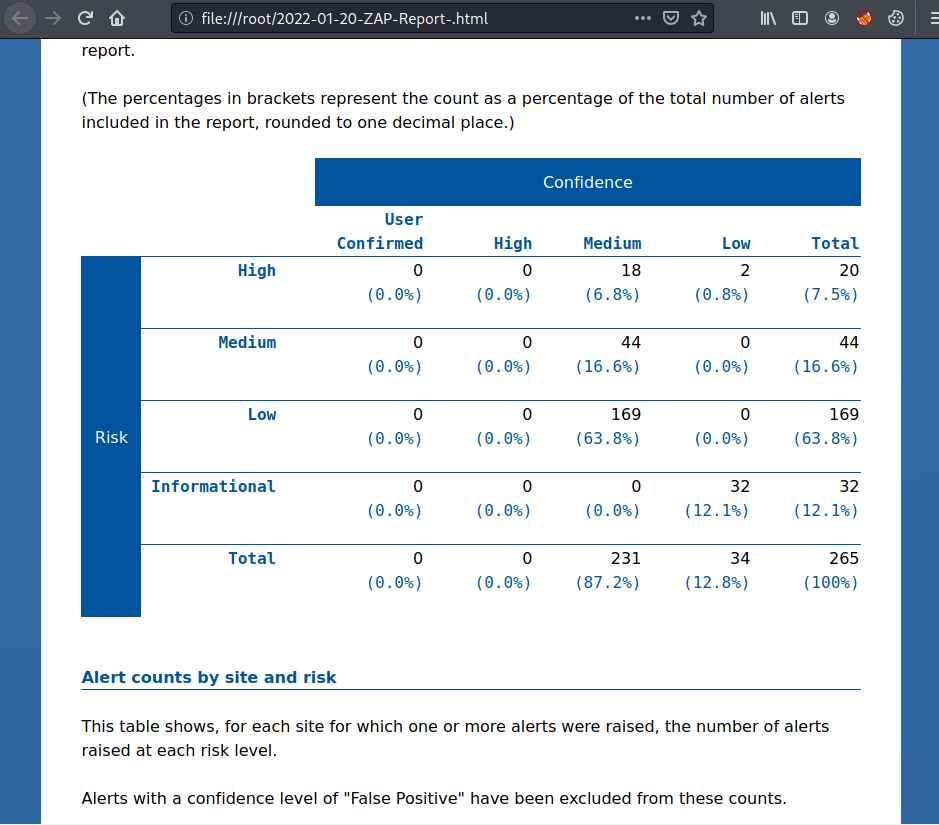
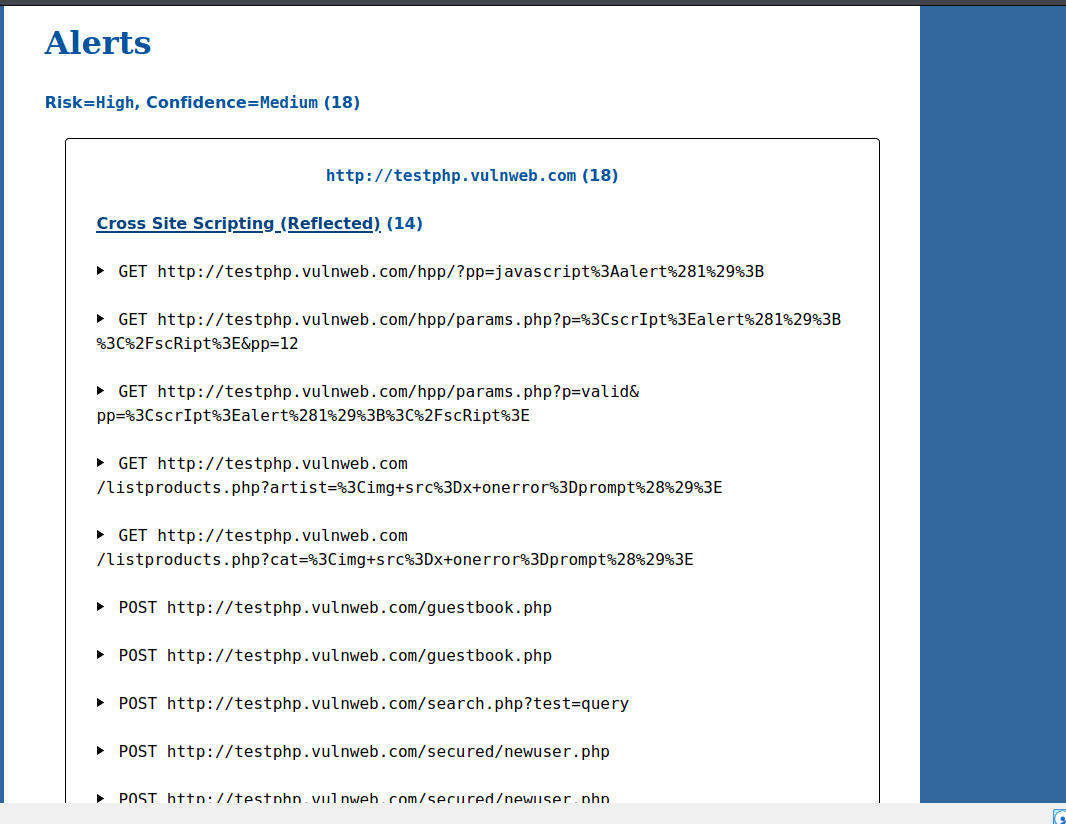
**Project 2:  
Task 1:**  Take some website (vulnerable website) . Scan using OWASP ZAP Tool (quick/automated ) .  
Set of vulnerabilities -◊ make a report with mitigation's

