Bypassing Client Side Controls

- Since the user controls the data that is being sent to to server through the client, the user has full control of bypassing client side control mechanisms.
  This is the core prroblem with web applications. → Submission of arbitrary input.

- Transmission of data through a Client
  → Often done by developers to ease the pressure on the server.
  → Removes the need to track all data of the user. → Reducing the per-session data
  → Allows rapid development
However transmitting sensitive information using this method is the reason vulns arise.

- Hidden Form Fields
  → Not visible on screen, but stored in the form and sent back to the server on submission.
  → Intercepting and editing these hidden values in forms can lead to a vuln. → IDOR

- HTTP Cookies
  → As with cookies, they are not visible elements.
  → If agreed parameters are transmitted through the Cookie header, then try changing them to test a vuln.
  → If the modified value reflects a change in the webapp, it can lead to a vuln

- URL Parameters
  → Application frequently use URL params to transfer data. Probing these params can lead to major vulns.
  → Search for
    ⇒ Embedded images using URL with params
    ⇒ URL with params used to load a frame's contents
    ⇒ Preset Params in a POST method form

- The Referer Header
  → Included to indicate from which domain the request has been generated.
  → Can be modified to forge requests and establish trust in a web application
  Note : This header is strictly optional according to the w3 standards

****

HACK STEPS :

1) Locate all hidden form fields, cookies and URL params that are possibly trasmitting data via the client.
2) Attempt to discover the role the item plays in the logic of the webapp.
3) Modify and observe if arbitrary inputs are rendered and whether this leads to a vuln

****

- Opaque Data
  → Sometimes data being transmitted will be cryptic or even obfuscated by the application.
  → To attack these surfaces, you must understand the method of obfuscation and wrap your payloads with these.

****
HACK STEPS :

1) If the plaintext behind opaque data is known, then try deciphering the alg used.
2) Try to leverage other functionalities that may return a ciphered result of any text submitted by you.
   Reuse output from these.
4) Reusuing other available ciphered results may result in a vuln.
5) Attack server side logic that decrypts the ciphered data by submitting malformed data and
   analyzing the response to these inputs. (Last Resort)

****

- Attacking ASP ViewState:
  → Instead of params, ASP.NET tranismits a ViewState hidden param that is obfuscated and protected with an optional hash (MAC).
  → If hash is not enabled, burp can decode the param.
  → Edit the values and see what happens to the web app.

- Capturing User Data : HTML Forms
  → Try Disabling normal HTML Form controls to bypass any validation checks.

- Script Based Validation
  → HTML validations are insufficient and quite siimple, many web apps may use custom script based validation.
  → To Bypass,
    ⇒ Bad method
      • Disable JS for the website to bypass the script based checks. But this may break the web app if it requires JS to function.
    ⇒ Good Method
      • Submit a proper value
      • Intercept request and change the value to payload intended
    ⇒ Alternate
      • Capture the initial reponse with form elements in it and remove the event handlers in the HTML.
      • Hence disabling the calls to the validation functions

- Disabled Elements
  → Greyed out elements that are not interactable with.
  → These are not sent to the form handler as well.
  → Try enabling these elements by modifying the form and submitting input. If the input is processed then they can be levergaed to break something.

- Skipped the Browser extension Section

- Handling Client-Side Data Securely
  → Sensitive data should be stored and retrieved on server side. (Ex: dataabses)
  → Or if there is no other alternative, then client data being sent has to be validated using either obfuscation or hashing techniques
    ⇒ Pitfallls to avoid are
      • Webapp maybe susceptible to **Replay Attack** i.e Reusing obfuscated data available in the web app to leverage something.
      • Cracking the encryption alg. Crypt- analysis can lead to deciphering of obfuscated data.
      • In ASP.NET use ViewState to store only state information. And always enable MAC protection to protect the data from being tampered.

- Validating Client Data
  → The only secure way to validate client-generated data is on the server side ofthe application. Every item of data received from the client should be regardedas tainted and potentially malicious.

- Logging and Alerting
  → The server-side logic thatperforms validation of client-submitted data should be aware of the validationthat has already occurred on the client side.
  → If data that would have been blockedby client-side validation is received, the application may infer that a user is actively circumventing this validation and therefore is likely to be malicious.