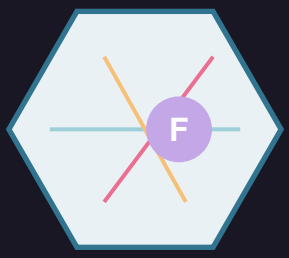
FFUF

Fuzz Faster U Fool

Comprehensive Cheat Sheet



Fast Web Fuzzing Tool

For Directory Discovery, Parameter Testing,
Subdomain Enumeration & Security Assessment

Version: v2.0
Updated: November 2, 2025
Purpose: Ethical Hacking & Penetration Testing

A comprehensive reference guide compiled from multiple sources
for authorized security testing only

Contents

1 Introduction to FFUF

1.1 What is FFUF?

FFUF (Fuzz Faster U Fool) is a high-performance web fuzzing tool written in Go, designed for discovering hidden web content such as files, directories, subdomains, parameters, and virtual hosts through systematic brute-forcing with wordlists.

Information

FFUF is prized for its exceptional speed, flexibility, and extensive filtering capabilities, making it a staple tool in every pentester's arsenal alongside DirBuster and Gobuster.

1.2 Key Features

- **Lightning Fast:** Multi-threaded Go implementation
- **Highly Flexible:** Fuzz URLs, headers, parameters, POST data
- **Advanced Filtering:** By status code, size, words, lines, regex
- **Multiple Wordlists:** Use several wordlists simultaneously
- **Recursive Fuzzing:** Discover nested directories automatically
- **Output Formats:** JSON, HTML, Markdown, CSV support
- **Integration Ready:** Works with Burp Suite and other tools
- **Custom Headers:** Full control over HTTP requests

1.3 Installation

1.3.1 Quick Install (Linux)

```
# Ubuntu/Debian/Kali
sudo apt update && sudo apt install ffuf

# Using Go (latest version)
go install github.com/ffuf/ffuf/v2@latest

# Verify installation
ffuf -V
```

1.3.2 Other Platforms

```
# macOS (Homebrew)
brew install ffuf

# Arch Linux
sudo pacman -S ffuf

# From source
git clone https://github.com/ffuf/ffuf
cd ffuf
```

```
go build
sudo mv ffuf /usr/local/bin/
```

1.4 The FUZZ Keyword

The power of FFUF lies in the **FUZZ** keyword. Place it anywhere in:

- **URLs:** `http://target.com/FUZZ`
- **Headers:** `-H "Host: FUZZ.target.com"`
- **POST Data:** `-d "username=FUZZ&password=test"`
- **Parameters:** `http://target.com/page?id=FUZZ`

Pro Tip

You can use multiple FUZZ keywords with different names: FUZZ, FUZ2Z, FUZ3Z, etc. Each can reference a different wordlist!

1.5 Basic Syntax

Essential Command Structure

```
ffuf -u <URL> -w <WORDLIST> [OPTIONS]
```

Core Components:

- `-u` : Target URL with FUZZ keyword
- `-w` : Path to wordlist file
- `-H` : Custom HTTP headers
- `-X` : HTTP method (GET, POST, PUT, DELETE, etc.)
- `-d` : POST data

1.6 Common Wordlist Locations

```
# SecLists (most comprehensive)
/usr/share/seclists/

# Dirb
/usr/share/wordlists/dirb/

# Common locations
/usr/share/wordlists/
/usr/share/seclists/Discovery/Web-Content/
/usr/share/seclists/Discovery/DNS/
/usr/share/seclists/Fuzzing/
```

Warning

Always ensure you have proper authorization before conducting fuzzing activities. Unauthorized testing is illegal and unethical.

2 Basic Fuzzing Techniques

2.1 Directory and File Discovery

2.1.1 Simple Directory Fuzzing

```
# Basic directory enumeration
ffuf -u http://target.com/FUZZ -w /usr/share/wordlists/dirb/common.txt

# With increased threads for speed
ffuf -u http://target.com/FUZZ -w /usr/share/wordlists/dirb/common.txt -t
    100

# Colored output for better visibility
ffuf -u http://target.com/FUZZ -w /usr/share/wordlists/dirb/common.txt -c
```

2.1.2 File Fuzzing with Extensions

```
# Single extension
ffuf -u http://target.com/FUZZ.php -w /usr/share/wordlists/dirb/common.txt

# Multiple extensions
ffuf -u http://target.com/FUZZ -w common.txt -e .php,.html,.txt,.js,.xml

# Common web extensions
ffuf -u http://target.com/FUZZ -w common.txt \
     -e .php,.asp,.aspx,.jsp,.html,.js,.txt,.pdf,.zip
```

Pro Tip

Use the `-e` flag to append extensions to each wordlist entry automatically. This is much more efficient than creating separate wordlists for each extension.

2.2 Recursive Fuzzing

```
# Enable recursion with depth limit
ffuf -u http://target.com/FUZZ -w common.txt -recursion -recursion-depth 2

# Recursive with extensions
ffuf -u http://target.com/FUZZ -w custom-wordlist.txt \
     -recursion -recursion-depth 3 -e .php,.asp,.html

# Limit time per recursive branch
ffuf -u http://target.com/FUZZ -w wordlist.txt \
     -recursion -recursion-depth 2 -maxtime-job 60
```

Note

When using recursion, the FUZZ keyword must be at the END of the URL. The recursion will append discovered paths and continue fuzzing.

2.3 Subdomain Enumeration

2.3.1 Virtual Host Discovery (Host Header)

```
# Basic subdomain enumeration via Host header
ffuf -u http://target.com/ -w subdomains.txt -H "Host: FUZZ.target.com"

# With common subdomains wordlist
ffuf -u http://target.com/ \
     -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt \
     -H "Host: FUZZ.target.com"

# HTTPS subdomain fuzzing
ffuf -u https://target.com/ \
     -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-110000.txt \
     -H "Host: FUZZ.target.com" -mc 200,301,302,403
```

2.3.2 Direct Subdomain Fuzzing

```
# Fuzz subdomain directly in URL
ffuf -u http://FUZZ.target.com/ -w subdomains.txt -t 100

# With specific status code matching
ffuf -u http://FUZZ.target.com/ \
     -w /usr/share/seclists/Discovery/DNS/bitquark-subdomains-top100000.txt \
     -mc 200,301,302,403 -t 100
```

2.3.3 Real-World Example (HTB board.htb)

Example

```
# Finding virtual hosts on board.htb
ffuf -w /usr/share/wordlists/SecLists/Discovery/DNS/bitquark-subdomains-top100000.txt:FUZZ \
     -u http://board.htb/ \
     -H 'Host: FUZZ.board.htb' \
     -fs 15949
```

This command:

- Uses a comprehensive subdomain wordlist
- Tests virtual hosts on board.htb
- Filters out responses with size 15949 bytes (default error page)
- Reveals hidden subdomains like crm.board.htb, dev.board.htb

2.4 Parameter Fuzzing

2.4.1 GET Parameter Name Discovery

```
# Fuzz GET parameter names
ffuf -u http://target.com/page.php?FUZZ=test_value \
     -w /usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt

# Filter by response size
ffuf -u http://target.com/script.php?FUZZ=test \
     -w parameter-names.txt -fs 4242
```

