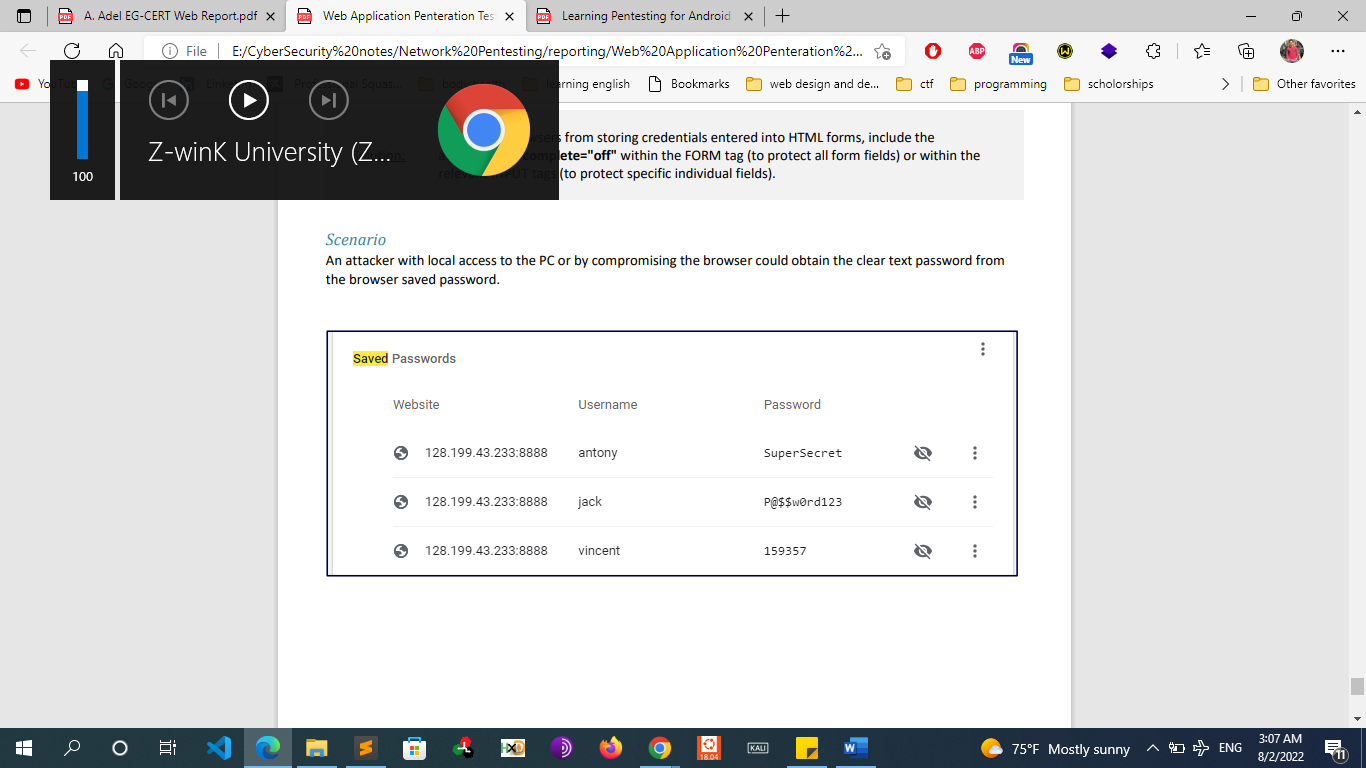
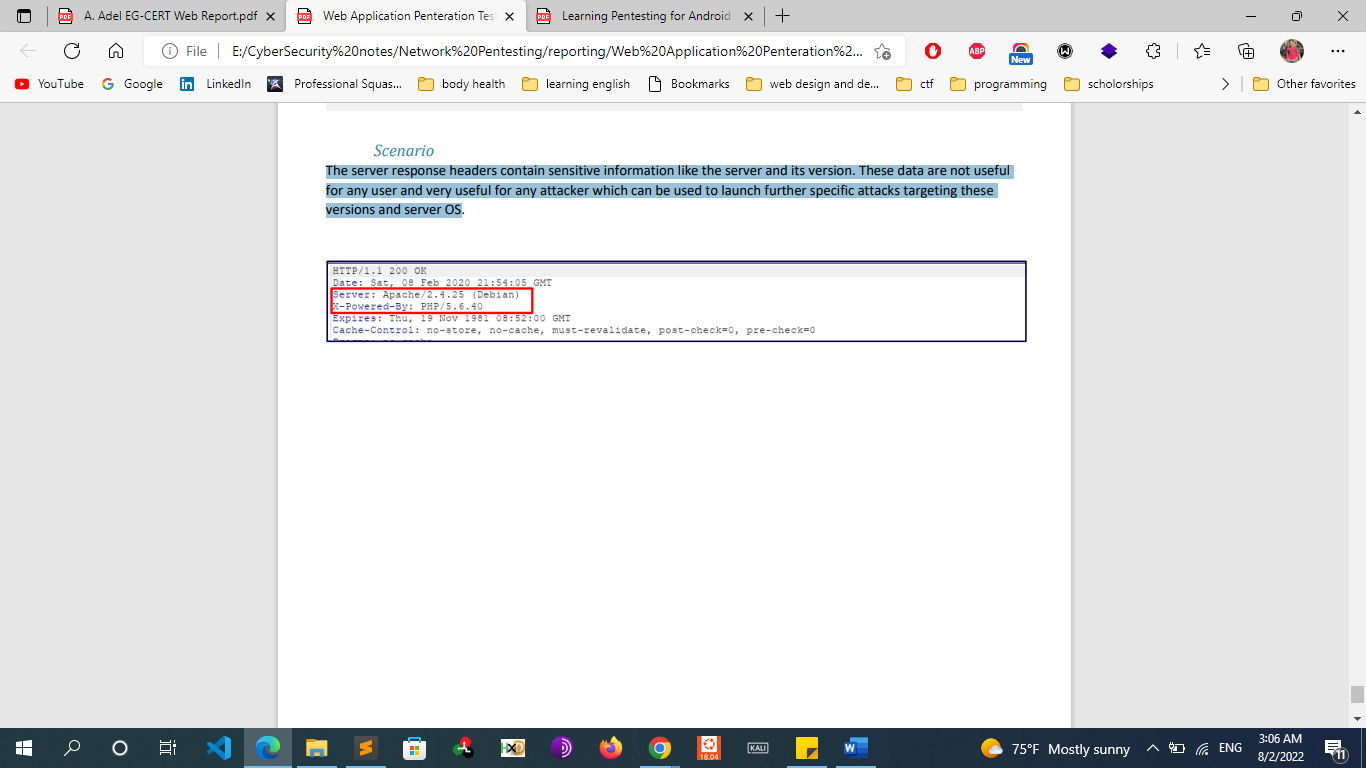
**Bug Bounty Report Snippets**