SOLUTION:

[**REPORT LINK**](https://drive.google.com/file/d/1-aKi6LBqpWqVDORC4pEWy1ZM_WKPy1r5/view?usp=sharing)

**SCREENSHOT :**





**ZAP Scanning Report**

**Summary of Alerts**

|  |  |
| --- | --- |
| **Risk Level** | **Number of Alerts** |
| [High](https://theses.cz/id/tj2mdi/JENKINS_ZAP_VULNERABILITY_REPORT.html#high) | 5 |
| [Medium](https://theses.cz/id/tj2mdi/JENKINS_ZAP_VULNERABILITY_REPORT.html#medium) | 4 |
| [Low](https://theses.cz/id/tj2mdi/JENKINS_ZAP_VULNERABILITY_REPORT.html#low) | 5 |
| [Informational](https://theses.cz/id/tj2mdi/JENKINS_ZAP_VULNERABILITY_REPORT.html#info) | 0 |

**Alert Detail**

|  |  |
| --- | --- |
| **High (Medium)** | **Path Traversal** |
| Description | The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal.  Most web sites restrict user access to a specific portion of the file-system, typically called the "web document root" or "CGI root" directory. These directories contain the files intended for user access and the executable necessary to drive web application functionality. To access files or execute commands anywhere on the file-system, Path Traversal attacks will utilize the ability of special-characters sequences.  The most basic Path Traversal attack uses the "../" special-character sequence to alter the resource location requested in the URL. Although most popular web servers will prevent this technique from escaping the web document root, alternate encodings of the "../" sequence may help bypass the security filters. These method variations include valid and invalid Unicode-encoding ("..%u2216" or "..%c0%af") of the forward slash character, backslash characters ("..\") on Windows-based servers, URL encoded characters "%2e%2e%2f"), and double URL encoding ("..%255c") of the backslash character.  Even if the web server properly restricts Path Traversal attempts in the URL path, a web application itself may still be vulnerable due to improper handling of user-supplied input. This is a common problem of web applications that use template mechanisms or load static text from files. In variations of the attack, the original URL parameter value is substituted with the file name of one of the web application's dynamic scripts. Consequently, the results can reveal source code because the file is interpreted as text instead of an executable script. These techniques often employ additional special characters such as the dot (".") to reveal the listing of the current working directory, or "%00" NULL characters in order to bypass rudimentary file extension checks. |
|  | |
| Solution | Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.  When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."  For filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses, and exclude directory separators such as "/". Use a whitelist of allowable file extensions.  Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised.  Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.  Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes ".." sequences and symbolic links.  Run your code using the lowest privileges that are required to accomplish the necessary tasks. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.  When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.  Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by your software.  OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows you to specify restrictions on file operations.  This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise. |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **High (Medium)** | **Remote File Inclusion** |
| Description | Remote File Include (RFI) is an attack technique used to exploit "dynamic file include" mechanisms in web applications. When web applications take user input (URL, parameter value, etc.) and pass them into file include commands, the web application might be tricked into including remote files with malicious code.  Almost all web application frameworks support file inclusion. File inclusion is mainly used for packaging common code into separate files that are later referenced by main application modules. When a web application references an include file, the code in this file may be executed implicitly or explicitly by calling specific procedures. If the choice of module to load is based on elements from the HTTP request, the web application might be vulnerable to RFI.  An attacker can use RFI for:  \* Running malicious code on the server: any code in the included malicious files will be run by the server. If the file include is not executed using some wrapper, code in include files is executed in the context of the server user. This could lead to a complete system compromise.  \* Running malicious code on clients: the attacker's malicious code can manipulate the content of the response sent to the client. The attacker can embed malicious code in the response that will be run by the client (for example, Javascript to steal the client session cookies).  PHP is particularly vulnerable to RFI attacks due to the extensive use of "file includes" in PHP programming and due to default server configurations that increase susceptibility to an RFI attack. |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Solution | Phase: Architecture and Design  When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.  For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap provide this capability.  Phases: Architecture and Design; Operation  Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by your software.  OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows you to specify restrictions on file operations.  This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.  Be careful to avoid CWE-243 and other weaknesses related to jails.  