2.4.2 GET Parameter Value Fuzzing

```
# When parameter name is known, fuzz values
ffuf -u http://target.com/page.php?id=FUZZ -w values.txt

# Filter out 401 Unauthorized responses
ffuf -u http://target.com/script.php?valid_name=FUZZ \
     -w values.txt -fc 401

# Numeric value fuzzing
ffuf -u http://target.com/user.php?id=FUZZ \
     -w <(seq 1 1000) -mc 200
```

Pro Tip

Use `seq` to generate numeric sequences on-the-fly: `ffuf -w <(seq 0 255) -u http://target/page?id=FUZZ`

3 Advanced Fuzzing Techniques

3.1 POST Request Fuzzing

3.1.1 Basic POST Data Fuzzing

```
# Fuzz POST parameter value
ffuf -u http://target.com/login.php \
     -X POST \
     -d "username=admin&password=FUZZ" \
     -w passwords.txt \
     -H "Content-Type: application/x-www-form-urlencoded" \
     -fc 401

# Fuzz username field
ffuf -u http://target.com/login.php \
     -X POST \
     -d "username=FUZZ&password=test123" \
     -w usernames.txt \
     -H "Content-Type: application/x-www-form-urlencoded"
```

3.1.2 JSON POST Data Fuzzing

```
# Fuzz JSON values
ffuf -u http://target.com/api/login \
     -X POST \
     -d '{"username":"admin","password":"FUZZ"}' \
     -w passwords.txt \
     -H "Content-Type: application/json" \
     -fr "error"

# Multiple JSON fields
ffuf -u http://target.com/api/auth \
     -X POST \
     -d '{"email":"FUZZ","token":"test"}' \
     -w emails.txt \
     -H "Content-Type: application/json"
```

3.2 Multiple Wordlists (Clusterbomb, Pitchfork, Sniper)

FFUF supports three fuzzing modes when using multiple wordlists:

3.2.1 Clusterbomb Mode (Default)

Tests all possible combinations of wordlists.

```
# All combinations of users and passwords
ffuf -u http://target.com/login \
     -X POST \
     -d "username=USER&password=PASS" \
     -w usernames.txt:USER \
     -w passwords.txt:PASS \
```

```
      -mode clusterbomb \
      -H "Content-Type: application/x-www-form-urlencoded"

# Multiple URL positions
ffuf -u http://target.com/FUZZ/FUZ2Z \
      -w dirs.txt:FUZZ \
      -w files.txt:FUZ2Z \
      -mode clusterbomb
```

Note

Clusterbomb mode can generate MANY requests. If you have 100 usernames and 1000 passwords, that's 100,000 requests!

3.2.2 Pitchfork Mode

Tests wordlists in parallel (first line of each, then second line, etc.).

```
# Parallel testing (line by line)
ffuf -u http://target.com/login \
      -X POST \
      -d "username=USER&password=PASS" \
      -w users.txt:USER \
      -w passes.txt:PASS \
      -mode pitchfork

# Useful for testing user:pass combinations from breach data
ffuf -u http://target.com/api/auth \
      -X POST \
      -d '{"user":"USER","pass":"PASS"}' \
      -w leaked-users.txt:USER \
      -w leaked-passes.txt:PASS \
      -mode pitchfork \
      -H "Content-Type: application/json"
```

3.2.3 Sniper Mode

Tests one position at a time, keeping others static.

```
# Focus on one parameter
ffuf -u http://target.com/search?q=FUZZ&category=FUZ2Z \
      -w search-terms.txt:FUZZ \
      -w categories.txt:FUZ2Z \
      -mode sniper
```

3.3 Authentication and Cookie Fuzzing

3.3.1 Cookie-Based Authentication

```
# Fuzz with cookies
ffuf -u http://target.com/admin/FUZZ \
      -w wordlist.txt \
```

```
    -b "session=abc123; user=admin"

# Fuzz cookie values
ffuf -u http://target.com/profile \
    -w session-ids.txt \
    -b "PHPSESSID=FUZZ"
```

3.3.2 Authorization Header Fuzzing

```
# Bearer token fuzzing
ffuf -u http://target.com/api/admin/FUZZ \
    -w endpoints.txt \
    -H "Authorization: Bearer eyJhbGc...token...xyz"

# Basic auth fuzzing
ffuf -u http://target.com/admin/FUZZ \
    -w admin-paths.txt \
    -H "Authorization: Basic YWRtaW46cGFzc3dvcmQ="
```

3.3.3 JWT Token Fuzzing

```
# Fuzz JWT secrets
ffuf -u http://target.com/api/admin \
    -H "Authorization: Bearer eyJ0eXAi...FUZZ" \
    -w /usr/share/seclists/Fuzzing/jwt-secrets.txt \
    -fr "Invalid token" \
    -c

# Test different JWT algorithms
ffuf -u http://target.com/api/validate \
    -X POST \
    -d '{"token":"FUZZ"}' \
    -w jwt-tokens.txt \
    -H "Content-Type: application/json"
```

3.4 Custom Header Fuzzing

```
# User-Agent fuzzing
ffuf -u http://target.com/FUZZ \
    -w wordlist.txt \
    -H "User-Agent: FUZZ" \
    -w user-agents.txt

# Multiple custom headers
ffuf -u http://target.com/admin \
    -w paths.txt \
    -H "X-Forwarded-For: 127.0.0.1" \
    -H "X-Originating-IP: 127.0.0.1" \
    -H "X-Remote-IP: 127.0.0.1" \
    -H "X-Client-IP: 127.0.0.1"

# Fuzz header values
```

```
ffuf -u http://target.com/api \
     -w ips.txt \
     -H "X-Forwarded-For: FUZZ"
```

Pro Tip

Multiple `-H` flags can bypass some WAF rules. Try adding headers like X-Forwarded-For, X-Originating-IP, and X-Remote-Addr with localhost values.

3.5 Using Raw HTTP Requests

```
# Create a raw HTTP request file
cat > request.txt << 'EOF'
POST /api/login HTTP/1.1
Host: target.com
Content-Type: application/json
Content-Length: 45

{"username":"admin","password":"FUZZ"}
EOF

# Use the raw request
ffuf -request request.txt -request-proto https -w passwords.txt
```

3.6 Dynamic Data Generation

3.6.1 Generate Data On-The-Fly

```
# Generate numeric range
seq 0 255 | ffuf -u 'http://target.com/user?id=FUZZ' -w - -c

# Using Ruby for range generation
ruby -e '(0..255).each{|i| puts i}' | ffuf -u 'http://target.com/?id=FUZZ'
     -w -

# Using Bash loops
for i in {0..255}; do echo $i; done | \
    ffuf -u 'http://target.com/?id=FUZZ' -w -

# Date range generation
for d in {1..31}; do echo "2024-01-$d"; done | \
    ffuf -u 'http://target.com/logs/FUZZ.log' -w -
```

3.6.2 Path Traversal Sequences

```
# Generate path traversal payloads
for i in {1..10}; do printf '../%.0s' $(seq 1 $i); echo; done | \
    ffuf -u 'http://target.com/file=FUZZetc/passwd' -w -

# Encoded path traversal
echo -e "../\n../../\n../../../" | \
```

```
ffuf -u 'http://target.com/?file=FUZZ' -w -
```

4 Filtering and Matching

4.1 Understanding Matchers and Filters

Information

Matchers (-m): Include only responses that match criteria
Filters (-f): Exclude responses that match criteria
Both can be combined for powerful result refinement!

