range of data types natively (numbers, strings, booleans, arrays, objects, and nulls), whereas regular URL/Body parameters often only support strings and arrays.

26. Describe 3 techniques you might use to find sensitive data being exposed by an application.

There are of course far more than 3 techniques, and any of the following would count:

1. Source code analysis.
2. Directory busting.
3. Causing errors / exceptions / stack traces by fuzzing.
4. Access control exploitation.
5. Google dorking.
6. Git repo history analysis.
7. Exploiting SQL injections.

27. Describe the attributes of a request which make it effectively immune to CSRF (i.e. CSRF mitigation is not required).

Again there are a few possible answers here:

1. If authentication uses an Authorization header and a non-trivial token (i.e. not Basic Auth), such as a JWT, or any kind of custom header with an unpredictable value.
2. If the server doesn't support CORS or has a locked down policy, and a non-standard HTTP method is used (e.g. PUT, DELETE), or the request body uses JSON/XML and requires an appropriate

Content-Type.

3. If the request relies on a "secret" value which effectively becomes an anti-CSRF token. For example, login requests are immune to CSRF because if the attacker knows the victim's credentials, they don't even need to perform a CSRF attack*.

*There are some rare edge cases where performing a CSRF attack against a login, despite knowing the victim's credentials, would be useful.

29. Name 5 vulnerabilities which could potentially lead to OS command execution on a web app.

There are quite a few ways, though several are rare or require highly specific setups to work:

1. OS Command Injection
2. Insecure Deserialization
3. Server-Side Template Injection
4. File Upload Vulnerabilities
5. File Inclusion Vulnerabilities
6. Server-Side Prototype Pollution
7. Code Injection
8. SQL Injection
9. XXE

31. Describe how you would test for Vertical Access Control vulnerabilities on an application with 20 roles and 300+ different “functional” requests.

While a manual effort is possible, the best way to do this is via some form of guided automation. In Burp Suite, the Auth Analyzer extension can be used to track multiple sessions (one for each role) and replay each request with updated session tokens, comparing the response to the original.

For the brave, the AuthMatrix extension allows for more complex automation, and can handle logging users in, tracking anti-CSRF tokens, etc. Access rules can be configured per request/role pair, and the entire setup can be saved and replayed at a later date to validate fixes.

32. Under what circumstances is a tab’s Session Storage instance preserved?

A tab's Session Storage instance is preserved if the page is reloaded, or if the user browses to another origin in the tab and later returns. If the user closes the tab, the instance is still preserved, provided the browser has the ability to reopen tabs.

In some browsers, Session Storage for tabs is preserved if the browser instance crashes rather than exiting cleanly, allowing users to resume their browsing session.

33. Other than uploading XML via a form, how else might one find and exploit XXE?

Many file formats use XML as a base and may trigger XXE if parsed insecurely. Examples include SVG, Microsoft documents (e.g. docx, xlsx), and other markup languages like KML.

In addition, SOAP services use XML-formatted requests. In some cases, APIs which default to JSON-formatted inputs will also accept the same inputs as XML.

45. What are some questions you would ask a customer during a web app pentest scoping call?

Many questions would depend on a demo of the application, however here are a few general ones:

1. How much functionality does the app contain (e.g. no. of "pages")?
2. How complex is the functionality (e.g. any learning curves, lengthy processes, etc.)?
3. How many different roles are there / should be tested?
4. Which environment is being tested (e.g. dev, staging, prod)?
5. Do our accounts have access to test/dummy data?
6. Are there any access restrictions (e.g. VPN, IP block)?
7. Are there any custom protocols being used (e.g. proprietary encoding/encryption)?
8. Is there any rate limiting, WAF/IPS in place?
9. Are there any out of scope areas, or vulnerabilities which should not be tested (e.g. Denial of Service)?

---

34. Name some common password reset flow vulnerabilities.

1. Basing the password reset on a user identifier (e.g. username) rather than a secret token.
2. Using host header injection to modify password reset links in emails in order to steal the token.
3. Easily guessable password reset tokens (bonus if they don't expire quickly / once used).
4. Using security questions instead of a secret token to authenticate the user.
5. Username enumeration based on password reset success messages.