For PHP, the interpreter offers restrictions such as open basedir or safe mode which can make it more difficult for an attacker to escape out of the application. Also consider Suhosin, a hardened PHP extension, which includes various options that disable some of the more dangerous PHP features.  Phase: Implementation  Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.  When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."  For filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.  Phases: Architecture and Design; Operation  Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately.  This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce your attack surface.  Phases: Architecture and Design; Implementation  Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.  Many file inclusion problems occur because the programmer assumed that certain inputs could not be modified, especially for cookies and URL components. |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **High (Medium)** | **Cross Site Scripting (Reflected)** |
| Description | Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.  When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.  There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.  Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.  Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code. |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **High (Medium)** | **SQL Injection - MySQL** |
| Description | SQL injection may be possible. |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Solution | Do not trust client side input, even if there is client side validation in place.  In general, type check all data on the server side.  If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'  If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries.  If database Stored Procedures can be used, use them.  Do \*not\* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality!  Do not create dynamic SQL queries using simple string concatenation.  Escape all data received from the client.  Apply a 'whitelist' of allowed characters, or a 'blacklist' of disallowed characters in user input.  Apply the principle of least privilege by using the least privileged database user possible.  In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact.  Grant the minimum database access that is necessary for the application. |
| Other information | RDBMS [MySQL] likely, given error message regular expression [\Qcom.mysql.jdbc.exceptions\E] matched by the HTML results.  The vulnerability was detected by manipulating the parameter to cause a database error message to be returned and recognised |
|  | |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **High (Medium)** | **External Redirect** |
| Description | URL redirectors represent common functionality employed by web sites to forward an incoming request to an alternate resource. This can be done for a variety of reasons and is often done to allow resources to be moved within the directory structure and to avoid breaking functionality for users that request the resource at its previous location. URL redirectors may also be used to implement load balancing, leveraging abbreviated URLs or recording outgoing links. It is this last implementation which is often used in phishing attacks as described in the example below. URL redirectors do not necessarily represent a direct security vulnerability but can be abused by attackers trying to social engineer victims into believing that they are navigating to a site other than the true destination. |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Solution | Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.  When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."  Use a whitelist of approved URLs or domains to be used for redirection.  Use an intermediate disclaimer page that provides the user with a clear warning that they are leaving your site. Implement a long timeout before the redirect occurs, or force the user to click on the link. Be careful to avoid XSS problems when generating the disclaimer page.  When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.  For example, ID 1 could map to "/login.asp" and ID 2 could map to "http://www.example.com/". Features such as the ESAPI AccessReferenceMap provide this capability.  Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.  Many open redirect problems occur because the programmer assumed that certain inputs could not be modified, such as cookies and hidden form fields. |
| Other information | The response contains a redirect in its Location header which allows an external Url to be set. |
|  | |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Medium (Medium)** | **X-Frame-Options Header Not Set** |
| Description | X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks. |
|  | |
|  |  |
|  |  |
| Solution | Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers). |
|  |  |
|  | |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Medium (Medium)** | **X-Frame-Options Header Not Set** |
| Description | X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks. |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Solution | Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers). |
|  |  |
|  | |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Medium (Medium)** | **Format String Error** |
| Description | A Format String error occurs when the submitted data of an input string is evaluated as a command by the application. |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Solution | Rewrite the background program using proper deletion of bad character strings. This will require a recompile of the background executable. |
| Other information | Potential Format String Error. The script closed the connection on a /%s |
|  | |
| Reference | https://www.owasp.org/index.php/Format\_string\_attack |
| CWE Id | 134 |
| WASC Id | 6 |