4.2 Status Code Filtering

4.2.1 Match Status Codes (-mc)

```
# Match only 200 responses
ffuf -u http://target.com/FUZZ -w wordlist.txt -mc 200

# Match multiple status codes
ffuf -u http://target.com/FUZZ -w wordlist.txt -mc 200,301,302

# Match all except 404
ffuf -u http://target.com/FUZZ -w wordlist.txt -mc all -fc 404

# Common success codes
ffuf -u http://target.com/FUZZ -w wordlist.txt -mc
    200,204,301,302,307,401,403
```

4.2.2 Filter Status Codes (-fc)

```
# Filter out 404 Not Found
ffuf -u http://target.com/FUZZ -w wordlist.txt -fc 404

# Filter multiple codes
ffuf -u http://target.com/FUZZ -w wordlist.txt -fc 404,403,500

# Filter ranges
ffuf -u http://target.com/FUZZ -w wordlist.txt -fc 400-499
```

4.3 Response Size Filtering

4.3.1 Filter by Size (-fs)

```
# Filter exact size (common for error pages)
ffuf -u http://target.com/FUZZ -w wordlist.txt -fs 4242

# Filter size range
ffuf -u http://target.com/FUZZ -w wordlist.txt -fs 1-100,500-1024

# Multiple sizes
ffuf -u http://target.com/FUZZ -w wordlist.txt -fs 615,628,4242
```

Example

Real-world scenario: When fuzzing board.htb subdomains, all invalid hosts returned 15949 bytes. Using `-fs 15949` filtered out noise and revealed valid subdomains.

```
ffuf -u http://board.htb/ \
     -H 'Host: FUZZ.board.htb' \
     -w subdomains.txt \
     -fs 15949
```

4.3.2 Match by Size (-ms)

```
# Match specific response size
ffuf -u http://target.com/FUZZ -w wordlist.txt -ms 1337

# Match size range
ffuf -u http://target.com/FUZZ -w wordlist.txt -ms 1000-2000
```

4.4 Word and Line Filtering

4.4.1 Filter by Word Count (-fw)

```
# Filter responses with exactly 100 words
ffuf -u http://target.com/FUZZ -w wordlist.txt -fw 100

# Filter multiple word counts
ffuf -u http://target.com/FUZZ -w wordlist.txt -fw 33,42,100
```

4.4.2 Match by Word Count (-mw)

```
# Match specific word count
ffuf -u http://target.com/FUZZ -w wordlist.txt -mw 1337

# Match word count range
ffuf -u http://target.com/FUZZ -w wordlist.txt -mw 500-1000
```

4.4.3 Filter by Line Count (-fl)

```
# Filter exact line count
ffuf -u http://target.com/FUZZ -w wordlist.txt -fl 25

# Multiple line counts
ffuf -u http://target.com/FUZZ -w wordlist.txt -fl 10,20,30
```

4.4.4 Match by Line Count (-ml)

```
# Match responses with specific lines
ffuf -u http://target.com/FUZZ -w wordlist.txt -ml 42
```

4.5 Regex Pattern Filtering

4.5.1 Filter by Regex (-fr)

```
# Filter responses containing "error"
ffuf -u http://target.com/FUZZ -w wordlist.txt -fr "error"

# Filter multiple patterns
ffuf -u http://target.com/FUZZ -w wordlist.txt -fr "error|forbidden|denied
    "

# Case-insensitive filtering
ffuf -u http://target.com/api/FUZZ -w endpoints.txt -fr "(?i)invalid"
```

4.5.2 Match by Regex (-mr)

```
# Match responses containing pattern
ffuf -u http://target.com/FUZZ -w wordlist.txt -mr "success|admin|password
    "

# Match specific HTML elements
ffuf -u http://target.com/FUZZ -w wordlist.txt -mr "<title>Admin"

# Match JSON responses
ffuf -u http://target.com/api/FUZZ -w endpoints.txt -mr '"status":"ok"'
```

4.6 Time-Based Filtering

4.6.1 Filter by Response Time (-ft)

```
# Filter responses faster than 2 seconds
ffuf -u http://target.com/FUZZ -w wordlist.txt -ft 2000

# Detect slow endpoints (potential SQL injection)
ffuf -u http://target.com/page?id=FUZZ \
     -w sqli-payloads.txt \
     -ft 5000 \
     -mr "error"
```

4.7 Combining Filters and Matchers

```
# Complex filtering example
ffuf -u http://target.com/FUZZ \
     -w wordlist.txt \
```

```
      -mc 200,301,302 \
      -fs 4242 \
      -fw 33 \
      -fr "error|forbidden" \
      -c -v

# Match all, filter specific
ffuf -u http://target.com/FUZZ \
      -w wordlist.txt \
      -mc all \
      -fc 404 \
      -fs 615 \
      -c

# Precise targeting
ffuf -u http://target.com/api/FUZZ \
      -w endpoints.txt \
      -mc 200 \
      -mr '"success":true' \
      -fs 42 \
      -c -v
```

Pro Tip

Start with broad matching (`-mc all`) and progressively add filters (`-fc`, `-fs`, `-fw`) to eliminate false positives. This approach is more efficient than trying to guess the right filters upfront.

4.8 Filter Mode

```
# AND mode (default) - all filters must match
ffuf -u http://target.com/FUZZ -w wordlist.txt -fs 42 -fw 10 -fmode and

# OR mode - any filter match excludes result
ffuf -u http://target.com/FUZZ -w wordlist.txt -fs 42 -fw 10 -fmode or
```

4.9 Calibration Mode

FFUF can automatically detect common response patterns and filter them out.

```
# Auto-calibration (makes a few test requests first)
ffuf -u http://target.com/FUZZ -w wordlist.txt -ac

# Calibration with keyword strategy
ffuf -u http://target.com/FUZZ -w wordlist.txt -ac -acc "random"
```

Note

Auto-calibration (`-ac`) sends test requests with random values to learn what a typical "not found" response looks like, then automatically filters similar responses.

5 Performance Optimization & Evasion

5.1 Thread and Rate Control

5.1.1 Thread Management

```
# Default threads (40)
ffuf -u http://target.com/FUZZ -w wordlist.txt

# Increase threads for speed
ffuf -u http://target.com/FUZZ -w wordlist.txt -t 100

# Decrease threads for stability
ffuf -u http://target.com/FUZZ -w wordlist.txt -t 10

# Maximum speed (use with caution)
ffuf -u http://target.com/FUZZ -w wordlist.txt -t 200
```

Warning

Too many threads can:

- Overwhelm the target server (DoS)
- Trigger WAF/IPS detection
- Result in connection errors
- Get your IP banned

Start with 40-50 threads and increase gradually while monitoring results.