* **Low risk bugs**
  + **COOKIE WITHOUT HTTPONLY FLAG SET**
    - **Summary**
      * A cookie was found without the HTTPOnly flag set, making the cookies more vulnerable to XSS attacks.
      * If a browser that supports HttpOnly detects a cookie containing the HttpOnly flag, and client side script code attempts to read the cookie, the browser returns an empty string as the result. This causes the attack to fail by preventing the malicious (usually XSS) code from sending the data to an attacker's website.
      * Reference
        + https://owasp.org/www-community/HttpOnly
    - **Steps To Reproduce**
      * Go to the home page in the web application while intercepting the request with burp suite proxy
      * At the response or the request, you will notice that cookies don’t have Http flag set
    - **Remediation**
      * Include the HttpOnly flag by including it as an attribute
  + **COOKIE WITHOUT secure FLAG SET**
    - **Summary**
      * Cookies were issued by the application and do not have the secure flag set. The secure flag is an option that can be set by the application server when sending a new cookie to the user within an HTTP Response. The purpose of the secure flag is to prevent cookies from being observed by unauthorized parties due to the transmission of them in clear text.
      * To accomplish this goal, browsers which support the secure flag will only send cookies with the secure flag when the request is going to an HTTPS page. Said in another way, the browser will not send a cookie with the secure flag set over an unencrypted HTTP request. By setting the secure flag, the browser will prevent the transmission of a cookie over an unencrypted channel.
      * Reference:
        + https://owasp.org/www-community/controls/SecureCookieAttribute
    - **Steps To Reproduce**
      * Go to the home page in the web application while intercepting the request with burp suite proxy
      * At the response or the request, you will notice that cookies don’t have secure flag set
    - **Remediation**
      * It's recommended to set on the "Secure" flag for any authentication and session cookies. Using such an approach, the browser will not send a cookie with the secure flag set over an unencrypted HTTP request.
  + **click-jacking (Missing X-Frame-Options)**
    - **summary**
      * Clickjacking, also known as a “UI redress attack”, is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top level page. Thus, the attacker is “hijacking” clicks meant for their page and routing them to another page, most likely owned by another application, domain, or both.
      * Using a similar technique, keystrokes can also be hijacked. With a carefully crafted combination of stylesheets, iframes, and text boxes, a user can be led to believe they are typing in the password to their email or bank account, but are instead typing into an invisible frame controlled by the attacker.
      * If a page fails to set an appropriate X-Frame-Options or Content-Security-Policy HTTP header, it might be possible for a page controlled by an attacker to load it within an iframe. This may enable a clickjacking attack, in which the attacker's page overlays the target application's interface with a different interface provided by the attacker. By inducing victim users to perform actions such as mouse clicks and keystrokes, the attacker can cause them to unwittingly carry out actions within the application that is being targeted. This technique allows the attacker to circumvent defenses against cross-site request forgery, and may result in unauthorized actions.
      * Reference
        + <https://owasp.org/www-community/attacks/Clickjacking>
        + https://owasp.org/www-community/Clickjacking
    - **Steps To Reproduce**
      * Go to Any page at the web application
      * Check the response searching for the X-Frame-Options header
      * You will notice that every page in the application don’t have X-Frame-Options header, so the attacker could embedded these urls in Iframe tag .
        + <iframe src="http://128.199.43.233:8888/login.php" height="900" width="1900">
      * Show a Screenshot
    - **Remediation**
      * The X-Frame-Options HTTP response header can be used to indicate whether or not a browser should be allowed to render a page in a or . Sites can use this to avoid Clickjacking attacks, by ensuring that their content is not embedded into other sites. Set the X-Frame-Options header for all responses containing HTML content. The possible values are "DENY", "SAMEORIGIN", or "ALLOW-FROM uri"
      * Note that some applications attempt to prevent these attacks from within the HTML page itself, using "framebusting" code. However, this type of defense is normally ineffective and can usually be circumvented by a skilled attacker
      * Reference
        + https://owasp.org/www-project-cheatsheets/cheatsheets/Clickjacking\_Defense\_Cheat\_Sheet.htm
  + **WEEK password POLICY in Register page**
    - **Summary**
      * An important concern when using passwords for authentication is password strength. A strong password policy will make it difficult, if not improbable, for someone to guess a password through either manual or automated means.
      * The following characteristics define a strong password:
        + Password Length: Longer passwords include a greater combination of characters making it more difficult to guess. Passwords shorter than 10 characters are considered weak.
        + Password Complexity: Passwords containing a combination of upper-case and lower-case letters, numbers, and special characters are recommended.
        + Passphrases: Sentences or combinations of words can be much longer than typical passwords yet much easier to remember.
      * Reference
        + https://owasp.org/www-project-top-ten/2017/A2\_2017-Broken\_Authentication
        + <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/04-Authentication_Testing/07-Testing_for_Weak_Password_Policy>
    - **Steps To Reproduce**
      * Go to registration and input any dummy data and use one character as password
      * After clicking register you will notice that your account is created successfully with one character at password
    - **Remediation**
      * introduce additional authentication controls (i.e. two-factor authentication) or introduce a strong password policy. The simplest and cheapest of these is the introduction of a strong password policy that ensures password length, complexity, reuse and aging; although ideally both of them should be implemented.
  + **SESSION FIXATION**
    - **OWASP :** 
      * Broken Authentication
