Security Report

Scenario: appPenHaxHar.scenario

Mon Oct 14 09:07:45 UTC 2019

Step: 1.1) http://hax.tor.hu/

Alert Detail

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://hax.tor.hu/robots.txt
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16

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://hax.tor.hu/sitemap.xml
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco de.com/2014/03/guidelines-for-setting-security-headers/
WASC Id	14
CWE Id	933

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://hax.tor.hu/sitemap.xml?query=query+AND+1%3D1+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco de.com/2014/03/guidelines-for-setting-security-headers/

WASC Id	14
CWE Id	933
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://hax.tor.hu/board/?query=query+AND+1%3D1+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/board/?query=query+AND+1%3D1+--+
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/board/?query=query+AND+1%3D1+--+
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/login/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.

Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco de.com/2014/03/guidelines-for-setting-security-headers/
WASC Id	14
CWE Id	933
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://hax.tor.hu/login/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/login/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.as px
WASC Id	15
CWE Id	16
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://hax.tor.hu/html/haxtor.css
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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://hax.tor.hu/board/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/board/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/board/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/haxmin/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/haxmin/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/haxmin/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/peek/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
Low(Medium)	**Private IP Disclosure**
Description	A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.
URL	http://hax.tor.hu/peek/
Parameter	
Other information	10.0.0.5
Attack	
Solution	Remove the private IP address from the HTTP response body. For comments, use JSP/ASP/PHP comment instead of HTML/JavaScript comment which can be seen by client browsers.
Reference	https://tools.ietf.org/html/rfc1918
WASC Id	13
CWE Id	200
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://hax.tor.hu/peek/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/peek/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/html/haxtor.css?query=%2Fhaxtor.css
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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://hax.tor.hu/irc/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/irc/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/irc/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.as px
WASC Id	15
CWE Id	16
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://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco de.com/2014/03/guidelines-for-setting-security-headers/
WASC Id	14
CWE Id	933
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://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15

CWE Id	16
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.
URL	http://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
Low(Medium)	Cross-Domain JavaScript Source File Inclusion
Description	The page includes one or more script files from a third-party domain.
URL	http://hax.tor.hu/links/
Parameter	http://pagead2.googlesyndication.com/pagead/show_ads.js
Other information	
Attack	
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	
WASC Id	15
CWE Id	829
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://hax.tor.hu/links/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/links/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.

URL	http://hax.tor.hu/links/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/irc/?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/irc/?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/irc/?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/board/?query=query+AND+1%3D1
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/board/?query=query+AND+1%3D1
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/board/?query=query+AND+1%3D1
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/html/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/html/?query=query+AND+1%3D1+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933

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.
URL	http://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B
Parameter	query
Other information	
Attack	javascript:alert(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. Examples of libraries and framework

High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/board/?query=query+AND+1%3D1+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 AND 1=1 --] and [query AND 1=1 OR 1=1 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query AND 1=1 OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89

High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/html/haxtor.css?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query AND 1=2 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/sitemap.xml?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query AND 1=2 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/links/?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query OR 1=1 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--+%22+AND+%221%22%3D%221%22+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' -- " AND "1"="1" --] and [query' AND '1'='1' -- " AND "1"="2" --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The

	vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query' AND '1'='1' -- '' AND "1"="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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/stats/detect.txt?query=query+AND+1%3D1
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1] and [query OR 1=1] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query OR 1=1 --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/board/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	query

Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' --] and [query' AND '1'='2' --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query' AND '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/robots.txt?query=query+OR+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query OR 1=1 --] and [query AND 1=2 --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/login/?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query" AND "1"="1" --] and [query" AND "1"="2" --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query" AND "1"="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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
	SQL injection may be possible.

Description	
URL	http://hax.tor.hu/stats?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query" AND "1"="1" --] and [query" OR "1"="1" --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query" OR "1"="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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/peek/?all=1%27+AND+%271%27%3D%271
Parameter	all
Other information	The page results were successfully manipulated using the boolean conditions [1' AND '1'='1] and [1' AND '1'='2] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	1' AND '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/irc/?query=query%22+AND+%221%22%3D%221%22+--++OR+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query" AND "1"="1" -- OR 1=1 --] and [query" AND "1"="1" -- AND 1=2 --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query" AND "1"="1" -- OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/html?query=query%25
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query%] and [query%] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query%
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/board/?query=query%25
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query%] and [query%] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query%
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/irc/?query=query%25
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query%] and [query%] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query%
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B+ASC++--+
Parameter	query
Other information	The original page results were successfully replicated using the "ORDER BY" expression [javascript:alert(1); ASC --] as the parameter value The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison
Attack	javascript:alert(1); ASC --
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
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://hax.tor.hu/welcome/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/welcome/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/welcome/
Parameter	X-Content-Type-Options
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.
Attack	