5.1.2 Rate Limiting

```
# Limit to 50 requests per second
ffuf -u http://target.com/FUZZ -w wordlist.txt -rate 50

# Conservative approach (10 req/sec)
ffuf -u http://target.com/FUZZ -w wordlist.txt -rate 10

# Combine with threads
ffuf -u http://target.com/FUZZ -w wordlist.txt -t 20 -rate 30
```

5.1.3 Request Delays

```
# Fixed delay of 0.1 seconds between requests
ffuf -u http://target.com/FUZZ -w wordlist.txt -p 0.1

# Random delay between 0.5-1.5 seconds
ffuf -u http://target.com/FUZZ -w wordlist.txt -p 0.5-1.5

# Long delay for stealth
ffuf -u http://target.com/FUZZ -w wordlist.txt -p 2-5 -t 10
```

5.2 Timeout Settings

```
# Set timeout to 10 seconds (default)
ffuf -u http://target.com/FUZZ -w wordlist.txt -timeout 10

# Longer timeout for slow servers
ffuf -u http://target.com/FUZZ -w wordlist.txt -timeout 30

# Quick timeout for fast scanning
ffuf -u http://target.com/FUZZ -w wordlist.txt -timeout 5
```

5.3 Maximum Execution Time

```
# Stop after 5 minutes
ffuf -u http://target.com/FUZZ -w large-wordlist.txt -maxtime 300

# Per-job timeout (useful with recursion)
ffuf -u http://target.com/FUZZ -w wordlist.txt -recursion -maxtime-job 60

# Quick scan with time limit
ffuf -u http://target.com/FUZZ -w wordlist.txt -maxtime 120 -t 100
```

5.4 WAF and IDS Evasion

5.4.1 Mimic Legitimate Browser

```
# Full browser header set
ffuf -u http://target.com/FUZZ \
     -w wordlist.txt \
     -H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36" \
     -H "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8" \
     -H "Accept-Language: en-US,en;q=0.9" \
     -H "Accept-Encoding: gzip, deflate" \
     -H "DNT: 1" \
     -H "Connection: keep-alive" \
     -H "Upgrade-Insecure-Requests: 1" \
     -H "Sec-Fetch-Dest: document" \
     -H "Sec-Fetch-Mode: navigate" \
     -H "Sec-Fetch-Site: none" \
     -H "Cache-Control: max-age=0" \
     -p 0.5-1.5 \
     -t 10
```

5.4.2 Bypass IP Restrictions

```
# Add localhost headers
ffuf -u http://target.com/FUZZ \
     -w wordlist.txt \
     -H "X-Forwarded-For: 127.0.0.1" \
```

```
      -H "X-Originating-IP: 127.0.0.1" \
      -H "X-Remote-IP: 127.0.0.1" \
      -H "X-Remote-Addr: 127.0.0.1" \
      -H "X-Client-IP: 127.0.0.1"

# Try different internal IPs
ffuf -u http://target.com/admin \
      -w internal-ips.txt \
      -H "X-Forwarded-For: FUZZ"
```

5.4.3 Stealth Configuration

```
# Maximum stealth approach
ffuf -u http://target.com/FUZZ \
      -w small-wordlist.txt \
      -t 5 \
      -p 2-5 \
      -rate 3 \
      -H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit
    /537.36" \
      -timeout 30
```

5.5 Following Redirects

```
# Follow redirects (default: false)
ffuf -u http://target.com/FUZZ -w wordlist.txt -r

# Don't follow redirects (see the 301/302 directly)
ffuf -u http://target.com/FUZZ -w wordlist.txt

# Match redirect codes
ffuf -u http://target.com/FUZZ -w wordlist.txt -mc 301,302,307 -r
```

Note

By default, FFUF does NOT follow redirects. Use `-r` to enable redirect following if you want to see the final destination.

5.6 Proxy Integration

5.6.1 Burp Suite Integration

```
# Send all traffic through Burp proxy
ffuf -u http://target.com/FUZZ -w wordlist.txt -x http://127.0.0.1:8080

# Replay proxy (only successful results)
ffuf -u http://target.com/FUZZ -w wordlist.txt \
      -replay-proxy http://127.0.0.1:8080

# Use with filter to send matches to Burp
ffuf -u http://target.com/FUZZ -w wordlist.txt \
```

```
    -mc 200 \
    -replay-proxy http://127.0.0.1:8080
```

5.6.2 SOCKS Proxy

```
# Use SOCKS5 proxy (Tor, etc.)
ffuf -u http://target.com/FUZZ -w wordlist.txt \
    -x socks5://127.0.0.1:9050
```

5.7 HTTP/2 Support

```
# Enable HTTP/2
ffuf -u https://target.com/FUZZ -w wordlist.txt -http2

# Some sites require HTTP/2 for certain endpoints
ffuf -u https://target.com/api/FUZZ -w endpoints.txt -http2
```

5.8 Ignore Response Body

```
# Don't fetch response body (faster, but less info)
ffuf -u http://target.com/FUZZ -w large-wordlist.txt -ignore-body

# Useful for quick status code checks
ffuf -u http://target.com/FUZZ -w wordlist.txt -ignore-body -mc 200
```

5.9 Stop Conditions

```
# Stop after finding N results
ffuf -u http://target.com/FUZZ -w wordlist.txt -maxtime 300 -t 50

# Automatic stop when pattern detected
ffuf -u http://target.com/FUZZ -w wordlist.txt -sa

# Stop on spurious errors
ffuf -u http://target.com/FUZZ -w wordlist.txt -se
```

Pro Tip

For large wordlists against slow servers:

- Start with `-t 20 -rate 30`
- Add `-timeout 20`
- Use `-maxtime 600` (10 minutes)
- Enable `-se` to handle errors gracefully

6 Output and Reporting

6.1 Output Formats

6.1.1 Supported Formats

```
# JSON output (default)
ffuf -u http://target.com/FUZZ -w wordlist.txt -o results.json

# Specify format explicitly
ffuf -u http://target.com/FUZZ -w wordlist.txt -o results.json -of json

# HTML report
ffuf -u http://target.com/FUZZ -w wordlist.txt -o report.html -of html

# CSV format
ffuf -u http://target.com/FUZZ -w wordlist.txt -o results.csv -of csv

# Markdown format
ffuf -u http://target.com/FUZZ -w wordlist.txt -o results.md -of md

# EJSON (Compact JSON)
ffuf -u http://target.com/FUZZ -w wordlist.txt -o results.ejson -of ejson

# All formats at once
ffuf -u http://target.com/FUZZ -w wordlist.txt -o results -of all
```

6.2 Output Directory

```
# Save matched responses to directory
ffuf -u http://target.com/FUZZ -w wordlist.txt -od output-dir/

# Combined with output file
ffuf -u http://target.com/FUZZ -w wordlist.txt \
     -o results.json \
     -od response-bodies/
```

Note

The `-od` flag saves the actual response content for each match to separate files. Useful for later analysis of discovered pages.

6.3 Verbose Output

```
# Verbose mode (show all requests and responses)
ffuf -u http://target.com/FUZZ -w wordlist.txt -v

# Colored output
ffuf -u http://target.com/FUZZ -w wordlist.txt -c

# Both verbose and colored
ffuf -u http://target.com/FUZZ -w wordlist.txt -v -c
```

6.4 Silent Mode

```
# Minimal output (only matches)
ffuf -u http://target.com/FUZZ -w wordlist.txt -s

# Silent with output file
ffuf -u http://target.com/FUZZ -w wordlist.txt -s -o results.json
```

6.5 Debug Logging

```
# Enable debug log
ffuf -u http://target.com/FUZZ -w wordlist.txt -debug-log debug.log

# Review debug information
cat debug.log
```

6.6 Parsing Results with jq

6.6.1 Extract URLs from JSON

```
# Get all discovered URLs
cat results.json | jq -r '.results[].url'

# Get URLs with status code 200
cat results.json | jq -r '.results[] | select(.status==200) | .url'

# Get URLs and status codes
cat results.json | jq -r '.results[] | "\(.url) - \(.status)"'

# Count results by status code
cat results.json | jq '.results | group_by(.status) |
    map({status: .[0].status, count: length})'
```

