|  |  |
| --- | --- |
| **High (Medium)** | **SQL Injection - PostgreSQL - Time Based** |
| Description | SQL injection may be possible |
| URL | http://www.cypd.gov.tw/ |
| Method | GET |
| Parameter | query |
| Attack | field: [query], value [case when cast(pg\_sleep(15) as varchar) > '' then 0 else 1 end] |
| Instances | 1 |
| 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 privilege 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 | The query time is controllable using parameter value [case when cast(pg\_sleep(15) as varchar) > '' then 0 else 1 end], which caused the request to take [20,022] milliseconds, when the original unmodified query with value [query] took [614] milliseconds |
| Reference | https://www.owasp.org/index.php/Top\_10\_2010-A1  https://www.owasp.org/index.php/SQL\_Injection\_Prevention\_Cheat\_Sheet |
| CWE Id | 89 |
| WASC Id | 19 |
| Source ID | 1 |

|  |  |
| --- | --- |
| **High (Medium)** | **Anti CSRF Tokens Scanner** |
| Description | A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.  CSRF attacks are effective in a number of situations, including:  \* The victim has an active session on the target site.  \* The victim is authenticated via HTTP auth on the target site.  \* The victim is on the same local network as the target site.  CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy. |
|  | |
| URL | http://www.cypd.gov.tw/ |
| Method | GET |
| Evidence | <form id="searchform1" action="/home/search" class="inside-search"> |
| URL | http://www.cypd.gov.tw/ |
| Method | GET |
| Evidence | <form action="/home/search" class="inside-search"> |
| Instances | 2 |
| Solution | Phase: Architecture and Design  Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.  For example, use anti-CSRF packages such as the OWASP CSRFGuard.  Phase: Implementation  Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.  Phase: Architecture and Design  Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).  Note that this can be bypassed using XSS.  Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.  Note that this can be bypassed using XSS.  Use the ESAPI Session Management control.  This control includes a component for CSRF.  Do not use the GET method for any request that triggers a state change.  Phase: Implementation  Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons. |
| Reference | http://projects.webappsec.org/Cross-Site-Request-Forgery  http://cwe.mitre.org/data/definitions/352.html |
| CWE Id | 352 |
| WASC Id | 9 |
| Source ID | 1 |

|  |  |
| --- | --- |
| **High (Medium)** | **SQL Injection - Hypersonic SQL - Time Based** |
| Description | SQL injection may be possible |
|  | |
| URL | http://www.cypd.gov.tw/ |
| Method | GET |
| Parameter | query |
| Attack | field: [query], value ['; select "java.lang.Thread.sleep"(15000) from INFORMATION\_SCHEMA.SYSTEM\_COLUMNS where TABLE\_NAME = 'SYSTEM\_COLUMNS' and COLUMN\_NAME = 'TABLE\_NAME' -- ] |
| Instances | 1 |
| 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 privilege 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 | The query time is controllable using parameter value ['; select "java.lang.Thread.sleep"(15000) from INFORMATION\_SCHEMA.SYSTEM\_COLUMNS where TABLE\_NAME = 'SYSTEM\_COLUMNS' and COLUMN\_NAME = 'TABLE\_NAME' -- ], which caused the request to take [20,024] milliseconds, when the original unmodified query with value [query] took [474] milliseconds |
| Reference | https://www.owasp.org/index.php/Top\_10\_2010-A1  https://www.owasp.org/index.php/SQL\_Injection\_Prevention\_Cheat\_Sheet |
| CWE Id | 89 |
| WASC Id | 19 |
| Source ID | 1 |

|  |  |
| --- | --- |
| **High (Medium)** | **SQL Injection - Oracle - Time Based** |
| Description | SQL injection may be possible |
| URL | http://www.cypd.gov.tw/ |
| Method | GET |
| Parameter | query |
| Attack | field: [query], value [(SELECT UTL\_INADDR.get\_host\_name('10.0.0.1') from dual union SELECT UTL\_INADDR.get\_host\_name('10.0.0.2') from dual union SELECT UTL\_INADDR.get\_host\_name('10.0.0.3') from dual union SELECT UTL\_INADDR.get\_host\_name('10.0.0.4') from dual union SELECT UTL\_INADDR.get\_host\_name('10.0.0.5') from dual)] |
| Instances | 1 |
| 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 privilege 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 | The query time is controllable using parameter value [(SELECT UTL\_INADDR.get\_host\_name('10.0.0.1') from dual union SELECT UTL\_INADDR.get\_host\_name('10.0.0.2') from dual union SELECT UTL\_INADDR.get\_host\_name('10.0.0.3') from dual union SELECT UTL\_INADDR.get\_host\_name('10.0.0.4') from dual union SELECT UTL\_INADDR.get\_host\_name('10.0.0.5') from dual)], which caused the request to take [20,017] milliseconds, when the original unmodified query with value [query] took [634] milliseconds |
| Reference | https://www.owasp.org/index.php/Top\_10\_2010-A1  https://www.owasp.org/index.php/SQL\_Injection\_Prevention\_Cheat\_Sheet |
| CWE Id | 89 |
| WASC Id | 19 |
| Source ID | 1 |