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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/welcome/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/links/?query=query+AND+1%3D1+--+
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
Low(Medium)	Cross-Domain JavaScript Source File Inclusion
Description	The page includes one or more script files from a third-party domain.
URL	http://hax.tor.hu/links/?query=query+AND+1%3D1+--+
Parameter	http://pagead2.googlesyndication.com/pagead/show_ads.js
Other information	
Attack	
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	
WASC Id	15
CWE Id	829
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://hax.tor.hu/links/?query=query+AND+1%3D1+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/links/?query=query+AND+1%3D1+--+
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/links/?query=query+AND+1%3D1+--+
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/board/?query=query+AND+1%3D1
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13

CWE Id	16
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://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/peek/?all=1
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/peek/?all=1
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco de.com/2014/03/guidelines-for-setting-security-headers/
WASC Id	14
CWE Id	933
Low(Medium)	Private IP Disclosure
Description	A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.
URL	http://hax.tor.hu/peek/?all=1
Parameter	
Other information	10.0.0.5
Attack	
Solution	Remove the private IP address from the HTTP response body. For comments, use JSP/ASP/PHP comment instead of HTML/JavaScript comment which can be seen by client browsers.
Reference	https://tools.ietf.org/html/rfc1918
WASC Id	13
CWE Id	200
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://hax.tor.hu/peek/?all=1
Parameter	X-Content-Type-Options
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.
Attack	

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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/peek/?all=1
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/peek/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/peek/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
Low(Medium)	Private IP Disclosure
Description	A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.
URL	http://hax.tor.hu/peek/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	
Other information	10.0.0.5
Attack	
Solution	Remove the private IP address from the HTTP response body. For comments, use JSP/ASP/PHP comment instead of HTML/JavaScript comment which can be seen by client browsers.
Reference	https://tools.ietf.org/html/rfc1918
WASC Id	13
CWE Id	200
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://hax.tor.hu/peek/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/peek/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/board/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/haxmin/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/irc/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
	http://www.owasp.org/index.php/HttpOnly

Reference	
WASC Id	13
CWE Id	16
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://hax.tor.hu/irc/?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/peek/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/login/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/stats/detect.txt
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--++AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' -- AND 1=1 --] and [query' AND '1'='1' -- AND 1=2 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The

	vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query' AND '1'='1' -- AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/html/haxtor.css?query=%2Fhaxtor.css+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [/haxtor.css AND 1=1 --] and [/haxtor.css AND 1=2 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	/haxtor.css AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [javascript:alert(1); AND 1=1 --] and [javascript:alert(1); OR 1=1 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	javascript:alert(1); OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/html?query=query+AND+1%3D1+--+
Parameter	query

Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query AND 1=2 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/peek/?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query AND 1=2 --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/irc/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' --] and [query' AND '1'='2' --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query' AND '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
	SQL injection may be possible.

Description	
URL	http://hax.tor.hu/links/?query=query+AND+1%3D1+--+%25
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 -- %] and [query AND 1=1 -- XYZABCDEFGHIJ] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/board/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' --] and [query' AND '1'='2' --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query' AND '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/html/haxtor.css?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query" AND "1"="1" --] and [query" OR "1"="1" --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query" OR "1"="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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19

CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/stats?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' --] and [query' OR '1'='1' --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query' OR '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/?query=query%25
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query%] and [queryXYZABCDEFGHIJ] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query%
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/peek/?all=1+AND+1%3D1+--+
Parameter	all
Other information	The page results were successfully manipulated using the boolean conditions [1 AND 1=1 --] and [1 OR 1=1 --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	1 OR 1=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 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.

Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/stats/?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query AND 1=2 --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' --] and [query' OR '1'='1' --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query' OR '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/board/?query=query+AND+1%3D1%27+AND+%271%27%3D%271
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1' AND '1'='1] and [query AND 1=1' AND '1'='2] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=1' AND '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Preventi on_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/links/?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query" AND "1"="1" --] and [query" OR "1"="1" --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query" OR "1"="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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Preventi on_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/irc/?query=query%22+AND+%221%22%3D%221%22+--+%22+AND+%221%22%3 D%221%22+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query" AND "1"="1" -- " AND "1"="1" --] and [query" AND "1"="1" -- " OR "1"="1" --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query" AND "1"="1" -- " OR "1"="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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Preventi on_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/sitemap.xml?query=query+ASC++--+
Parameter	query
Other information	The original page results were successfully replicated using the "ORDER BY" expression [query ASC --] as the parameter value The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison
Attack	query ASC --
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/stats/?graph=3-2&h=400&mode=top5comparison&topfrom=11&w=800
Parameter	graph
Other information	The original page results were successfully replicated using the expression [3-2] as the parameter value The parameter value being modified was stripped from the HTML output for the purposes of the comparison
Attack	3-2
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
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://hax.tor.hu/welcome/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/welcome/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/welcome/
Parameter	X-Content-Type-Options

