# Day 14: Brute Force Attacks

A brute-force attack is a method used to discover valid credentials by systematically attempting many possible combinations of usernames and passwords. They can be classified broadly into online and offline attacks, each with different implications, techniques, and countermeasures.

## Online vs Offline Brute Force Attacks

|  |  |  |
| --- | --- | --- |
| Aspect | Online Brute Force | Offline Brute Force |
| Definition | Attempts are sent directly to the live target (e.g., SSH, HTTP login). Limited by server defenses like rate limiting and CAPTCHAs. | Performed on stolen/obtained password hashes without interacting with the target system. |
| Speed | Slower due to network latency and rate limits. | Faster since it is done locally, can use GPUs for acceleration. |
| Stealth | Easier to detect (network traffic, intrusion detection). | Harder to detect because it doesn't touch the live system. |
| Requirements | Only needs the target service to be reachable. | Requires obtaining a database or hash dump first. |
| Examples | Testing SSH login with many passwords. | Using hashcat to crack dumped bcrypt hashes. |

## Types of Brute Force Attacks

|  |  |
| --- | --- |
| Attack Type | Description |
| Simple Brute Force | Tries all possible combinations of characters until the correct password is found. Guarantees success but is time-consuming for strong passwords. |
| Dictionary Attack | Uses a list of common or leaked passwords to guess credentials more efficiently. |
| Hybrid Attack | Combines dictionary attacks with variations such as adding numbers, symbols, or changing case. |
| Credential Stuffing | Uses known username:password pairs from previous breaches to try on other sites. |
| Password Spraying | Tries a few common passwords against many accounts to avoid account lockouts. |
| Rainbow Table Attack | Uses precomputed tables of hash-to-password mappings; ineffective if passwords are salted. |

## Role of CAPTCHA in Brute Force Prevention

CAPTCHAs introduce challenges that humans can solve but automated tools struggle with. They help slow down or block automated brute-force attacks. However, CAPTCHAs are not foolproof; some attackers use machine learning or third-party human solvers to bypass them.

## Tools Overview

Hydra: A fast and versatile brute-force tool supporting multiple protocols (SSH, FTP, HTTP, etc.).

Burp Suite & FoxyProxy: Used for intercepting and analyzing HTTP requests, identifying parameters, and crafting automated attack payloads in a controlled lab environment.

## Safe Lab Workflow (Authorized Testing Only)

1. Confirm testing is authorized.
2. Scan the target using Nmap to identify open ports and services.
3. Capture login requests with Burp Suite to understand parameters.
4. Select an attack method based on available information.
5. Run tests at low intensity to avoid DoS.
6. Verify found credentials through normal login methods.
7. Document all steps and findings.

## Defensive Measures

* Enforce MFA.
* Strong password policies with banned password lists.
* Rate limiting and progressive delays.
* Account lockouts after failed attempts.
* Implement CAPTCHAs.
* Monitor authentication patterns.
* Use strong password hashing algorithms with salts.

**How Burp helps for form-based testing (conceptual)**

* Intercept a legitimate login request to see the exact POST body and cookies.
* Identify which part of the response indicates “login failed” (error string or HTTP code), because automated tools use that to decide success/failure.
* If an anti-CSRF token is required and changes each request, automated testing must handle or reuse tokens appropriately — this often makes automated brute forcing harder.
* **How to verify an SSH credential (ethically)**
* On a system you’re allowed to access, the normal way to check a credential is to attempt an SSH connection with the username; the SSH client will prompt you for the password. If correct, you’ll get a shell. Don’t attempt this on systems without permission.
* **Detection & defensive measures (how to harden systems)**
* If your goal is to *defend* a service, use layered controls:
* **Enforce MFA** — single best mitigation against credential compromise.
* **Strong password policies** + encourage/passive checks (no plain complexity-only policies; use banned password lists).
* **Rate limiting** — throttle repeated attempts from the same IP or account.
* **Progressive delays & account lockouts** — after several failed attempts.
* **CAPTCHA** — as an additional friction for automated attacks.
* **IP reputation / geo checks** — block or flag suspicious origins.
* **Use strong password hashing** (Argon2 / bcrypt / scrypt) with per-password salt and high work factor.
* **Monitor and alert on anomalous auth patterns** (password spraying patterns, mass login attempts, credential stuffing indicators).
* **Use WAF and bot mitigation services** for web login protection.
* **Deploy logging and rotate logs to SIEM** — correlate failed attempts, spikes, and lockouts.
* **Ethics, legality, and best practice reminder**
* Always have **written authorization** before running any active attack on systems you don’t own. Even in pen-testing engagements you'll want a scoped rules-of-engagement.
* Use official learning platforms (TryHackMe, Hack The Box), intentionally vulnerable VMs (e.g., Metasploitable) or your own isolated VMs to practice.
* Document what you did and obtain permission for any tests that could impact availability or data.