35. What is the difference between encoding, encryption, and hashing?

Encoding is the process of transferring data from one format to another while preserving the integrity of the data. If the encoding algorithm is known, anyone can decode the original data.

Encryption is the process of scrambling data so that it can only be read by someone with the correct decryption key. Even if the encoding algorithm is known, unauthorized users will not be able to decrypt the data.

Hashing is the process of converting data into a number (aka hash) of fixed size (e.g. 256 bits), such that the same data results in the same number. This can be used to verify a user knows the initial data without needing to know the data itself (e.g. a password for a login). The process is irreversible, and in good hashing algorithms, it should be difficult to find two sets of data which result in the same hash.

36. Name some ways an attacker might exploit an HTTP Request Smuggling vulnerability.

1. Forcing a victim to trigger an XSS payload, including "unexploitable" payloads such as those contained within a UserAgent header.
2. Using some form of "save" functionality in the application to capture a victim's request, extracting their session token and hijacking their account.
3. Bypassing front-end access controls by smuggling a request to a disallowed area onto one of our own requests.

37. What is Server-Side Request Forgery and how can it be detected & exploited?

Server-Side Request Forgery (SSRF) occurs when an attacker can cause a server at the back-end of the application to make a "request" to a target it would not normally request from.

It can be detected by looking for parameters which contain references to URLs, hostnames, or file paths, and attempting to manipulate these parameters to see if a request is made to a server we control, or to some backend service we can detect.

SSRF can often be exploited to retrieve files from within the environment, perform basic port scanning, leak information from request headers, execute code, and even deliver XSS payloads.

38. Name some ways TLS / SSL can be misconfigured.

1. Outdated Protocols (e.g. SSLv3, TLSv1.0)
2. Insecure Private Key Sizes
3. Incomplete Certificate Chains
4. Expired / Revoked Certificates
5. Insecure Cipher Suites
6. Lack of Forward Secrecy
7. Insecure Key Exchange Algorithms
8. Insecure Client-Initiated Renegotiation

39. Give some reasons why sending sensitive data in a URL query parameter is insecure.

1. URLs are generally logged, by both the server and potentially proxy services in-between the user and application.
2. URLs are also saved to browser history, which may be preserved on shared public computers.
3. The data may be visible in screenshots and screen shares.
4. Users may think it is safe to copy URLs and share them.
5. If 3rd party resources are loaded by the client-side application, the data may get sent as part of the Referer header to the 3rd party.

40. In what ways could an open redirect be exploited?

1. A victim could be redirected to a malicious copy of the site and not notice, since the original URL was for the legitimate site.
2. If chained with an SSRF, it could be used to bypass URL validation and reach otherwise prohibited targets.
3. If chained with a misconfigured OAuth setup, it could be used to steal access tokens.
4. If the redirect uses the Location response header, we may be able to perform CRLF injection.

41. Describe two output encoding techniques and the context in which they should be used to mitigate Cross-site Scripting.

1. Encoding for HTML contexts involves converting the following characters into HTML entities: ∓ < > " '
2. Encoding for HTML attribute contexts is the same, provided all attribute values are quoted correctly. If not, all non-alphanumeric characters should be converted to HTML entities.
3. Encoding for JavaScript contexts involves converting all non-alphanumeric characters into the Unicode encoding format (e.g. \u0022).

42. Describe three “403 Forbidden” bypass techniques.

1. Using different HTTP methods (e.g. POST instead of GET), or using "method override" headers / URL parameters (e.g. X-HTTP-Method) if a back-end server supports them.
2. Using "Client Origin" HTTP headers (e.g. X-Forwarded-For) to forge our source IP address, bypassing IP-based blocklists.
3. Manipulating the URL path using directory traversal, case modification, adding characters, or double-URL encoding.

43. Describe some potential CAPTCHA weaknesses.

1. Replay attacks - using a previously confirmed correct answer.

2. Improper input validation - removing or blanking CAPTCHA-related parameters.
3. Leaked answers - the correct answer appears somewhere in the source code (I once found a CAPTCHA which worked by using CSS to distort text).
4. Low entropy - if the set of possible answers is too small, a brute-force attack may work.
5. Machine learning susceptible - with enough training data, a computer can solve the CAPTCHA.

44. You find XSS in an application, however the customer informs you that users should be able to submit HTML code. What advice would you give them to remain secure?