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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/welcome/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16

Step: 1.2) http://hax.tor.hu/

Alert Detail

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://hax.tor.hu/robots.txt
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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://hax.tor.hu/sitemap.xml
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco

Reference	de.com/2014/03/guidelines-for-setting-security-headers/
WASC Id	14
CWE Id	933
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://hax.tor.hu/sitemap.xml?query=query+AND+1%3D1+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco de.com/2014/03/guidelines-for-setting-security-headers/
WASC Id	14
CWE Id	933
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://hax.tor.hu/board/?query=query+AND+1%3D1+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco de.com/2014/03/guidelines-for-setting-security-headers/
WASC Id	14
CWE Id	933
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://hax.tor.hu/board/?query=query+AND+1%3D1+--+
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/board/?query=query+AND+1%3D1+--+
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.as px
WASC Id	15
CWE Id	16
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://hax.tor.hu/login/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco de.com/2014/03/guidelines-for-setting-security-headers/
WASC Id	14
CWE Id	933
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://hax.tor.hu/login/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/login/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.as px
WASC Id	15
CWE Id	16
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://hax.tor.hu/html/haxtor.css
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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://hax.tor.hu/board/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/board/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/board/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/haxmin/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/haxmin/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/haxmin/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/peek/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
Low(Medium)	Private IP Disclosure
Description	A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.

URL	http://hax.tor.hu/peek/
Parameter	
Other information	10.0.0.5
Attack	
Solution	Remove the private IP address from the HTTP response body. For comments, use JSP/ASP/PHP comment instead of HTML/JavaScript comment which can be seen by client browsers.
Reference	https://tools.ietf.org/html/rfc1918
WASC Id	13
CWE Id	200
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://hax.tor.hu/peek/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/peek/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/html/haxtor.css?query=%2Fhaxtor.css
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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
	http://hax.tor.hu/irc/

URL	
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/irc/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/irc/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.as px
WASC Id	15
CWE Id	16
Low(Medium)	Cross-Domain JavaScript Source File Inclusion
Description	The page includes one or more script files from a third-party domain.
URL	http://hax.tor.hu/links/
Parameter	http://pagead2.googlesyndication.com/pagead/show_ads.js
Other information	
Attack	
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	
WASC Id	15
CWE Id	829
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://hax.tor.hu/links/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco de.com/2014/03/guidelines-for-setting-security-headers/
WASC Id	14
CWE Id	933
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://hax.tor.hu/links/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/links/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/irc/?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/irc/?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16

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.
URL	http://hax.tor.hu/irc/?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/board/?query=query+AND+1%3D1
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/board/?query=query+AND+1%3D1
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/board/?query=query+AND+1%3D1
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16

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://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco de.com/2014/03/guidelines-for-setting-security-headers/
WASC Id	14
CWE Id	933
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://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.as px
WASC Id	15
CWE Id	16
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://hax.tor.hu/html/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco de.com/2014/03/guidelines-for-setting-security-headers/

WASC Id	14
CWE Id	933
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://hax.tor.hu/html/?query=query+AND+1%3D1+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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.
URL	http://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B
Parameter	query
Other information	
Attack	javascript:alert(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. Examples of libraries and framework
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/board/?query=query+AND+1%3D1+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 AND 1=1 --] and [query AND 1=1 OR 1=1 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query AND 1=1 OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/html/haxtor.css?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query AND 1=2 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/sitemap.xml?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query AND 1=2 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/links/?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query OR 1=1 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--+%22+AND+%221%22%3D%221%22+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' -- " AND "1"="1" --] and [query' AND '1'='1' -- " AND "1"="2" --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query' AND '1'='1' -- " AND "1"="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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/stats/detect.txt?query=query+AND+1%3D1
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1] and [query OR 1=1] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query OR 1=1 --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Preventi on_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/board/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' --] and [query' AND '1'='2' --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query' AND '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Preventi on_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/robots.txt?query=query+OR+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query OR 1=1 --] and [query AND 1=2 --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Preventi on_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/login/?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query" AND "1"="1" --] and [query" AND "1"="2" --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data

	was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query" AND "1"="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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/stats?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query" AND "1"="1" --] and [query" OR "1"="1" --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query" OR "1"="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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/peek/?all=1%27+AND+%271%27%3D%271
Parameter	all
Other information	The page results were successfully manipulated using the boolean conditions [1' AND '1'='1] and [1' AND '1'='2] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	1' AND '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/irc/?query=query%22+AND+%221%22%3D%221%22+--++OR+1%3D1+--+
Parameter	query

