Arjun: The Ultimate Parameter Discovery Tool For Bug Hunters

IronGhost Follow 5 min read · Apr 24, 2025

Uncovering Hidden Secrets in Web Apps with Arjun

Why didn't I find this vulnerability earlier?

If you've ever felt this way during a security assessment, Arjun might become your new best friend. Let's break down how this open-source tool helps you discover hidden HTTP parameters that most scanners overlook.

What is Arjun?

Arjun is a Python tool that brute-forces **hidden HTTP parameters** (like `?user_id=123` in URLs) that aren't visible in forms, APIs, or documentation. These parameters are often left unprotected and can lead to critical vulnerabilities like SQL injection, IDOR, or data leaks.

When Should You Use Arjun?

1. **Bug Bounty Hunting**: Find obscure parameters in targets for unique vulnerabilities.
2. **Pentesting**: Uncover hidden API endpoints or misconfigured parameters.
3. **API Testing**: Discover undocumented parameters in REST/SOAP APIs.
4. **Web App Recon**: Map out all possible inputs for a URL before deeper testing.

Example Scenario:
A site uses `https://example.com/profile?uid=123`. Arjun might find `admin_mode=true`, exposing an access control flaw.

Installation

```
pipx install arjun

Or follow the official github instructios. link given at the bottom.
```

Help Documentation

```
arjun -h
usage: arjun [-h] [-u URL] [-o JSON_FILE] [-oT TEXT_FILE] [-oB [BURP_PROXY]] [-d
             [-T TIMEOUT] [-c CHUNKS] [-q] [--rate-limit RATE_LIMIT] [--headers
             [--disable-redirects] [--casing CASING]

options:
  -h, --help            show this help message and exit
  -u URL                Target URL
  -o, -oJ JSON_FILE     Path for json output file.
  -oT TEXT_FILE         Path for text output file.
  -oB [BURP_PROXY]      Output to Burp Suite Proxy. Default is 127.0.0.1:8080.
```

```
  -d DELAY              Delay between requests in seconds. (default: 0)
  -t THREADS            Number of concurrent threads. (default: 5)
  -w WORDLIST           Wordlist file path. (default: {arjundir}/db/large.txt)
  -m METHOD             Request method to use: GET/POST/XML/JSON. (default: GET)
  -i [IMPORT_FILE]      Import target URLs from file.
  -T TIMEOUT            HTTP request timeout in seconds. (default: 15)
  -c CHUNKS             Chunk size. The number of parameters to be sent at once
  -q                    Quiet mode. No output.
  --rate-limit RATE_LIMIT
                        Max number of requests to be sent out per second (defaul
  --headers [HEADERS]   Add headers. Separate multiple headers with a new line.
  --passive [PASSIVE]   Collect parameter names from passive sources like waybac
  --stable              Prefer stability over speed.
  --include INCLUDE     Include this data in every request.
  --disable-redirects   disable redirects
  --casing CASING       casing style for params e.g. like_this, likeThis, liketh
```

Find Hidden Parameters

1. Basic Scanning & Output

`-u` : Scan a single URL for hidden parameters.

Use case: Quick recon on a suspicious endpoint.

```
arjun -u https://example.com/api/v1/user
```

`-oJ` : Save results to JSON for automation.

Use case: Export parameters to feed into `sqlmap` or `nuclei`.

```
arjun -u https://example.com/login -oJ params.json
```

`-oT` : Save results in human-readable text.
Use case: Share findings with non-technical teams.

```
arjun -u https://example.com/profile -oT params.txt
```

2. Stealth & Performance Tuning

`-d` : Add delays between requests.
Use case: Avoid triggering rate limits/WAFs during a bug bounty hunt.

```
arjun -u https://example.com -d 2  # 2-second delay
```

`-t` : Increase threads for speed.
Use case: Scan internal test environments faster.

```
arjun -u http://internal-app:8080 -t 10  # 10 threads
```

`--rate-limit` : Cap requests per second.
Use case: Stay under the radar in production systems.

```
arjun -u https://example.com --rate-limit 5  # 5 requests/sec
```

`--stable` : Prioritize reliability over speed.
Use case: Scan unstable/legacy servers prone to crashing.

```
arjun -u https://fragile-old-app.com --stable
```

3. Advanced Parameter Discovery

`-w` : Use a custom wordlist.
Use case: Target niche frameworks (e.g., Salesforce, SAP).

```
arjun -u https://example.com -w salesforce_params.txt
```