The easiest solution is likely to use an HTML sanitizer like DOMPurify with an allowlist of "safe" elements and attributes.

Another option is to use a separate "sandbox" domain to host the HTML code, displaying it using an iframe. Any JavaScript code will run in the security context of the sandbox and will not be able to affect the main application.

As an additional measure, a well-configured Content Security Policy can be used to instruct the browser to only run trusted JavaScript code.

30. What is prototype pollution, and what exploits could it lead to with both client / server-side variants?

Prototype Pollution is a JavaScript / NodeJS vulnerability that allows attackers to add properties to global object prototypes, which are then passed down to actual objects used in the application.

In client-side JS this can lead to DOM XSS. With server-side JS (e.g. NodeJS), it can lead to access control bypasses as well as potential RCEs.

28. What are 3 negative outcomes (i.e. bad for the tester) that could arise if “OR <true>” (or similar) is relied on for SQL injection testing? 👀

I've ranted about this before.

1. OR <true> can return all rows of a table, which could cause server issues if the table is large.
2. OR <true> can lead to false positives when testing for login bypasses, if the login expects only one row be returned for a valid login attempt.
3. OR <true> injected into an UPDATE or DELETE statement can be disastrous.

21. Describe 3 ways an Insecure Deserialization vulnerability could be exploited.

1. Modifying the value of an object attribute.

2. Modifying the type of an object attribute.
3. Using a Magic Method to make calls to other functions/methods (potentially leading to RCE).

22. List the checks an application might perform to ensure files cannot contain malicious content, and can only be uploaded to specific directories.

1. Only allowing files with certain extensions and mime-types to be uploaded.
2. Performing file analysis (to confirm the file type) and AV scans.
3. Performing path canonicalization before checking the end location of the file matches an allowed directory.

23. How does Mass Assignment work and what are some potential outcomes of exploiting such a vulnerability?

Mass Assignment occurs when functionality allowing users to create or update "objects" does not restrict which attributes a user can specify. This is more common in modern MVC-type frameworks.

This can lead to attackers being able to "upgrade" their role (e.g. to admin), add money to an account balance, assign potentially negative resources to other users, or perform a log forging attack by modifying date values, as well as countless other attacks.

24. What is GraphQL batching and how can it be used to bypass rate limiting?

GraphQL batching allows a user to send multiple queries or mutations to a GraphQL endpoint in a single request, either using arrays or aliases. Each query / mutation is then executed and a collection of results is returned in the response.

This can bypass rate limiting since instead of sending 1000 requests to the endpoint (for example), one request can be sent containing 1000 queries / mutations.

19. What recommendations would you give a customer for fixing DOM based XSS?

If possible, avoid passing untrusted inputs to potentially dangerous JavaScript functions. Checks should be implemented to ensure that values only include expected characters (as opposed to trying to detect bad characters). Encoding inputs is also a possiblity.

8. What is DOM Clobbering and how can it be used to bypass (some) HTML sanitizers, resulting in XSS?

DOM Clobbering is a way to manipulate the DOM using only HTML elements (i.e. no JavaScript). By using the id or name attribute of some elements, it is possible to create global variables in the DOM. This can lead to XSS in some cases.

I created a dynamic cheatsheet where you can see how DOM Clobbering works: DOM Clobbering Cheatsheet (works best in Chrome)

9. Describe how HTTP Parameter Pollution could be used to bypass a Web Application Firewall.

Some servers will concatenate parameter values if two or more identical parameters exist in requests, though often with a separator (e.g. a comma). For certain payloads, WAF detection can sometimes be bypassed if the payload can be split across multiple parameters.

10. Describe IDOR and explain how mitigating it is different from other access control vulnerabilities.

Insecure Direct Object References occur when an application provides functionality to access a resource using some unique reference (e.g. an ID) but does not perform adequate access control checks to determine if the user should have access to the specific resource.

Generally, the user should be able to access the functionality, but not all resources via the functionality. Thus, mitigation involves an access check comparing the user to the specific resource being requested, as opposed to the functionality itself.

11. What are JWKs and JKUs and how does their usage differ in JWTs?

A JSON Web Key (JWK) is a JSON object representing a signing key in a JWT. A JSON Web Key Set URL (JKU) is a URL which points to the location of a set of JWKs. In a JWT, both JWKs and JKUs go in the header.