6.6.2 Filter by Response Size

```
# Responses larger than 5000 bytes
cat results.json | jq -r '.results[] | select(.length > 5000) | .url'

# Responses between 1000-5000 bytes
cat results.json | jq -r '.results[] |
    select(.length > 1000 and .length < 5000) | .url'
```

6.6.3 Extract for Further Testing

```
# Create URL list for Nuclei
cat results.json | jq -r '.results[].url' > discovered-urls.txt
nuclei -l discovered-urls.txt -t nuclei-templates/

# Create URL list for other tools
```

```
cat results.json | jq -r '.results[] | select(.status==200) | .url' >
    targets.txt
```

6.7 Integration Examples

6.7.1 FFUF + Nuclei Pipeline

```
# Discover endpoints and scan with Nuclei
ffuf -u http://target.com/FUZZ \
     -w /usr/share/seclists/Discovery/Web-Content/raft-large-files.txt \
     -mc 200 \
     -o results.json \
     -of json \
     -c

# Extract URLs and scan
cat results.json | jq -r '.results[].url' > urls.txt
nuclei -l urls.txt -t nuclei-templates/
```

6.7.2 FFUF + Burp Suite Workflow

```
# Send matches to Burp for manual testing
ffuf -u http://target.com/FUZZ \
     -w wordlist.txt \
     -mc 200,403 \
     -replay-proxy http://127.0.0.1:8080 \
     -o results.json
```

6.7.3 FFUF + GoWitness (Screenshots)

```
# Discover URLs and take screenshots
ffuf -u http://target.com/FUZZ \
     -w wordlist.txt \
     -mc 200 \
     -o results.json

cat results.json | jq -r '.results[].url' > urls.txt
gowitness file -f urls.txt
```

6.8 Conditional Output

```
# Only create output file if results found
ffuf -u http://target.com/FUZZ -w wordlist.txt -o results.json -or

# Without -or, empty results still create file
ffuf -u http://target.com/FUZZ -w wordlist.txt -o results.json
```

6.9 Result Statistics

After fuzzing, FFUF displays helpful statistics:

Example

```
________________________________________________

 :: Method           : GET
 :: URL              : http://target.com/FUZZ
 :: Wordlist         : FUZZ: wordlist.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403,405
________________________________________________

admin                   [Status: 200, Size: 4523, Words: 234, Lines: 89]
backup                  [Status: 200, Size: 1337, Words: 67, Lines: 23]
config                  [Status: 403, Size: 278, Words: 16, Lines: 12]

:: Progress: [4614/4614] :: Job [1/1] :: 250 req/sec :: Duration: [0:00:18]
   :: Errors: 0 ::
```

7 Practical Examples & Use Cases

7.1 Web Application Testing

7.1.1 Complete Web App Scan

```
# Phase 1: Directory discovery
ffuf -u https://target.com/FUZZ \
     -w /usr/share/seclists/Discovery/Web-Content/raft-large-directories.txt \
     -mc 200,301,302,403 \
     -o phase1-dirs.json \
     -c -v

# Phase 2: File discovery with extensions
ffuf -u https://target.com/FUZZ \
     -w /usr/share/seclists/Discovery/Web-Content/raft-large-files.txt \
     -e .php,.html,.txt,.js,.xml,.json,.bak,.old \
     -mc 200 \
     -o phase2-files.json \
     -c -v

# Phase 3: Backup file hunting
ffuf -u https://target.com/FUZZ \
     -w backup-names.txt \
     -e .bak,.backup,.old,.tmp,.swp,.zip,.tar.gz \
     -mc 200 \
     -o phase3-backups.json \
     -c
```

7.1.2 API Endpoint Discovery

```
# Discover API endpoints
ffuf -u https://api.target.com/v1/FUZZ \
     -w /usr/share/seclists/Discovery/Web-Content/api/api-endpoints.txt \
     -mc 200,201,204,400,401,403 \
     -H "Content-Type: application/json" \
     -o api-endpoints.json \
     -c -v

# Test API versions
ffuf -u https://api.target.com/FUZZ/users \
     -w <(echo -e "v1\nv2\nv3\napi/v1\napi/v2") \
     -mc 200,401 \
     -c
```

7.2 Authentication Testing

7.2.1 Login Brute Force (Username)

```
# Test common usernames
ffuf -u https://target.com/login \
```

```
      -X POST \
      -d "username=FUZZ&password=password123" \
      -H "Content-Type: application/x-www-form-urlencoded" \
      -w /usr/share/seclists/Usernames/top-usernames-shortlist.txt \
      -fr "Invalid username" \
      -mc 200,302 \
      -c
```

7.2.2 Password Spraying

```
# Try common passwords against known users
ffuf -u https://target.com/api/login \
      -X POST \
      -d '{"username":"admin","password":"FUZZ"}' \
      -H "Content-Type: application/json" \
      -w /usr/share/seclists/Passwords/Common-Credentials/10-million-password-list-top-1000.txt \
      -fr "Invalid credentials" \
      -rate 5 \
      -p 1-2 \
      -c
```

Warning

Always use rate limiting and delays for authentication testing to avoid:

- Account lockouts
- IP bans
- Detection by security systems

Use `-rate 5 -p 1-2` for safety.

7.3 Vulnerability Fuzzing

7.3.1 SQL Injection Testing

```
# Test for SQL injection
ffuf -u 'http://target.com/page.php?id=FUZZ' \
      -w /usr/share/seclists/Fuzzing/SQLi/Generic-SQLi.txt \
      -mc 200,500 \
      -fr "error|mysql|syntax|database" \
      -c -v

# Time-based SQLi detection
ffuf -u 'http://target.com/user?id=FUZZ' \
      -w sqli-time-based.txt \
      -ft 5000 \
      -c
```

7.3.2 XSS Testing

```
# Test for reflected XSS
ffuf -u 'http://target.com/search?q=FUZZ' \
     -w /usr/share/seclists/Fuzzing/XSS/XSS-Jhaddix.txt \
     -mr "<script>|alert|onerror" \
     -mc 200 \
     -c -v

# POST XSS testing
ffuf -u http://target.com/comment \
     -X POST \
     -d "comment=FUZZ&name=test" \
     -w xss-payloads.txt \
     -H "Content-Type: application/x-www-form-urlencoded" \
     -mc 200 \
     -c
```

7.3.3 Command Injection

```
# Test for command injection
ffuf -u 'http://target.com/ping.php?host=FUZZ' \
     -w /usr/share/seclists/Fuzzing/command-injection-commix.txt \
     -mc 200 \
     -mr "root:|uid=|Directory" \
     -c -v
```

7.3.4 Local File Inclusion (LFI)

```
# Test for LFI
ffuf -u 'http://target.com/page.php?file=FUZZ' \
     -w /usr/share/seclists/Fuzzing/LFI/LFI-Jhaddix.txt \
     -mc 200 \
     -mr "root:x:|bin/bash" \
     -c -v

# Deep LFI with path traversal
for i in {1..10}; do
    prefix=$(printf '../%.0s' $(seq 1 $i))
    echo "${prefix}etc/passwd"
done | ffuf -u 'http://target.com/?page=FUZZ' -w - -mc 200 -c
```