|  |  |
| --- | --- |
| **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. |
| URL | http://www.cypd.gov.tw/ |
| Method | GET |
| Parameter | TS01542365 |
| Evidence | Set-Cookie: TS01542365 |
| Instances | 1 |
| Solution | Ensure that the HttpOnly flag is set for all cookies. |
| Reference | http://www.owasp.org/index.php/HttpOnly |
| CWE Id | 16 |
| WASC Id | 13 |
| Source ID | 3 |

|  |  |
| --- | --- |
| **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://www.cypd.gov.tw/ |
| Method | GET |
| Parameter | X-XSS-Protection |
| 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 |
| Source ID | 3 |

|  |  |
| --- | --- |
| **Low (Medium)** | **Absence of Anti-CSRF Tokens** |
| Description | No Anti-CSRF tokens were found in a HTML submission form.  A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.  CSRF attacks are effective in a number of situations, including:  \* The victim has an active session on the target site.  \* The victim is authenticated via HTTP auth on the target site.  \* The victim is on the same local network as the target site.  CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy. |
| URL | http://www.cypd.gov.tw/ |
| Method | GET |
| Evidence | <form action="/home/search" class="inside-search"> |
| Instances | 1 |
| Solution | Phase: Architecture and Design  Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.  For example, use anti-CSRF packages such as the OWASP CSRFGuard.  Phase: Implementation  Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.  Phase: Architecture and Design  Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).  Note that this can be bypassed using XSS.  Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.  Note that this can be bypassed using XSS.  Use the ESAPI Session Management control.  This control includes a component for CSRF.  Do not use the GET method for any request that triggers a state change.  Phase: Implementation  Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons. |
| Other information | No known Anti-CSRF tokens [anticsrf, CSRFToken, \_\_RequestVerificationToken, csrfmiddlewaretoken, authenticity\_token] were found in the following HTML forms: [Form 1: "cx" "cof" "ie" "q" "sa" ], [Form 2: "cx" "cof" "ie" "q" "sa" ]. |
| Reference | http://projects.webappsec.org/Cross-Site-Request-Forgery  http://cwe.mitre.org/data/definitions/352.html |
| CWE Id | 352 |
| WASC Id | 9 |
| Source ID | 3 |

|  |  |
| --- | --- |
| **Low (Medium)** | **Cross-Domain JavaScript Source File Inclusion** |
| Description | The page includes one or more script files from a third-party domain. |
|  | |
| URL | http://www.cypd.gov.tw/ |
| Method | GET |
| Parameter | http://www.google.com/jsapi |
| Evidence | <script src='http://www.google.com/jsapi'></script> |
| Instances | 1 |
| Solution | Ensure JavaScript source files are loaded from only trusted sources, and the sources can't be controlled by end users of the application. |
| Reference |  |
| CWE Id | 829 |
| WASC Id | 15 |
| Source ID | 3 |

|  |  |
| --- | --- |
| **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://www.cypd.gov.tw/ |
| Method | GET |
| Parameter | X-Content-Type-Options |
| 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 |
| Source ID | 3 |

|  |  |
| --- | --- |
| **Informational (Low)** | **Loosely Scoped Cookie** |
| Description | Cookies can be scoped by domain or path. This check is only concerned with domain scope.The domain scope applied to a cookie determines which domains can access it. For example, a cookie can be scoped strictly to a subdomain e.g. www.nottrusted.com, or loosely scoped to a parent domain e.g. nottrusted.com. In the latter case, any subdomain of nottrusted.com can access the cookie. Loosely scoped cookies are common in mega-applications like google.com and live.com. Cookies set from a subdomain like app.foo.bar are transmitted only to that domain by the browser. However, cookies scoped to a parent-level domain may be transmitted to the parent, or any subdomain of the parent. |
| URL | http://www.cypd.gov.tw/ |
| Method | GET |
| Instances | 1 |
| Solution | Always scope cookies to a FQDN (Fully Qualified Domain Name). |
| Other information | The origin domain used for comparison was:  www.cypd.gov.tw  ASP.NET\_SessionId=5gm5jlq4wmcs2ddurowqswiv  TS01542365=013991f1047b81ff3bbbd02ccc3f01ed6a23eeb317b467fd83e9f428bec57d4190a79e31b5afe6c145168173a05667200621ac8312 |
| Reference | https://tools.ietf.org/html/rfc6265#section-4.1  https://www.owasp.org/index.php/Testing\_for\_cookies\_attributes\_(OTG-SESS-002)  http://code.google.com/p/browsersec/wiki/Part2#Same-origin\_policy\_for\_cookies |
| CWE Id | 565 |
| WASC Id | 15 |
| Source ID | 3 |