When using a JWK, the entire public key is embedded within the header, whereas a JKU can point to a set of multiple public keys. In both cases a key ID (kid) is used to select the key to be used.

12. In the context of web apps, what is Business Logic and how does testing for Business Logic vulnerabilities differ compared to (for example) XSS, SQLi, etc?

Business logic is code which mimics real-world business operations / decisions, rather than code which handles how a user interacts with the application. Testing for business logic vulnerabilities usually involves identifying and challenging assumptions the developer has made about how someone uses the application, rather than technical oversights involving how data is processed.

It is impossible to identify business logic flaws using current scanners, since they require an understanding of the purpose of the application and are highly contextual.

13. Describe 3 payloads you could use to identify a server-side template engine by causing an error message.

1. Invalid syntax: ${{<%[%'"}}%\.
2. Divide by zero: ${1/0}
3. Invalid variable names: ${tib3rius}

14. What is the purpose of the Sec-WebSocket-Key header?

The "key" has nothing to do with security / encryption. Since WebSockets are created using an initial HTTP request, the Sec-WebSocket-Key header is used by the client to make sure the server supports WebSockets. If the client doesn't receive a correctly hashed version of the key from the server, it doesn't continue with the WebSocket setup.

16. Give an example of stateless authentication, and describe an inherent weakness of this authentication mechanism.

Authentication using a JWT is an example of stateless authentication. An inherent weakness of stateless authentication is the inability to forcibly expire user sessions, since all session information is stored on the client-side.

17. Describe 3 ways to mitigate Cross-Site Request Forgery.

1. Setting the SameSite cookie attribute to Lax or Strict on session cookies can prevent this cookie being added to cross-site requests, making forged requests unauthenticated. There are some exceptions if Lax is used.
2. Requiring Anti-CSRF Tokens to be submitted with vulnerable requests will prevent CSRF provided the tokens are unique, unpredictable, and are not (only) submitted in cookies.
3. Another option is to check the Referer header of a request to ensure it matches a trusted origin.

5. How does Boolean *Error* Inferential (Blind) SQL Injection work?

This one confused a lot of people. Boolean Inferential (or Blind) and Error-based SQL Injection are two different things, but neither were what I asked about. I very specifically wanted the Error variant of Boolean Inferential injection.

This is a variant where injecting "AND 1=1" and "AND 1=2" (for example) will return the same response! The trick is to purposefully cause a database error when a condition we want to test is true, and hope that error propagates back to the response somehow (e.g. a 500 Internal Server error).

Many ways to do this, but most use a CASE expression and some divide by zero if the condition is true. For example: AND 1=(SELECT CASE WHEN (1=1) THEN 1/0 ELSE 1 END)

6. What is the Same-Origin Policy (SOP) and how does it work?

The Same-Origin Policy is a security mechanism browsers use to prevent a variety of cross-origin attacks. The basic principle is that client-side app code can only read data from a specific URL if the URL has the same origin as the current app. Two URLs have the same origin if they share the same protocol, host, and port.

Note that reading and embedding data from URLs are treated differently, allowing applications to embed things like scripts, videos, images, etc. without actually being able to access the raw bytes of each.

1. What is the difference between Web Cache Deception and Web Cache Poisoning?

Web Cache Deception involves finding some dynamic page which you can access via a URL a web cache will automatically cache (e.g. if /transactions can be accessed at /transactions.jpg). If an attacker can trick a victim into visiting the cacheable URL, they can then load the same URL and retrieve the victim's information from the cache.

Web Cache Poisoning involves finding an input which results in some exploitable change in the response, but doesn't form part of the cache key for the request. When an attacker sends their payload, the exploited response will be cached and then delivered to anyone who accesses the page.

2. What two criteria must be met to exploit Session Fixation?

Session Fixation is not the same thing as Session Hijacking, but rather a type of Session Hijacking attack. The two criteria are:

1. Attacker must be able to forcibly set a (syntactically valid but otherwise inactive) session token in the victim's browser (e.g. using XSS / CRLF injection).
2. Once the victim authenticates, the application uses the session token already present and does not set a new one.