7.4 HackTheBox / CTF Scenarios

7.4.1 Virtual Host Discovery (Common in HTB)

```
# Example: board.htb
ffuf -w /usr/share/seclists/Discovery/DNS/bitquark-subdomains-top100000.
    txt:FUZZ \
     -u http://board.htb/ \
     -H 'Host: FUZZ.board.htb' \
```

```
    -fs 15949 \
    -c

# Common HTB pattern: test for dev/admin/api subdomains
echo -e "dev\nadmin\napi\ntest\nstaging\ncrm\nmail" | \
ffuf -u http://target.htb/ -H 'Host: FUZZ.target.htb' -w - -c
```

7.4.2 Hidden Parameter Discovery

```
# Find hidden parameters
ffuf -u 'http://target.htb/admin.php?FUZZ=test' \
    -w /usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt
    \
    -fs 1234 \
    -c -v

# Test numeric IDs
ffuf -u 'http://target.htb/user.php?id=FUZZ' \
    -w <(seq 1 1000) \
    -mc 200 \
    -fw 10 \
    -c
```

7.4.3 Backup and Config Files

```
# Common CTF hidden files
echo -e ".git\n.env\n.htaccess\nconfig.php\nbackup.zip\ndb.sql" | \
ffuf -u http://target.htb/FUZZ -w - -mc 200,403 -c

# Comprehensive backup search
ffuf -u http://target.htb/FUZZ \
    -w backup-files.txt \
    -e .bak,.old,.backup,.swp,.save,.tmp \
    -mc 200 \
    -c
```

7.5 Advanced Real-World Scenarios

7.5.1 Multi-Level Path Fuzzing

```
# Fuzz multiple path levels
ffuf -u http://target.com/FUZZ/FUZ2Z \
    -w dirs.txt:FUZZ \
    -w files.txt:FUZ2Z \
    -mode clusterbomb \
    -mc 200 \
    -c

# Example: /admin/backup or /admin/config
ffuf -u http://target.com/FUZZ/FUZ2Z \
    -w <(echo -e "admin\nuser\napi") \
```

```
      -w <(echo -e "backup\nconfig\ntest") \
      -mode clusterbomb \
      -mc 200 \
      -c
```

7.5.2 Custom Wordlist from Site

```
# Generate wordlist from target site
cewl http://target.com -d 2 -m 5 -w custom-wordlist.txt

# Use custom wordlist
ffuf -u http://target.com/FUZZ -w custom-wordlist.txt -mc 200 -c

# Combine with common extensions
ffuf -u http://target.com/FUZZ \
     -w custom-wordlist.txt \
     -e .php,.html,.txt \
     -mc 200 \
     -c
```

7.5.3 API Fuzzing with Different Methods

```
# Test multiple HTTP methods on discovered endpoints
for method in GET POST PUT DELETE PATCH OPTIONS; do
    echo "Testing $method method..."
    ffuf -u http://api.target.com/v1/FUZZ \
         -w api-endpoints.txt \
         -X $method \
         -H "Authorization: Bearer token123" \
         -mc 200,201,204,400,401,403,405 \
         -c
done
```

8 Visual Analysis & Workflows

8.1 FFUF Fuzzing Workflow

8.2 Finding Severity Distribution

Note

This pie chart represents a typical distribution of findings from a comprehensive FFUF scan:

- **Critical (15%):** SQL injection points, RCE endpoints, exposed admin panels
- **High (25%):** Authentication bypasses, sensitive file exposure, XSS vulnerabilities
- **Medium (35%):** Information disclosure, missing security headers, backup files
- **Low (25%):** Directory listings, verbose errors, minor misconfigurations

8.3 Vulnerability Severity by Fuzzing Type

8.4 FFUF Decision Tree

8.5 Response Time Analysis

Pro Tip

Sudden spikes in response time (especially around 5000ms) often indicate:

- Time-based SQL injection vulnerabilities
- Server-side processing delays
- Potential code execution points

Use `ffuf -ft 5000` to filter and highlight these suspicious responses.

9 Quick Reference

9.1 Essential Flags Cheat Sheet

Flag	Description
`-u`	Target URL with FUZZ keyword
`-w`	Wordlist file path
`-H`	Custom header
`-X`	HTTP method (GET, POST, PUT, DELETE)
`-d`	POST data
`-mc`	Match HTTP status codes
`-fc`	Filter HTTP status codes
`-fs`	Filter response size
`-fw`	Filter word count
`-fl`	Filter line count
`-fr`	Filter regex pattern
`-mr`	Match regex pattern
`-t`	Number of threads (default: 40)
`-rate`	Rate limit (requests per second)
`-p`	Delay between requests (seconds)
`-timeout`	HTTP request timeout (default: 10)
`-r`	Follow redirects
`-c`	Colorize output
`-v`	Verbose output
`-s`	Silent mode
`-o`	Output file
`-of`	Output format (json, html, md, csv, all)
`-od`	Output directory for responses
`-e`	File extensions to append
`-recursion`	Enable recursive fuzzing
`-recursion-depth`	Maximum recursion depth
`-mode`	Multi-wordlist mode (clusterbomb, pitchfork, sniper)
`-ac`	Auto-calibration
`-x`	Proxy URL
`-replay-proxy`	Replay proxy (send matches to proxy)
`-http2`	Use HTTP/2 protocol

Table 1: FFUF Essential Flags Reference

9.2 Common Command Templates

9.2.1 Template: Basic Directory Scan

Directory Discovery

```
ffuf -u http://TARGET/FUZZ -w WORDLIST -mc 200,301,302,403 -c
```

9.2.2 Template: Subdomain Enumeration

Subdomain/VHost Discovery

```
ffuf -u http://TARGET/ -H "Host:  FUZZ.TARGET" -w WORDLIST -fs SIZE -c
```

9.2.3 Template: API Fuzzing

API Endpoint Discovery

```
ffuf -u http://api.TARGET/v1/FUZZ -w WORDLIST -mc 200,401,403 -c -v
```

9.2.4 Template: POST Authentication

Login Brute Force

```
ffuf -u http://TARGET/login -X POST -d "user=admin&pass=FUZZ"
-w WORDLIST -H "Content-Type:  application/x-www-form-urlencoded" -fc 401
```

9.2.5 Template: Parameter Discovery

GET Parameter Fuzzing

```
ffuf -u 'http://TARGET/page.php?FUZZ=test' -w WORDLIST -fs SIZE -c
```

9.3 Recommended Wordlists

Use Case	Wordlist Path
Directories	/usr/share/seclists/Discovery/Web-Content/raft-large-directories.txt
Files	/usr/share/seclists/Discovery/Web-Content/raft-large-files.txt
Subdomains	/usr/share/seclists/Discovery/DNS/subdomains-top1million-110000.txt
Parameters	/usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt
API Endpoints	/usr/share/seclists/Discovery/Web-Content/api/api-endpoints.txt
Usernames	/usr/share/seclists/Usernames/Names/names.txt
Passwords	/usr/share/seclists/Passwords/Common-Credentials/10-million-password-list-top-1
SQLi Payloads	/usr/share/seclists/Fuzzing/SQLi/Generic-SQLi.txt
XSS Payloads	/usr/share/seclists/Fuzzing/XSS/XSS-Jhaddix.txt
LFI Payloads	/usr/share/seclists/Fuzzing/LFI/LFI-Jhaddix.txt
Common Files	/usr/share/wordlists/dirb/common.txt