    - **Vulnerability description**
      * Session Fixation is a type of vulnerability, where the attacker can trick a victim into authenticating in the application using Session Identifier provided by the attacker. Unlike Session Hijacking, this does not rely on stealing Session ID of an already authenticated user. Instead, the attacker makes the victim use SID, which he already knows and which can he later use to make requests using victim's authenticated session.
      * Reference
        + <https://owasp.org/www-community/attacks/Session_fixation>
        + https://owasp.org/www-project-top-ten/2017/A2\_2017-Broken\_Authentication
    - **Steps to reproduce:**
      * Go to login page
      * Login while intercepting the request using burp suite
      * You notice that response header doesn’t set new session for the login and use the unauthenticated session instead
      * So The attacker can set the document.cookie to any unauthenticated user using xss vulnerability
      * When the victim login , the session will be assigned to this fake cookie , and the attacker can use it :
    - **Remediation**
      * Always change Session ID when a user authenticates.
      * When using cookies for SIDs, make sure they are HttpOnly and Secured.
      * Make sure sessions have timeouts set and they are not too long.
      * Note
        + Some platforms make it easy to protect against Session Fixation, while others make it a lot more difficult. In most cases, simply discarding any existing session is sufficient to force the framework to issue a new sessionid cookie, with a new value. Unfortunately, some platforms, notably Microsoft ASP, do not generate new values for sessionid cookies, but rather just associate the existing value with a new session. This guarantees that almost all ASP apps will be vulnerable to session fixation, unless they have taken specific measures to protect against it.
      * Reference
        + <https://owasp.org/www-community/controls/Session_Fixation_Protection>
  + **Password field with autocomplete enabled**
    - **Summary**
      * Most browsers have a facility to remember user credentials that are entered into HTML forms. This function can be configured by the user and also by applications that employ user credentials. If the function is enabled, then credentials entered by the user are stored on their local computer and retrieved by the browser on future visits to the same application.
      * The stored credentials can be captured by an attacker who gains control over the user's computer. Further, an attacker who finds a separate application vulnerability such as cross-site scripting may be able to exploit this to retrieve a user's browser-stored credentials.
      * Scenario
        + An attacker with local access to the PC or by compromising the browser could obtain the clear text password from the browser saved password.
        + 
      * Reference
        + https://portswigger.net/kb/issues/00500800\_password-field-with-autocomplete-enabled
    - **Remediation** 
      * To prevent browsers from storing credentials entered into HTML forms, include the attribute **autocomplete="off"** within the FORM tag (to protect all form fields) or within the relevant INPUT tags (to protect specific individual fields).
  + **Information Disclosure** 
    - **Overview (server response headers contain sensitive information)**
      * Information disclosure is when an application fails to properly protect sensitive information from parties that are not supposed to have access to such information in normal circumstances. These type of issues are not exploitable in most cases, but are considered as web application security issues because they allows attackers to gather information which can be used later in the attack lifecycle, in order to achieve more than they could if they didn’t get access to such information.
      * The server response headers contain sensitive information like the server and its version. These data are not useful for any user and very useful for any attacker which can be used to launch further specific attacks targeting these versions and server OS
        + Example :
        + 
    - **Solution:** 
      * Make sure that your web server does not send out response headers that reveal information about the backend technology type or version.
      * Make sure that all the services running on the server’s open ports do not reveal information about their builds and versions.
      * Always make sure that proper access controls and authorizations are in place in order to disallow access for attackers on all web servers, services and web applications
  + **MISSING RATE LIMIT in Login page that Allow for brute-forcing (A6- Security Misconfiguration)**
    - **Summary**
      * Brute-force attacks are often used for attacking authentication, A common threat web developers face is a password-guessing attack known as a brute force attack. A brute-force attack is an attempt to discover a password by systematically trying every possible combination of letters, numbers, and symbols until you discover the one correct combination that works. If your web site requires user authentication, you are a good target for a brute-force attack.
      * Reference
        + https://owasp.org/www-project-top-ten/2017/A2\_2017-Broken\_Authentication
    - **Steps To Reproduce**
      * Go to login input any invalid credentials
      * Click Login while intercepting the request with burp suite proxy
      * Move the request to intruder
      * In payload type choose null payloads and 1000 request
      * Click Start Attack
      * You will notice that their no lock or block after 1k login requests
    - **Remediation**
      * we recommend implementing a timeout after a number of requests in a period of time
      * Another approach is implementing CAPTCHA mechanism on the form pages
      * Another approach is a password lockout mechanism that temporarily locks an account if more than a preset number of unsuccessful login attempts are made.
        + This approach significantly slows down attackers, while allowing the accounts to be open for legitimate users.
      * Reference
        + https://owasp.org/www-community/controls/Blocking\_Brute\_Force\_Attacks
  + **User/email enumeration** 
    - **Summary**
      * User enumeration is when a malicious actor can use brute-force to either guess or confirm valid users in a system. User enumeration is often a web application vulnerability, though it can also be found in any system that requires user authentication. And here it found occurs are in a site's
