Dictionary Attacks

While comprehensive, the brute-force approach can be time-consuming and resource-intensive, especially when dealing with complex passwords. That's where dictionary attacks come in.

The Power of Words

The effectiveness of a dictionary attack lies in its ability to exploit the human tendency to prioritize memorable passwords over secure ones. Despite repeated warnings, many individuals continue to opt for passwords based on readily available information such as dictionary words, common phrases, names, or easily guessable patterns. This predictability makes them vulnerable to dictionary attacks, where attackers systematically test a pre-defined list of potential passwords against the target system.

The success of a dictionary attack hinges on the quality and specificity of the wordlist used. A well-crafted wordlist tailored to the target audience or system can significantly increase the probability of a successful breach. For instance, if the target is a system frequented by gamers, a wordlist enriched with gaming-related terminology and jargon would prove more effective than a generic dictionary. The more closely the wordlist reflects the likely password choices of the target, the higher the chances of a successful attack.

At its core, the concept of a dictionary attack is rooted in understanding human psychology and common password practices. By leveraging this insight, attackers can efficiently crack passwords that might otherwise necessitate an impractically lengthy brute-force attack. In this context, the power of words resides in their ability to exploit human predictability and compromise otherwise robust security measures.

Brute Force vs. Dictionary Attack

The fundamental distinction between a brute-force and a dictionary attack lies in their methodology for generating potential password candidates:

- `Brute Force`: A pure brute-force attack systematically tests *every possible combination* of characters within a predetermined set and length. While this approach guarantees eventual success given enough time, it can be extremely time-consuming, particularly against longer or complex passwords.
- `Dictionary Attack`: In stark contrast, a dictionary attack employs a pre-compiled list of words and phrases, dramatically reducing the search space. This targeted methodology results in a far more efficient and rapid attack, especially when the target password is suspected to be a common word or phrase.

Feature	Dictionary Attack	Brute Force Attack	Explanation
`Efficiency`	Considerably faster and more resource-efficient.	Can be extremely time-consuming and resource-intensive.	Dictionary attacks leverage a pre-defined list, significantly narrowing the search space compared to brute-force.
`Targeting`	Highly adaptable and can be tailored to specific targets or systems.	No inherent targeting capability.	Wordlists can incorporate information relevant to the target (e.g., company name, employee names), increasing the success rate.
`Effectiveness`	Exceptionally effective against weak or commonly used passwords.	Effective against all passwords given sufficient time and resources.	If the target password is within the dictionary, it will be swiftly discovered. Brute force, while universally applicable, can be impractical for complex passwords due to the sheer volume of combinations.
`Limitations`	Ineffective against complex, randomly generated passwords.	Often impractical for lengthy or highly complex passwords.	A truly random password is unlikely to appear in any dictionary, rendering this attack futile. The astronomical number of possible combinations for lengthy passwords can make brute-force attacks infeasible.

Consider a hypothetical scenario where an attacker targets a company's employee login portal. The attacker might construct a specialized wordlist that incorporates the following:

- Commonly used, weak passwords (e.g., "password123," "qwerty")
- The company name and variations thereof
- Names of employees or departments
- Industry-specific jargon

Cheat Sheet

Go to Questions

Table of Contents

Introduction
Introduction
Password Security Fundamentals

Brute Force Attacks
Brute Force Attacks
Dictionary Attacks
Hybrid Attacks

Hydra
Hydra
Basic HTTP Authentication
Login Forms

Medusa
Medusa
Web Services

Custom Wordlists
Custom Wordlists

Skills Assessment
Skills Assessment Part 1
Skills Assessment Part 2

By deploying this targeted wordlist in a dictionary attack, the attacker significantly elevates their likelihood of successfully cracking employee passwords compared to a purely random brute-force endeavor.

Building and Utilizing Wordlists

Wordlists can be obtained from various sources, including:

- `Publicly Available Lists`: The internet hosts a plethora of freely accessible wordlists, encompassing collections of commonly used passwords, leaked credentials from data breaches, and other potentially valuable data. Repositories like SecLists offer various wordlists catering to various attack scenarios.
- `Custom-Built Lists`: Penetration testers can craft their wordlists by leveraging information gleaned during the reconnaissance phase. This might include details about the target's interests, hobbies, personal information, or any other data for password creation.
- `Specialized Lists`: Wordlists can be further refined to target specific industries, applications, or even individual companies. These specialized lists increase the likelihood of success by focusing on passwords that are more likely to be used within a particular context.
- `Pre-existing Lists`: Certain tools and frameworks come pre-packaged with commonly used wordlists. For instance, penetration testing distributions like ParrotSec often include wordlists like `rockyou.txt`, a massive collection of leaked passwords, readily available for use.