Table 2: Recommended Wordlists by Use Case

Code	Meaning	Significance
200	OK	Resource found and accessible
201	Created	Resource successfully created (APIs)
204	No Content	Success with no response body
301	Moved Permanently	Redirect to new location
302	Found	Temporary redirect
307	Temporary Redirect	Maintains HTTP method
400	Bad Request	Invalid request format
401	Unauthorized	Authentication required
403	Forbidden	Access denied (but exists!)
404	Not Found	Resource doesn't exist
405	Method Not Allowed	Valid path, wrong method
500	Internal Server Error	Server-side error (potential vuln)
502	Bad Gateway	Proxy/gateway error
503	Service Unavailable	Server temporarily down

Table 3: HTTP Status Codes for Fuzzing

9.4 Status Code Reference

Pro Tip

Key codes to match in FFUF:

- **Always include:** 200, 301, 302, 403
- **403 is important:** Often indicates existing paths protected by ACLs
- **401 for auth:** Reveals authentication endpoints
- **500 for testing:** May indicate injection vulnerabilities

9.5 Performance Quick Guide

Scenario	Settings	Command
Maximum Speed	100+ threads	`-t 100`
Balanced	40-50 threads	`-t 40` (default)
Stealth	5-10 threads, delays	`-t 5 -p 2-5 -rate 3`
Slow Server	Low threads, timeout	`-t 10 -timeout 30`
WAF Evasion	Rate limit, delays	`-t 10 -rate 10 -p 0.5-1.5`
Large Wordlist	Time limit	`-maxtime 600 -t 50`

Table 4: Performance Settings by Scenario

10 Troubleshooting & Best Practices

10.1 Common Issues and Solutions

10.1.1 Issue: Too Many False Positives

Warning

Problem: FFUF returns hundreds or thousands of matches, most irrelevant.

Solutions:

1. Identify the common error page size and filter it:

```
# First run to see response sizes
ffuf -u http://target.com/FUZZ -w wordlist.txt -c -v

# Filter the common error size (e.g., 4242 bytes)
ffuf -u http://target.com/FUZZ -w wordlist.txt -fs 4242 -c
```

2. Use auto-calibration:

```
ffuf -u http://target.com/FUZZ -w wordlist.txt -ac -c
```

3. Filter by word count if sizes vary:

```
ffuf -u http://target.com/FUZZ -w wordlist.txt -fw 33 -c
```

10.1.2 Issue: Connection Errors or Timeouts

Warning

Problem: Many requests failing with connection errors.

Solutions:

1. Reduce thread count:

```
ffuf -u http://target.com/FUZZ -w wordlist.txt -t 10
```

2. Increase timeout:

```
ffuf -u http://target.com/FUZZ -w wordlist.txt -timeout 30
```

3. Add delays:

```
ffuf -u http://target.com/FUZZ -w wordlist.txt -p 0.5 -t 20
```

4. Enable stop on errors:

```
ffuf -u http://target.com/FUZZ -w wordlist.txt -se
```

10.1.3 Issue: Getting Blocked or Banned

Warning

Problem: Target is blocking your IP or requests.

Solutions:

1. Implement stealth settings:

```
ffuf -u http://target.com/FUZZ \
     -w wordlist.txt \
     -t 5 \
     -rate 5 \
     -p 1-3 \
     -H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)" \
     -c
```

2. Use a proxy or VPN:

```
ffuf -u http://target.com/FUZZ -w wordlist.txt -x http://proxy:8080
```

3. Rotate User-Agents:

```
# Create user-agent list
cat > user-agents.txt << EOF
Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
Mozilla/5.0 (X11; Linux x86_64)
EOF

# Use with FFUF (requires script wrapper)
while read ua; do
    ffuf -u http://target.com/FUZZ -w wordlist.txt -H "User-Agent: $ua" -c
done < user-agents.txt
```

10.1.4 Issue: No Results Found

Warning

Problem: FFUF completes but finds nothing.

Solutions:

1. Verify target is accessible:

```
curl -I http://target.com
```

2. Test with known existing path:

```
# Create test wordlist with known paths
echo -e "index.html\nrobots.txt\nfavicon.ico" > test.txt
ffuf -u http://target.com/FUZZ -w test.txt -c -v
```

3. Use match-all and examine responses:

```
ffuf -u http://target.com/FUZZ -w wordlist.txt -mc all -c -v
```

4. Try different wordlists:

```
ffuf -u http://target.com/FUZZ -w /usr/share/wordlists/dirb/common.txt
     -c
```

10.2 Best Practices

10.2.1 1. Start Small, Scale Up

- Begin with small wordlists (e.g., common.txt with 4,000 words)
- Test with low thread counts first
- Observe response patterns before scaling
- Progressively increase wordlist size and threads

Success/Tip

```
# Step 1: Quick test
ffuf -u http://target.com/FUZZ -w /usr/share/wordlists/dirb/common.txt
    -c

# Step 2: If successful, scale up
ffuf -u http://target.com/FUZZ \
     -w /usr/share/seclists/Discovery/Web-Content/raft-large-files.txt
    \
     -t 50 -c
```

10.2.2 2. Always Use Filtering

- Never run FFUF without at least basic filtering
- Start with status code filtering: `-fc 404`
- Add size filtering after initial reconnaissance
- Combine multiple filters for precision

10.2.3 3. Save Your Results

- Always output to a file: `-o results.json`
- Use descriptive filenames with timestamps
- Save in multiple formats for different uses
- Keep response bodies for critical findings: `-od responses/`

Success/Tip

```
# Comprehensive output example
ffuf -u http://target.com/FUZZ \
     -w wordlist.txt \
     -o "scan-$(date +%Y%m%d-%H%M%S).json" \
     -of json \
     -od response-bodies/ \
     -c -v
```

10.2.4 4. Monitor Your Traffic

- Use `-replay-proxy` to send matches to Burp Suite
- Review interesting findings manually
- Validate automated results before reporting
- Document the fuzzing methodology

10.2.5 5. Respect Rate Limits

- Start with conservative settings: `-t 20 -rate 30`
- Add delays for sensitive targets: `-p 0.5-1.5`
- Monitor for 429 (Too Many Requests) responses
- Be a responsible pentester - don't DoS the target

10.2.6 6. Legal and Ethical Considerations

Warning

CRITICAL REMINDERS:

- **Always obtain written authorization** before testing
- **Stay within scope** - only test authorized targets
- **Respect rate limits** - don't cause service disruption
- **Follow responsible disclosure** for found vulnerabilities
- **Document everything** - methodology, findings, timestamps
- **Stop immediately** if you find critical issues

10.3 Pro Tips

Pro Tip

Tip 1: Use Multiple FUZZ Keywords
Instead of running separate scans, use FUZZ, FUZ2Z, FUZ3Z, etc.:

```
ffuf -u http://target.com/FUZZ/FUZ2Z -w dirs.txt:FUZZ -w files.txt:
    FUZ2Z
```

Pro Tip

Tip 2: Generate Dynamic Wordlists
Use shell commands to create wordlists on-the-fly:

```
# Date ranges
for y in 2023 2024; do
    for m in {01..12}; do
        echo "${y}-${m}"
    done
done | ffuf -u 'http://target.com/reports/FUZZ.pdf' -w -

# Numeric IDs
seq 1000 9999 | ffuf -u 'http://target.com/invoice?id=FUZZ' -w -
```