`-c` : Send parameters in chunks.
Use case: Test APIs that accept multiple parameters at once.

```
arjun -u https://api.example.com -c 5  # 5 params per request
```

`--casing` : Match parameter naming conventions.
Use case: Find `camelCase` params in JavaScript-heavy apps.

```
arjun -u https://react-app.example.com --casing camel
```

4. Complex Workflows

`-m` : Test POST endpoints/APIs.
Use case: Discover hidden form fields or API parameters.

```
arjun -u https://example.com/login -m POST
```

`--headers` : Add auth tokens or cookies.
Use case: Scan authenticated endpoints.

```
arjun -u https://example.com/dashboard --headers "Cookie: session=123"
```

`--include` : Inject static data (e.g., API keys).
Use case: Test endpoints requiring mandatory fields.

```
arjun -u https://api.example.com --include "token=xyz" -m POST
```

5. Recon & Automation

`-i` : Bulk-scan URLs from a file.
Use case: Test all endpoints from a sitemap.

```
arjun -i urls.txt
```

`--passive` : Find parameters via Wayback Machine/CommonCrawl.
Use case: Pre-scanning without alerting the target.

```
arjun --passive example.com  # Uses historical data only
```

`-oB` : Send results directly to Burp Suite.
Use case: Manually verify findings in Burp.

```
arjun -u https://example.com -oB  # Proxy to Burp
```

6. Edge Cases & Troubleshooting

`--disable-redirects` : Handle redirect loops.
Use case: Scan endpoints that 302-redirect by default.

```
arjun -u https://example.com/old-page --disable-redirects
```

`-T` : Increase timeout for slow servers.
Use case: Scan overloaded APIs with delayed responses.

```
arjun -u https://slow-api.example.com -T 30  # 30-second timeout
```

`-q` : Quiet mode for scripting/automation.
Use case: Integrate Arjun into CI/CD pipelines.

```
arjun -u https://example.com -q -oJ params.json
```

Combined Use Cases

1. Auth + POST + Wordlist + Proxy

Scenario: Test an authenticated API endpoint with a custom wordlist and inspect traffic in Burp.

```
arjun -u https://api.example.com/update \
  -m POST \
  --headers "Authorization: Bearer xyz" \
  -w api_params.txt \
  -oB
```

2. Passive + Casing + JSON Output

Scenario: Gather historical parameters for a React app and save results.

```
arjun --passive example.com \
  --casing camel \
  -oJ passive_params.json
```

3. Bulk Scan with Delays

Scenario: Safely scan 100 URLs from a file with rate limiting.

```
arjun -i urls.txt \
  -d 1 \
  --rate-limit 3 \
  -oT all_params.txt
```

Pro Tip

Combine `--include` with `-m JSON` to test APIs expecting nested data:

```
arjun -u https://api.example.com \
  -m JSON \
  --include '{"user": "test"}'
```

Real-World Use Cases

1. **IDOR (Insecure Direct Object Reference):**
   Arjun finds `?account_id=456`. Changing this value lets you access other users' data.
2. **Debug Parameters:**
   Discover `?debug=true` exposing stack traces or internal data.
3. **API Parameter Abuse:**
   Find `?limit=1000` to bypass pagination and scrape data.
4. **SQL Injection:**
   Uncover a forgotten `?search=` parameter vulnerable to SQLi.

Conclusion

Arjun turns the needle-in-a-haystack problem of parameter discovery into a streamlined process. Whether you're securing your own app or hunting for bugs, it's a must-have tool for uncovering hidden attack surfaces.

Final Command Cheatsheet:

```
# Basic scan
arjun -u https://example.com
```

```
# Save output and automate
arjun -u https://example.com -o params.json

# POST request with proxy
arjun -u https://example.com/login --m POST --oB http://localhost:8080
```

References

https://github.com/s0md3v/Arjun

Querystring Arjun Bug Bounty Hacking Fuzzing

Written by IronGhost

17 followers · 1 following

Follow

Responses (1)

Write a response

What are your thoughts?

Steiner254
May 20

Interesting!