        + login page.
        + Register Page
        + Password Reset Page
      * This vulnerability is discovered by submitting both known valid and invalid usernames/emails to the password reset functionality of the application, and comparing application responses.
      * If the invalid user is identified through the app error message, an attacker is able to enumerate a list of emails used by brute-forcing possible emails and noting the differences in responses.
      * Reference
        + https://owasp.org/www-project-web-security-testing-guide/latest/4-Web\_Application\_Security\_Testing/03-Identity\_Management\_Testing/04-Testing\_for\_Account\_Enumeration\_and\_Guessable\_User\_Account
    - **Remediation**
      * An effective remediation would be to have the server respond with a generic message that does not indicate which field is incorrect. When the response does not indicate whether the username or the password is incorrect, the malicious actor cannot infer whether usernames are valid.
  + **Insecure Protocol HTTP**
    - **Description:** 
      * Using HTTP within the web service communication is insecure as the HTTP sends the traffic in clear text format, with MITM attack performed we could see or alter the traffic sent.
      * Scenario
        + Using MITM and wireshark , the attacker can sniff the credentials
        + Screenshot POC using Wireshark
    - **Solution:** 
      * Use HTTPS which encrypts the sent traffic between the web services
  + **Missing HTTP Strict Transport Security (HSTS) Header**
    - **summary**
      * It has been discovered that the affected application is using HTTPS, however does not use the HSTS header. The HTTP protocol by itself is clear text, meaning that any data that is transmitted via HTTP can be captured and the contents viewed. To keep data private and prevent it from being intercepted, HTTP is often tunneled through either Secure Sockets Layer (SSL) or Transport Layer Security (TLS). When either of these encryption standards are used, it is referred to as HTTPS.
      * HTTP Strict Transport Security (HSTS) is an optional response header that can be configured on the server to instruct the browser to only communicate via HTTPS. This will be enforced by the browser even if the user requests a HTTP resource on the same server.
      * Cyber-criminals will often attempt to compromise sensitive information passed from the client to the server using HTTP. This can be conducted via various Man-in-The-Middle (MiTM) attacks or through network packet captures.
      * reference
        + https://www.owasp.org/index.php/HTTP\_Strict\_Transport\_Security\_Cheat\_Sheet http://cwe.mitre.org/data/definitions/200.html
    - **Remediation**
      * Consider implementing the HSTS header.
      * Depending on the framework being used the implementation methods will vary, however it is advised that the Strict-Transport-Security header be configured on the server.
      * One of the options for this header is max-age, which is a representation (in milliseconds) determining the time in which the client’s browser will adhere to the header policy. Depending on the environment and the application this time period could be from as low as minutes to as long as days.
  + **Cross-Domain policy Misconfigurations**
    - **Vulnerability Description**
      * Cross Origin Resource Sharing or CORS is a mechanism that enables a web browser to perform "cross-domain" requests using the XMLHttpRequest L2 API in a controlled manner. In the past, the XMLHttpRequest L1 API only allowed requests to be sent within the same origin as it was restricted by the same origin policy.
      * Cross-Origin requests have an Origin header, that identifies the domain initiating the request and is always sent to the server. CORS defines the protocol to use between a web browser and a server to determine whether a cross-origin request is allowed. In order to accomplish this goal, there are a few HTTP headers involved in this process, that are supported by all major browsers and we will cover below including: Origin, Access-Control- Request-Method, Access-Control-Request-Headers, Access-Control-Allow-Origin, Access-Control-Allow- Credentials, Access-Control-Allow-Methods, Access-Control-Allow-Headers.
      * The CORS specification mandates that for non-simple requests, such as requests other than GET or POST or requests that uses credentials, a pre-flight OPTIONS request must be sent in advance to check if the type of request will have a bad impact on the data. The pre-flight request checks the methods, headers allowed by the server, and if credentials are permitted, based on the result of the OPTIONS request, the browser decides whether the request is allowed or not.
      * Reference
        + https://owasp.org/www-project-web-security-testing-guide/latest/4-Web\_Application\_Security\_Testing/11-Client-side\_Testing/07-Testing\_Cross\_Origin\_Resource\_Sharing
    - **Remediation**
      * CORS should be using a stricter policy of allowed domains and methods and validate the origin.
  + **Missing SPF Record**
    - **Vulnerability name:**
      * Missing SPF record
    - **Vulnerability description:**
      * An SPF record is a type of Domain Name Service (DNS) record that identify which mail servers are permitted to send email on behalf of your domain. The purpose of an SPF record is to prevent spammers from sending messages with forged from addresses at your domain
    - **checking Missing SPE**
      * There Are Various Ways of Checking Missing SPF Records on a website but the Most Common and Popular way mxtools.com
    - **Steps to Check SPF Records on**
      * 1- Go to <http://w.kitterman.com/spf/validate.html> or mxtools.com
      * 2-Enter Target Website “**targetsite.com**”
        + If you seem any SPF Record than Domain is Not Vulnerable But if you see Nothing Here then "! You Found a Bug"
    - **Attack Scenario &impact:**
      * If there is No SPF Records. An Attacker can spoof Email Via any Fake mailer Like “Anonymousmail.me” and Send Email From "Support or admin etc.."
        + Example Email: **support@targetsite.com**
      * And with some Social engineering He can takeover a user account Because The Email was sent from the Authorized Domain. So users can get tricked Easily.
    - **Remediation** 
      * As a website owner you should prevent your domains being used in spam mail by adopting both of the following approaches:
        + Implement the Sender Policy Framework (SPF): publish a DNS record to explicitly state which servers are allowed to send email from your domain.