Pro Tip

Tip 3: Combine with Other Tools
FFUF works great in toolchains:

```
# FFUF -> jq -> httpx -> Nuclei
ffuf -u http://target.com/FUZZ -w wordlist.txt -mc 200 -o r.json && \
cat r.json | jq -r '.results[].url' | httpx -silent | \
nuclei -t cves/ -silent
```

Pro Tip

Tip 4: Context-Aware Fuzzing
Tailor your wordlist to the target technology:

- PHP sites: Focus on .php extensions
- ASP.NET: Use .aspx, .asmx extensions
- APIs: Use JSON endpoints, versioning
- CMS: Use CMS-specific paths (wp-admin, administrator)

11 Resources and Further Reading

11.1 Official Resources

- **GitHub Repository:** https://github.com/ffuf/ffuf
- **FFUF Wiki:** Comprehensive documentation with examples
- **FFUF.me:** Practice environment for learning FFUF

11.2 Essential Reading

- **"Everything you need to know about FFUF"** by Michael Skelton (@codingo)
- **SecLists Repository:** https://github.com/danielmiessler/SecLists
- **Assetnote Wordlists:** High-quality fuzzing wordlists

11.3 Practice Platforms

- **HackTheBox:** Real-world practice with FFUF techniques
- **TryHackMe:** Guided rooms for learning fuzzing
- **PentesterLab:** Structured exercises for web application testing
- **PortSwigger Web Security Academy:** Free labs for practice

11.4 Related Tools

- **Gobuster:** Alternative directory brute-forcer
- **Wfuzz:** Python-based web fuzzer
- **Feroxbuster:** Rust-based recursive fuzzer
- **Dirsearch:** Python directory scanner

Legal Disclaimer

This cheat sheet is intended for authorized security testing and educational purposes only.

Unauthorized access to computer systems is illegal.

Always obtain proper authorization before conducting any security assessments.

The author assumes no liability for misuse of this information.

Document Version: v2.0
Last Updated: November 2, 2025
Compiled with: LaTeX

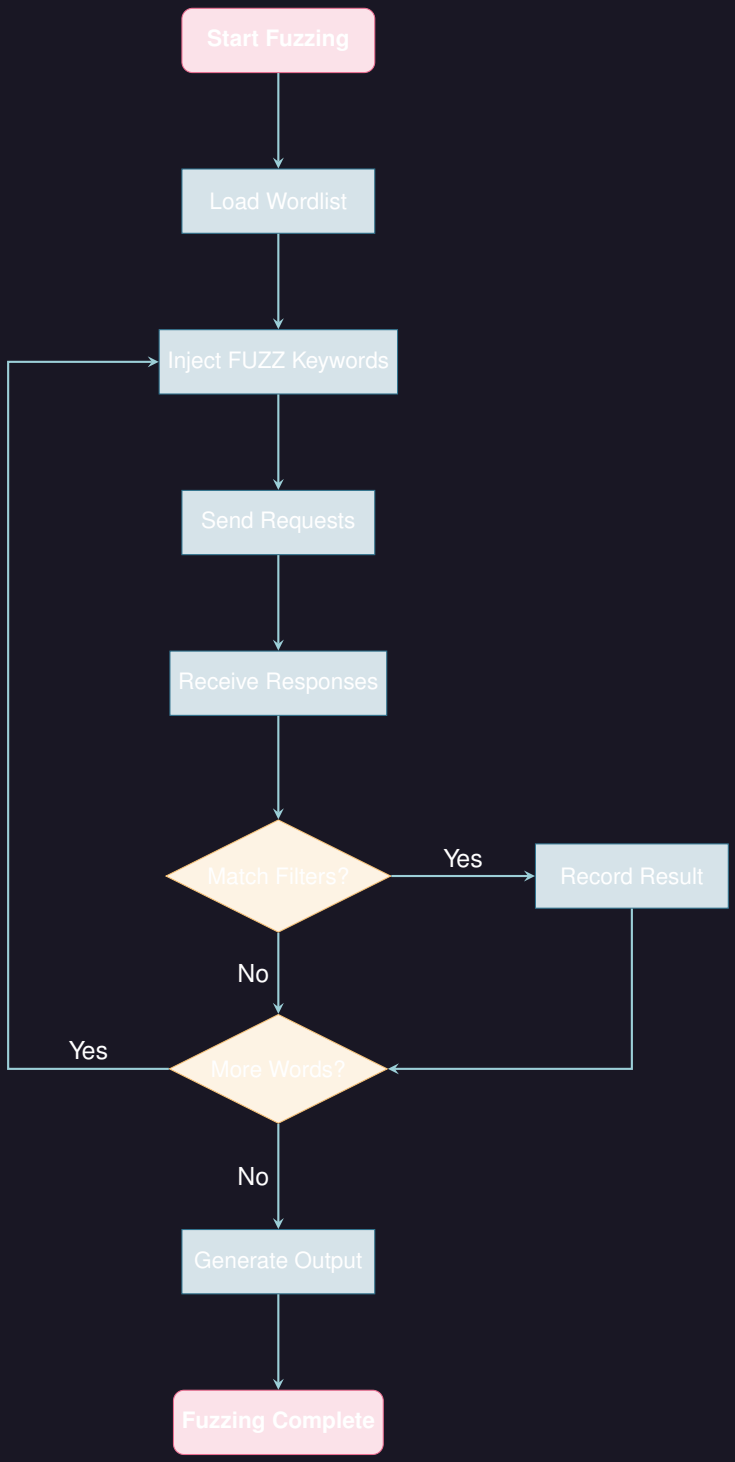


Figure 1: Basic FFUF Fuzzing Workflow

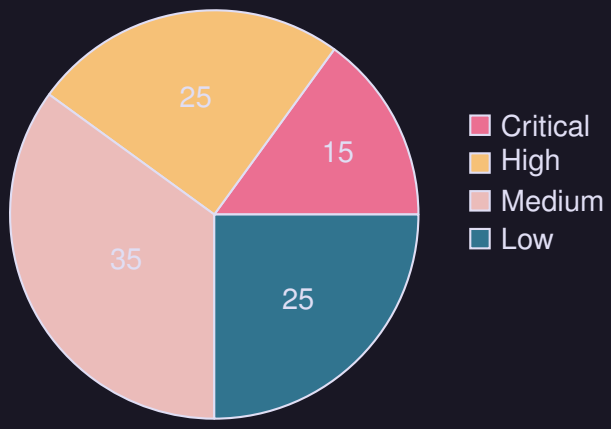


Figure 2: Example Vulnerability Distribution from FFUF Scan

Figure 3: Vulnerability Severity Distribution by Fuzzing Type

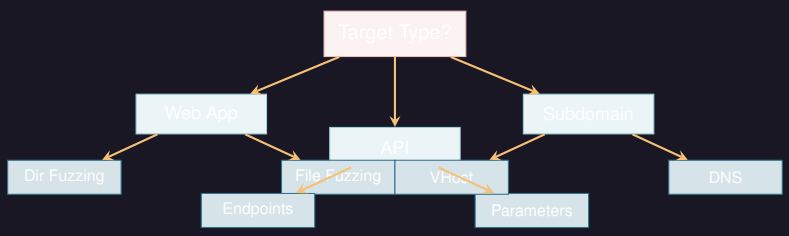


Figure 4: FFUF Target Selection Decision Tree

Potential SQLi!

Figure 5: Response Time Analysis - Detecting Time-Based SQL Injection