Reply

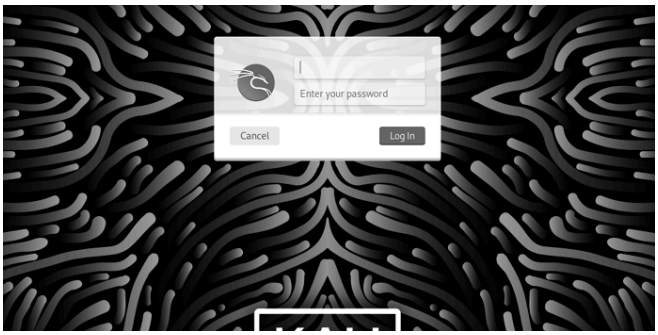
More from IronGhost

IronGhost

Top WordList for Hackers in 2025

Choosing the Right Wordlist in SecLists for Every Security Testing Scenario

Apr 22



IronGhost

GraphQL API hacking Series for Bug Hunters Part 02

Setting up a security lab for GraphQL testing

Apr 29 1 1

IronGhost

Mastering FFUF: The Web Fuzzing Tool For Hackers

FFUF (Fuzz Faster U Fool)

Apr 9

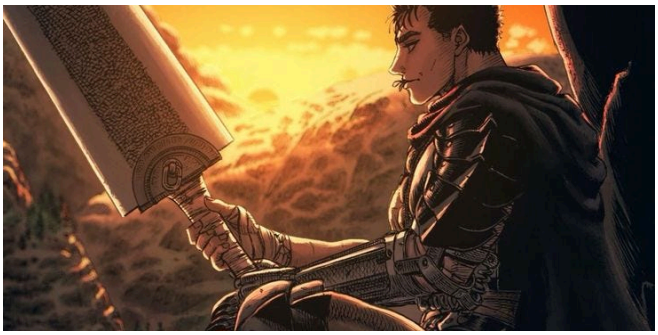
IronGhost

GraphQL API hacking Series for Bug Hunters Part 01

Understanding GraphQL and Its Security Implications

Apr 28 2 1

See all from IronGhost



Israel Aráoz Severiche

Hacking APIs: Enumeration and Recon Techniques for Modern APIs

Before exploiting APIs, attackers need to understand how they work. API...

Jun 10 20

AOX_TrojanPs

How I Found My First Bug (RXSS)

بسم الله و الصلاة و السلام على نبينا المجاهد الشهيد

Jan 7 418 2

In MeetCyber by AbhirupKonwar

DOM XSS Custom Nuclei Template

Find DOM XSS sources and sink using nuclei private template

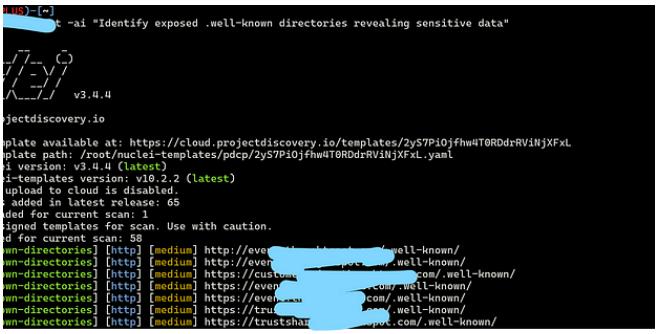
Apr 6 44

In System Weakness by Taahir Mujawarr

Recon Done, Now What? A Beginner's Guide to Finding Bugs...

By Taahir Mujawarr, Certified Ethical Hacker & Cyber Security Researcher

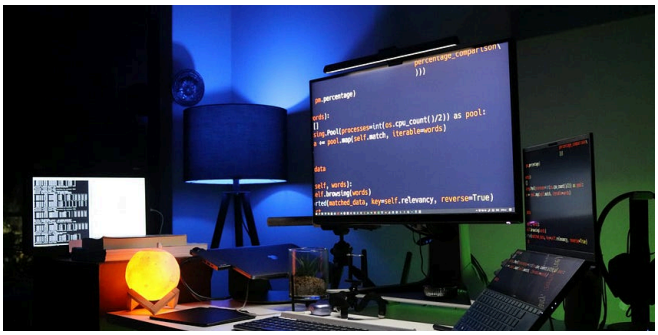
Mar 12 16



loyalonlytoday

Detect Vulnerabilities With Nuclei AI

Find vulnerabilities on your bug bounty target easily



Ibtissam hammadi

I Found a $4,200 Bug in 15 Minutes

Here's How It Happened

Jun 13 56

5d ago 115 8

See more recommendations