Add an SPF record for your domain name at your DNS provider.

Create a new TXT record.

Set the Host field to the name of your subdomain (for example, mail if your email address is contact@mail.example.com), or to @ if you do not use a subdomain.

Enter your SPF record for the TXT Value field (for example, v=SPF1 a mx include:secureserver.net ~all)

* + - * + Implement Domain Key Identified Mail (DKIM): use a digital signature to prove that outgoing email was legitimately sent from your domain, and that it wasn’t modified in transit.
        + There is also an emerging umbrella standard called DMARC (“Domain-based Message Authentication, Reporting & Conformance”) that you should be aware of.
  + **Check for secure Transmission**
    - **Check credentials only delivered over HTTPS**
    - **Check that the login/registration form is delivered over HTTPS**
    - **Check session tokens only delivered over HTTPS**
* **Medium Risk Bugs**
  + **Reflective XSS**
    - **Report Title:**
      * Reflective XSS
    - **Vulnerability Details**
      * **Vulnerability description:**
        + URL redirection is sometimes used as a part of phishing attacks that confuse visitors about which web site they are visiting
      * **Impact**
        + whenever a user visit this URL it will redirect them To site.com It is used in phishing attacks
      * **Risk Breakdown**
        + Risk: High
        + Difficulty to Exploit: Medium
        + CVSS2 Score: 7.9 (AV:N/AC:M/Au:S/C:C/I:C/A:N)
      * **Vulnerable Url:**
        + URL