Other information	The page results were successfully manipulated using the boolean conditions [query" AND "1"="1" -- OR 1=1 --] and [query" AND "1"="1" -- AND 1=2 --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query" AND "1"="1" -- OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/html?query=query%25
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query%] and [query%] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query%
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/board/?query=query%25
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query%] and [query%] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query%
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection

Description	SQL injection may be possible.
URL	http://hax.tor.hu/irc/?query=query%25
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query%] and [query%] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query%
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B+ASC++--+
Parameter	query
Other information	The original page results were successfully replicated using the "ORDER BY" expression [javascript:alert(1); ASC --] as the parameter value The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison
Attack	javascript:alert(1); ASC --
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
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://hax.tor.hu/welcome/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/welcome/
Parameter	X-XSS-Protection
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).
Attack	

Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/welcome/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/welcome/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/links/?query=query+AND+1%3D1+--+
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
Low(Medium)	**Cross-Domain JavaScript Source File Inclusion**
Description	The page includes one or more script files from a third-party domain.
URL	http://hax.tor.hu/links/?query=query+AND+1%3D1+--+
Parameter	http://pagead2.googlesyndication.com/pagead/show_ads.js
Other information	
Attack	
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	
WASC Id	15
CWE Id	829
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
	http://hax.tor.hu/links/?query=query+AND+1%3D1+--+

URL	
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/links/?query=query+AND+1%3D1+--+
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/links/?query=query+AND+1%3D1+--+
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B

Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/board/?query=query+AND+1%3D1
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/peek/?all=1
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/peek/?all=1
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco de.com/2014/03/guidelines-for-setting-security-headers/
WASC Id	14
CWE Id	933
Low(Medium)	Private IP Disclosure
Description	A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.
URL	http://hax.tor.hu/peek/?all=1
Parameter	
Other information	10.0.0.5

Attack	
Solution	Remove the private IP address from the HTTP response body. For comments, use JSP/ASP/PHP comment instead of HTML/JavaScript comment which can be seen by client browsers.
Reference	https://tools.ietf.org/html/rfc1918
WASC Id	13
CWE Id	200
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://hax.tor.hu/peek/?all=1
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/peek/?all=1
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.as px
WASC Id	15
CWE Id	16
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://hax.tor.hu/peek/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/peek/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veraco

	de.com/2014/03/guidelines-for-setting-security-headers/
WASC Id	14
CWE Id	933
Low(Medium)	Private IP Disclosure
Description	A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.
URL	http://hax.tor.hu/peek/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	
Other information	10.0.0.5
Attack	
Solution	Remove the private IP address from the HTTP response body. For comments, use JSP/ASP/PHP comment instead of HTML/JavaScript comment which can be seen by client browsers.
Reference	https://tools.ietf.org/html/rfc1918
WASC Id	13
CWE Id	200
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://hax.tor.hu/peek/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/peek/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/board/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/haxmin/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/irc/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/irc/?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/peek/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/login/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/stats/detect.txt
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--++AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' -- AND 1=1 --] and [query' AND '1'='1' -- AND 1=2 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query' AND '1'='1' -- AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/html/haxtor.css?query=%2Fhaxtor.css+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [/haxtor.css AND 1=1 --] and [/haxtor.css AND 1=2 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	/haxtor.css AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/login/?query=javascript%3Aalert%281%29%3B+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [javascript:alert(1); AND 1=1 --] and [javascript:alert(1); OR 1=1 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	javascript:alert(1); OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Preventi on_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/html?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query AND 1=2 --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Preventi on_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/peek/?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query AND 1=2 --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Preventi on_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/irc/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' --] and [query' AND '1'='2' --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data

	was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query' AND '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/links/?query=query+AND+1%3D1+--+%25
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 -- %] and [query AND 1=1 -- XYZABCDEFGHIJ] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/board/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' --] and [query' AND '1'='2' --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query' AND '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/html/haxtor.css?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	query

Other information	The page results were successfully manipulated using the boolean conditions [query" AND "1"="1" --] and [query" OR "1"="1" --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query" OR "1"="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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/stats?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' --] and [query' OR '1'='1' --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query' OR '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/?query=query%25
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query%] and [queryXYZABCDEFGHIJ] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query%
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
	SQL injection may be possible.