Here is a table of some of the more useful wordlists for login brute-forcing:

Wordlist	Description	Typical Use	Source
`rockyou.txt`	A popular password wordlist containing millions of passwords leaked from the RockYou breach.	Commonly used for password brute force attacks.	RockYou breach dataset
`top-usernames-shortlist.txt`	A concise list of the most common usernames.	Suitable for quick brute force username attempts.	SecLists
`xato-net-10-million-usernames.txt`	A more extensive list of 10 million usernames.	Used for thorough username brute forcing.	SecLists
`2023-200_most_used_passwords.txt`	A list of the 200 most commonly used passwords as of 2023.	Effective for targeting commonly reused passwords.	SecLists
`Default-Credentials/default-passwords.txt`	A list of default usernames and passwords commonly used in routers, software, and other devices.	Ideal for trying default credentials.	SecLists

Throwing a dictionary at the problem

To follow along, start the target system via the question section at the bottom of the page.

The instance application creates a route (`/dictionary`) that handles POST requests. It expects a `password` parameter in the request's form data. Upon receiving a request, it compares the submitted password against the expected value. If there's a match, it responds with a JSON object containing a success message and the flag. Otherwise, it returns an error message with a 401 status code (Unauthorized).
Copy and paste this Python script below as `dictionary-solver.py` onto your machine. You only need to modify the IP and port variables to match your target system information.

Code: python

```python
import requests

ip = "127.0.0.1"  # Change this to your instance IP address
port = 1234       # Change this to your instance port number

# Download a list of common passwords from the web and split it into lines
passwords = requests.get("https://raw.githubusercontent.com/danielmiessler/SecLists/refs/heac

# Try each password from the list
for password in passwords:
    print(f"Attempted password: {password}")

    # Send a POST request to the server with the password
    response = requests.post(f"http://{ip}:{port}/dictionary", data={'password': password})

    # Check if the server responds with success and contains the 'flag'
```

```
    if response.ok and 'flag' in response.json():
        print(f"Correct password found: {password}")
        print(f"Flag: {response.json()['flag']}")
        break
```

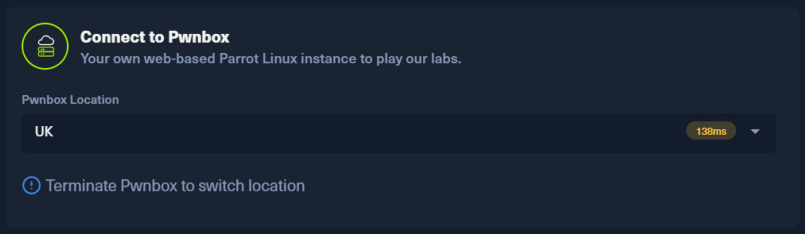
The Python script orchestrates the dictionary attack. It performs the following steps:

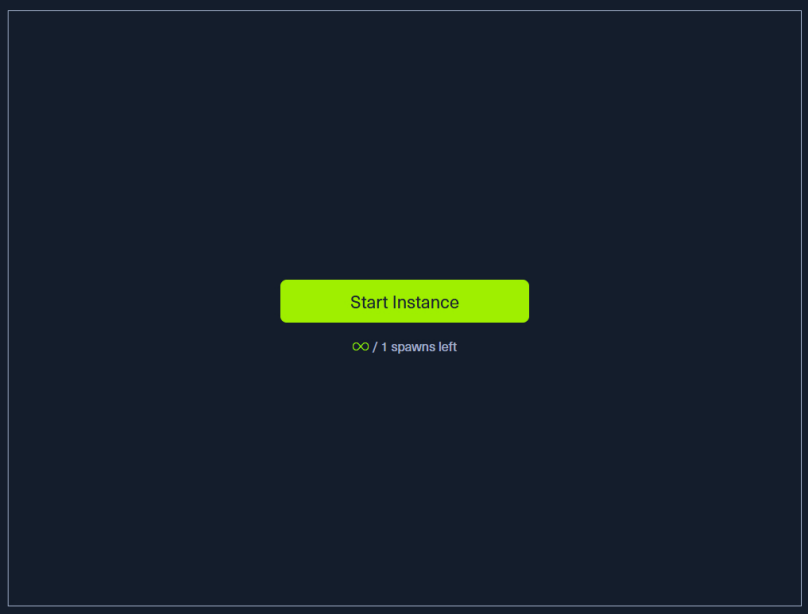
1. **Downloads the Wordlist**: First, the script fetches a wordlist of 500 commonly used (and therefore weak) passwords from SecLists using the `requests` library.
2. **Iterates and Submits Passwords**: It then iterates through each password in the downloaded wordlist. For each password, it sends a POST request to the Flask application's `/dictionary` endpoint, including the password in the request's form data.
3. **Analyzes Responses**: The script checks the response status code after each request. If it's 200 (OK), it examines the response content further. If the response contains the "flag" key, it signifies a successful login. The script then prints the discovered password and the captured flag.
4. **Continues or Terminates**: If the response doesn't indicate success, the script proceeds to the next password in the wordlist. This process continues until the correct password is found or the entire wordlist is exhausted.

Dictionary Attacks

```
MisaelMacias@htb[/htb]$ python3 dictionary-solver.py

...
Attempted password: turtle
Attempted password: tiffany
Attempted password: golf
Attempted password: bear
Attempted password: tiger
Correct password found: ...
Flag: HTB{...}
```





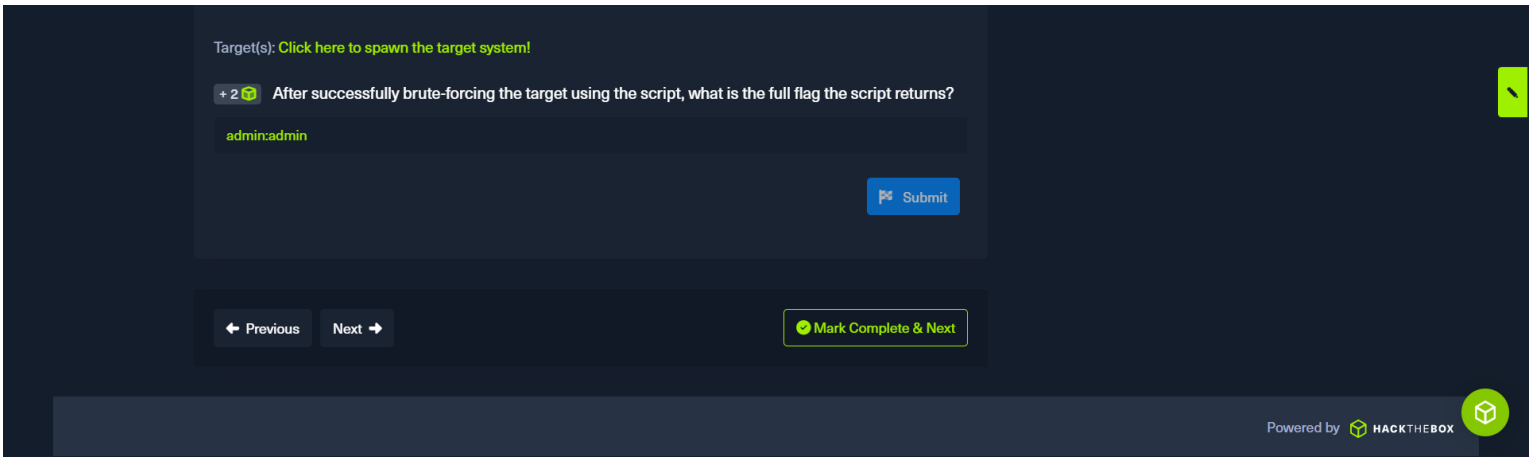
Waiting to start...

Enable step-by-step solutions for all questions

Questions

Cheat Sheet

Answer the question(s) below to complete this Section and earn cubes!


Target(s): Click here to spawn the target system!
+ 2
After successfully brute-forcing the target using the script, what is the full flag the script returns?
admin:admin
Submit
Previous
Next
Mark Complete & Next
Powered by HACKTHEBOX