<https://protect2.fireeye.com/url?k-171e4e52f8ee2b.8817lefe-1138524403bba8dfsushttps://www.site.com>

* + - * + Param
      * **Payload**
        + <script>alert(1)<script>
    - **How to reproduce this vulnerability**
      * Login to the web application
      * Go to the URL bellow
        + http://128.199.43.233:8888/modifyfile.php?content=/development/&newcontent=%3Cimg%20sr c=x%20onerror=alert(document.cookie)%3E
      * You will notice a popup with cookie
    - **POC**
    - **Remediation**
      * Your script should properly sanitize user input.
  + **Cross-Site Request Forgery**
    - **Report Title**
      * Cross-Site Request Forgery
    - **Vulnerability Details**
      * **Vulnerability Description**
        + Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they’re currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker’s choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.
        + Reference

https://owasp.org/www-community/attacks/csrf

* + - * **Impact**
        + whenever a user visit this URL it will redirect them To site.com It is used in phishing attacks
      * **Risk Breakdown**
        + Risk: High
        + Difficulty to Exploit: Medium
        + CVSS2 Score: 7.9 (AV:N/AC:M/Au:S/C:C/I:C/A:N)
      * **Vulnerable Url:**
        + https://protect2.fireeye.com/url?k-171e4e52f8ee2b.8817lefe-1138524403bba8dfsushttps://www.site.com
    - **How to reproduce this vulnerability**
      * 1-> open this URL "https://protect2.fireeye.com/url?k=88171e45218ee2b.88171efe-1138524403bba@dfsu-https://www.bing.com
      * 2-> it will directly redirect to bing.com
    - **POC**
      * Attach proof-of-concept scripts, screenshots, screen recordings, and so on
    - **Remediation**
      * Based on the risk of whether the form submission performs a sensitive action, the addition of anti-CSRF tokens may be required. These tokens can be configured in such a way that each session generates a new anti-CSRF token or such that each individual request requires a new token.
      * Reference
        + <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html>
  + **Unvalidated URL Redirection (open Redirect)**
    - **Report Title:**
      * Unvalidated URL redirection
    - **Vulnerability Details**
      * **Vulnerability description:**
        + URL redirection is sometimes used as a part of phishing attacks that confuse visitors about which web site they are visiting
        + Reference

https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated\_Redirects\_and\_Forwards\_Cheat\_Sheet.html

* + - * **Impact**
        + whenever a user visit this URL it will redirect them To site.com It is used in phishing attacks
      * **Risk Breakdown**
        + Risk: High
        + Difficulty to Exploit: Medium
        + CVSS2 Score: 7.9 (AV:N/AC:M/Au:S/C:C/I:C/A:N)
      * **Vulnerable Url:**
        + https://protect2.fireeye.com/url?k-171e4e52f8ee2b.8817lefe-1138524403bba8dfsushttps://www.site.com
    - **How to reproduce this vulnerability**
      * 1-> open this URL "https://protect2.fireeye.com/url?k=88171e45218ee2b.88171efe-1138524403bba@dfsu-https://www.bing.com
      * 2-> it will directly redirect to bing.com
    - **POC**
      * Attach proof-of-concept scripts, screenshots, screen recordings, and so on
    - **Remediation**
      * Your script should properly sanitize user input.
      * Also you can check
        + <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html>
  + **Sensitive Git Information Leakage** 
    - **Vulnerability Details**
      * **Vulnerability Description**
        + Git files, in relation to a website, can be publicly accessible if incorrectly configured. The “.git” directory was found on the web-server which allowed public access to the configuration file. Additionally, attackers can download the directory and use git commands to access logs, commit history, and the files in the repository.
      * **Impact**
        + whenever a user visit this URL it will redirect them To site.com It is used in phishing attacks
      * **Risk Breakdown**
        + Risk: High
        + Difficulty to Exploit: Medium
        + CVSS2 Score: 7.9 (AV:N/AC:M/Au:S/C:C/I:C/A:N)
      * **Vulnerable Url:**
        + http:// admin.contoso.com /assets/.git
    - **How to reproduce this vulnerability**
      * 1-> open this URL "https://protect2.fireeye.com/url?k=88171e45218ee2b.88171efe-1138524403bba@dfsu-https://www.bing.com
      * 2-> it will directly redirect to bing.com
    - **POC**
      * Attach proof-of-concept scripts, screenshots, screen recordings, and so on
    - **Remediation**
      * Remove the .git directory from being publicly accessible by adding a simple .htaccess file at the root of the affected web server(s). It should contain one line, “RedirectMatch 404 /\.git” and will also hide many additional Git files that may be web-accessible.
* **High/Critical Risk Bugs**
  + **SQL Injection**
    - **Vulnerability Details**
      * **Vulnerability Description**
        + SQL injection vulnerabilities arise when user-controllable data is incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query. This kind of attack can be detrimental to client and company data.
        + A wide range of damaging attacks can often be delivered via SQL injection, including reading, adding, modifying, or deleting critical application data, interfering with application logic, escalating privileges within the database and executing commands on the underlying operating system.
        + Reference