Description	
URL	http://hax.tor.hu/peek/?all=1+AND+1%3D1+--+
Parameter	all
Other information	The page results were successfully manipulated using the boolean conditions [1 AND 1=1 --] and [1 OR 1=1 --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	1 OR 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/stats/?query=query+AND+1%3D1+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1 --] and [query AND 1=2 --] The parameter value being modified was stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	**SQL Injection**
Description	SQL injection may be possible.
URL	http://hax.tor.hu/haxmin/?query=query%27+AND+%271%27%3D%271%27+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query' AND '1'='1' --] and [query' OR '1'='1' --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query' OR '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19

CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/board/?query=query+AND+1%3D1%27+AND+%271%27%3D%271
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query AND 1=1' AND '1'='1] and [query AND 1=1' AND '1'='2] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was returned for the original parameter. The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter
Attack	query AND 1=1' AND '1'='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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/links/?query=query%22+AND+%221%22%3D%221%22+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query" AND "1"="1" --] and [query" OR "1"="1" --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query" OR "1"="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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/irc/?query=query%22+AND+%221%22%3D%221%22+--+%22+AND+%221%22%3D%221%22+--+
Parameter	query
Other information	The page results were successfully manipulated using the boolean conditions [query" AND "1"="1" -- " AND "1"="1" --] and [query" AND "1"="1" -- " OR "1"="1" --] The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison Data was NOT returned for the original parameter. The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter
Attack	query" AND "1"="1" -- " OR "1"="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 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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Preventi on_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/sitemap.xml?query=query+ASC++--+
Parameter	query
Other information	The original page results were successfully replicated using the "ORDER BY" expression [query ASC --] as the parameter value The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison
Attack	query ASC --
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Preventi on_Cheat_Sheet
WASC Id	19
CWE Id	89
High(Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://hax.tor.hu/stats/?graph=3-2&h=400&mode=top5comparison&topfrom=11&w=800
Parameter	graph
Other information	The original page results were successfully replicated using the expression [3-2] as the parameter value The parameter value being modified was stripped from the HTML output for the purposes of the comparison
Attack	3-2
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.
Reference	https://www.owasp.org/index.php/Top_10_2010-A1 https://www.owasp.org/index.php/SQL_Injection_Preventi on_Cheat_Sheet
WASC Id	19
CWE Id	89
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://hax.tor.hu/welcome/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16
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://hax.tor.hu/welcome/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/welcome/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/welcome/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16

Step: 1.3) http://hax.tor.hu/irc/

Alert Detail

The alert list is empty

Step: 1.4) http://hax.tor.hu/board/

Alert Detail

The alert list is empty

Step: 1.5) http://hax.tor.hu/peek/

Alert Detail

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://hax.tor.hu/peek/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
Low(Medium)	Private IP Disclosure
Description	A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.
URL	http://hax.tor.hu/peek/
Parameter	
Other information	10.0.0.5
Attack	
Solution	Remove the private IP address from the HTTP response body. For comments, use JSP/ASP/PHP comment instead of HTML/JavaScript comment which can be seen by client browsers.
Reference	https://tools.ietf.org/html/rfc1918
WASC Id	13
CWE Id	200
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://hax.tor.hu/peek/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/peek/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/peek/?all=1
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
Low(Medium)	Private IP Disclosure
Description	A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.
URL	http://hax.tor.hu/peek/?all=1
Parameter	
Other information	10.0.0.5
Attack	
Solution	Remove the private IP address from the HTTP response body. For comments, use JSP/ASP/PHP comment instead of HTML/JavaScript comment which can be seen by client browsers.
Reference	https://tools.ietf.org/html/rfc1918
WASC Id	13
CWE Id	200
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://hax.tor.hu/peek/?all=1
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/peek/?all=1
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/peek/

Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16

Step: 1.6) http://hax.tor.hu/haxmin/

Alert Detail

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://hax.tor.hu/haxmin/
Parameter	X-XSS-Protection
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).
Attack	
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
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/
WASC Id	14
CWE Id	933
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://hax.tor.hu/haxmin/
Parameter	X-Content-Type-Options
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.
Attack	
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.
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
WASC Id	15
CWE Id	16
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.
URL	http://hax.tor.hu/haxmin/
Parameter	X-Frame-Options
Other information	
Attack	
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).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
WASC Id	15
CWE Id	16
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://hax.tor.hu/haxmin/
Parameter	HAXTOR
Other information	
Attack	
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
WASC Id	13
CWE Id	16