|  |  |
| --- | --- |
| **Medium (Medium)** | **CRLF Injection** |
| Description | Cookie can be set via CRLF injection. It may also be possible to set arbitrary HTTP response headers. In addition, by carefully crafting the injected response using cross-site script, cache poisoning vulnerability may also exist. |
|  | |
| URL | http://localhost:8180/JavaVulnerableLab/Open?url=any%0D%0ASet-cookie%3A+Tamper%3D8f123149-ec50-4e07-95b6-322f64658c82 |
| Parameter | url |
| Attack | Set-cookie: Tamper=8f123149-ec50-4e07-95b6-322f64658c82 |
| Instances | 1 |
| Solution | Type check the submitted parameter carefully. Do not allow CRLF to be injected by filtering CRLF. |
| Other information | any  Set-cookie: Tamper=8f123149-ec50-4e07-95b6-322f64658c82 |
|  | |
| Reference | http://www.watchfire.com/resources/HTTPResponseSplitting.pdf  http://webappfirewall.com/lib/crlf-injection.txtnull  http://www.securityfocus.com/bid/9804 |
| CWE Id | 113 |
| WASC Id | 25 |

|  |  |
| --- | --- |
| **Low (Medium)** | **Web Browser XSS Protection Not Enabled** |
| Description | Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server |
|  | |
| URL | http://detectportal.firefox.com/success.txt |
| Instances | 1 |
| Solution | Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'. |
| Other information | The X-XSS-Protection HTTP response header allows the web server to enable or disable the web browser's XSS protection mechanism. The following values would attempt to enable it:  X-XSS-Protection: 1; mode=block  X-XSS-Protection: 1; report=http://www.example.com/xss  The following values would disable it:  X-XSS-Protection: 0  The X-XSS-Protection HTTP response header is currently supported on Internet Explorer, Chrome and Safari (WebKit).  Note that this alert is only raised if the response body could potentially contain an XSS payload (with a text-based content type, with a non-zero length). |
|  | |
| Reference | https://www.owasp.org/index.php/XSS\_(Cross\_Site\_Scripting)\_Prevention\_Cheat\_Sheet  https://blog.veracode.com/2014/03/guidelines-for-setting-security-headers/ |
| CWE Id | 933 |
| WASC Id | 14 |

|  |  |
| --- | --- |
| **Low (Medium)** | **X-Content-Type-Options Header Missing** |
| Description | The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing. |
|  | |
| URL | http://detectportal.firefox.com/success.txt |
| Instances | 1 |
| Solution | Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.  If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing. |
| Other information | This issue still applies to error type pages (401, 403, 500, etc) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type.  At "High" threshold this scanner will not alert on client or server error responses. |
|  | |
| Reference | http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx  https://www.owasp.org/index.php/List\_of\_useful\_HTTP\_headers |
| CWE Id | 16 |
| WASC Id | 15 |

|  |  |
| --- | --- |
| **Low (Medium)** | **Web Browser XSS Protection Not Enabled** |
| Description | Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Solution | Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'. |
|  |  |
|  | |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Low (Medium)** | **X-Content-Type-Options Header Missing** |
| Description | The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing. |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Solution | Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.  If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing. |
|  |  |
|  | |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Low (Medium)** | **Cookie No HttpOnly Flag** |
| Description | A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible. |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Solution | Ensure that the HttpOnly flag is set for all cookies. |
|  |  |
|  |  |
|  |  |