https://owasp.org/www-community/attacks/SQL\_Injection

* + - * **Impact**
        + whenever a user visit this URL it will redirect them To site.com It is used in phishing attacks
      * **Risk Breakdown**
        + Risk: High
        + Difficulty to Exploit: Medium
        + CVSS2 Score: 7.9 (AV:N/AC:M/Au:S/C:C/I:C/A:N)
      * **Vulnerable Url:**
        + https://protect2.fireeye.com/url?k-171e4e52f8ee2b.8817lefe-1138524403bba8dfsushttps://www.site.com
    - **Steps To reproduce** 
      * 1-> open this URL "https://protect2.fireeye.com/url?k=88171e45218ee2b.88171efe-1138524403bba@dfsu-https://www.bing.com
      * 2-> it will directly redirect to bing.com
    - **POC**
      * Attach proof-of-concept scripts, screenshots, screen recordings, and so on
    - **Remediation**
      * **parameterized queries** 
        + Use parameterized queries (also known as prepared statements) for all database queries. Parameterized queries force the developer to first define the SQL query structure, and then pass in each parameter to the query later. This coding style allows the database to distinguish between the actual query and the supplied data, regardless of what user input is supplied.
        + Prepared statements ensure that an attacker is not able to change the intent of a query, even if SQL commands are inserted by an attacker. A regular SQL statement could be interrupted by inputting a single quote, a comma, a comment character, or a semi-colon, depending on the structure. With prepared statements, these characters would always be interpreted as values, rather than interfering with the query itself.
      * **Stored Procedures** 
        + Language specific recommendations:

Java EE – use PreparedStatement() with bind variables.

.NET – use parameterized queries like SqlCommand() or OleDbCommand() with bind variables

PHP– use PDO with strongly typed parameterized queries (using bindParam())

SQLite - use sqlite3\_prepare() to create a statement object

* + - * + Stored procedures have the same effect as the use of prepared statements (when implemented safely) which is the norm for most stored procedure languages. They require the developer to build SQL statements with parameters which are automatically parameterized. The difference between prepared statements and stored procedures is that the SQL code for a stored procedure is defined and stored in the database itself, and then called from the application.
      * Both of these techniques have the same effectiveness in preventing SQL injection so your organization should choose which approach is most appropriate.
      * Reference:
        + https://owasp.org/www-project-cheatsheets/cheatsheets/SQL\_Injection\_Prevention\_Cheat\_Sheet.html
  + **Local File Disclosure**
    - **Vulnerability Details**
      * **Vulnerability Description**
        + Local File Disclosure (also known as file path traversal) is a web security vulnerability that allows an attacker to read arbitrary files on the server that is running an application. This might include application code and data, credentials for back-end systems, and sensitive operating system files. In some cases, an attacker might be able to write to arbitrary files on the server, allowing them to modify application data or behavior, and ultimately take full control of the server.
      * **Reference:** 
        + https://owasp.org/www-community/attacks/Path\_Traversa
      * **Impact**
        + whenever a user visit this URL it will redirect them To site.com It is used in phishing attacks
      * **Risk Breakdown**
        + Risk: High
        + Difficulty to Exploit: Medium
        + CVSS2 Score: 7.9 (AV:N/AC:M/Au:S/C:C/I:C/A:N)
      * **Vulnerable Url:**
        + http://128.199.43.233:8888/viewdocument.php?doc=/etc/hosts
    - **Steps To reproduce** 
      * Go to the link bellow.
        + http://128.199.43.233:8888/viewdocument.php?doc=/etc/hosts
      * you will notice content of hosts file of the backend server printed in the pd
    - **POC**
      * Attach proof-of-concept scripts, screenshots, screen recordings, and so on
    - **Remediation**
      * I highly recommend implementing one of the following filtering techniques:
        + Escaping path traversal characters.
      * Please refer to OWASP XSS prevention guide:
        + https://owasp.org/www-community/attacks/File\_System#Path\_traversal
  + **Stored XSS**
    - **Report Title:**
      * Stored XSS
    - **Vulnerability Details**
      * **Vulnerability description:**
        + Stored XSS occurs when a web application gathers input from a user which might be malicious, and then stores that input in a data store for later use. The input that is stored is not correctly filtered. As a consequence, the malicious data will appear to be part of the web site and run within the user’s browser under the privileges of the web application. Since this vulnerability typically involves at least two requests to the application, this may also call second-order XSS, and I have noticed that modify-file function is vulnerable to cross-site scripting.
      * **Reference**
      * **Impact**
        + this vulnerability can be used to conduct a number of browser-based attacks including:

Takeover the whole victim’s account

Hijacking another user's browser

Capturing sensitive information viewed by application users

Pseudo defacement of the application

Port scanning of internal hosts ("internal" in relation to the users of the web application)

Directed delivery of browser-based exploits

Other malicious activities

* + - * **Risk Breakdown**
        + Risk: High
        + Difficulty to Exploit: Medium
        + CVSS2 Score: 7.9 (AV:N/AC:M/Au:S/C:C/I:C/A:N)
      * **Vulnerable Url:**
        + URL

<https://protect2.fireeye.com/url?k-171e4e52f8ee2b.8817lefe-1138524403bba8dfsushttps://www.site.com>

* + - * + Param
      * **Payload**
        + <script>alert(1)<script>
    - **How to reproduce this vulnerability**
      * 1- open this URL "https://protect2.fireeye.com/url?k=88171e45218ee2b.88171efe-1138524403bba@dfsu-https://www.bing.com
      * 2- it will directly redirect to bing.com
    - **POC**
    - **Remediation**
      * I highly recommend implementing one of the following filtering techniques against the body of the emails:
        + Escaping.
        + Validating.
        + Sanitizing.
      * Please refer to OWASP XSS prevention guide:
        + https://github.com/OWASP/CheatSheetSeries/blob/master/cheatshe ets/Cross\_Site\_Scripting\_Prevention\_Cheat\_Sheet.md
  + **IDOR**
    - **Description:** 
      * Applications frequently use the actual name or key of an object when generating web pages. Applications don’t always verify the user is authorized for the target object. This results in an insecure direct object reference (IDOR) flaw. Testers can easily manipulate parameter values to detect such flaws and code analysis quickly shows whether authorization is properly verified.
      * Reference:
        + https://www.owasp.org/index.php/Testing\_for\_Insecure\_Direct\_Obj ect\_References\_(OTG-AUTHZ-004
    - **Steps to reproduce**
      * 1. Login to the web application
      * 2. Go to <http://128.199.43.233:8888/profile.php?id=1>
      * 3. Change the id parameter value to disclose other users’ information’
    - **Remediation:** 
      * Preventing insecure direct object references requires selecting an approach for protecting each user accessible object (e.g., object number, filename):
      * 1. Use per user or session indirect object references. This prevents attackers from directly targeting unauthorized resources. For example, instead of using the resource’s database key, a drop down list of six resources authorized for the current user could use the numbers 1 to 6 to indicate which value the user selected. The application has to map the per-user indirect reference back to the actual database key on the server. OWASP’s ESAPI includes both sequential and random access reference maps that developers can use to eliminate direct object references.
      * 2. Check access. Each use of a direct object reference from an untrusted source must include an access control check to ensure the user is authorized for the requested object.
  + **A5: Broken Access Control**
    - **Vulnerability Details**
      * **Vulnerability Description**
        + Broken access controls are a commonly encountered and often critical security vulnerability. Design and management of access controls is a complex and dynamic problem that applies business, organizational, and legal constraints to a technical implementation. Access control design decisions have to be made by humans, not technology, and the potential for errors is high.
      * **Impact**
        + whenever a user visit this URL it will redirect them To site.com It is used in phishing attacks
      * **Risk Breakdown**
        + Risk: High
        + Difficulty to Exploit: Medium
        + CVSS2 Score: 7.9 (AV:N/AC:M/Au:S/C:C/I:C/A:N)
      * **Vulnerable Url:**
        + https://protect2.fireeye.com/url?k-171e4e52f8ee2b.8817lefe-1138524403bba8dfsushttps://www.site.com
    - **How to reproduce this vulnerability**
      * 1-> open this URL "https://protect2.fireeye.com/url?k=88171e45218ee2b.88171efe-1138524403bba@dfsu-https://www.bing.com
      * 2-> it will directly redirect to bing.com
    - **POC**
      * Attach proof-of-concept scripts, screenshots, screen recordings, and so on
    - **Remediation**
      * use of an access control matrix to define the access control rules. Without documenting the security policy, there is no definition of what it means to be secure for that site. The policy should document what types of users can access the system, and what functions and content each of these types of users should be allowed to access.
  + **Unrestricted File Upload**
    - **Vulnerability Details**
      * **Vulnerability Description**
        + Uploaded files represent a significant risk to applications. The first step in many attacks is to get some code to the system to be attacked. Then the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step. The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement. It depends on what the application does with the uploaded file and especially where it is stored.
        + Reference

https://owasp.org/www-community/vulnerabilities/Unrestricted\_File\_Upload

* + - **Remediation**
      * Restrict file types accepted for upload:
      * check the file extension and only allow certain files to be uploaded.
      * Use a whitelist approach instead of a blacklist.
      * Check for double extensions such as .php.png.
      * Check for files without a filename like .htaccess (on ASP.NET, check for configuration files like web.config).
      * Change the permissions on the upload folder so the files within it are not executable.
      * If possible, rename the files that are uploaded.priate.