Documentation & Guide

01.

Title: Automated DNS Reconnaissance & Enumeration Utility

Objective The objective of this project was to create a script that automates the discovery of public DNS records. In offensive cybersecurity, this is used to identify the "Attack Surface" of an organization. By finding Mail Servers (MX) or internal IP ranges (A/AAAA), a tester determines where to focus subsequent attacks.

Methodology & Code Logic The tool works by initializing a DNS resolver instance and iterating through a predefined list of record types.

- **Input Handling:** The script accepts a target domain and optional custom nameservers via command-line arguments.
- **Resolution Loop:** It iterates through record types (`["A", "MX", "TXT", etc.]`).
- **Exception Management:** The code uses `try/except` blocks specifically for `dns.resolver.NoAnswer` and `dns.resolver.NXDOMAIN` to differentiate between "No records found" and "Domain does not exist."

Key Code Segments (Highlight this in your PDF)

- *The Resolver Setup:*

  Python

  ```python
  resolver = dns.resolver.Resolver()
  if nameserver:
      resolver.nameservers = [nameserver]
  ```

  Explanation: This allows the tool to bypass local DNS caches and query specific upstream servers, useful for verifying split-horizon DNS.

- *The Query Logic:*

  Python

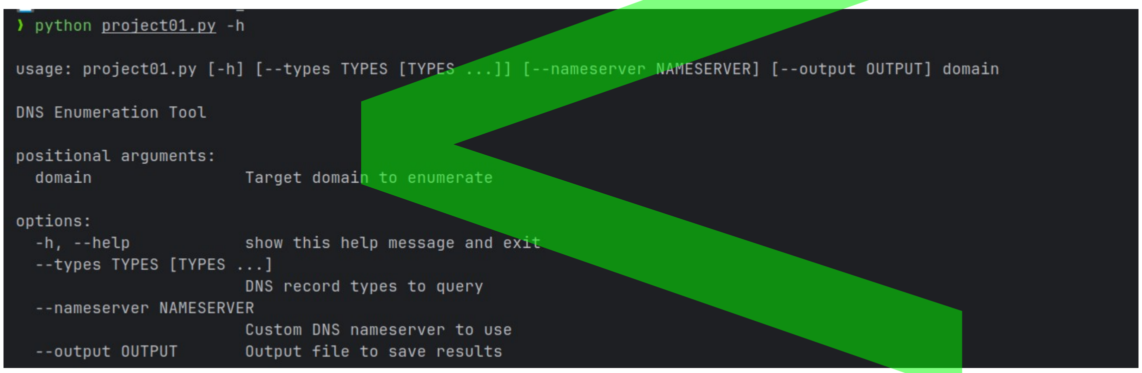
  ```python
  answer = resolver.resolve(target_domain, record_type)
  results[record_type] = [str(data) for data in answer]
  ```

  Explanation: This fetches the raw answer and converts the object data into a readable string format for the user.

Screenshots

Help Menu:

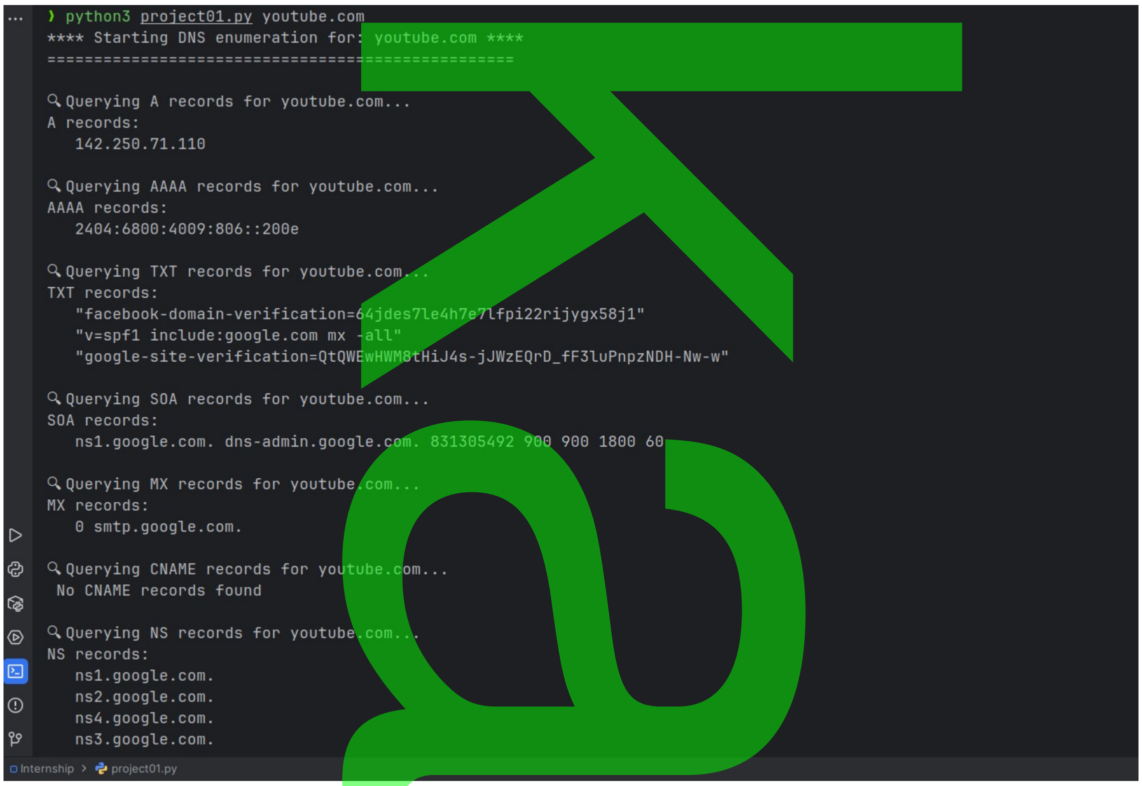
- Run: `python project01.py -h`



```
❯ python project01.py -h

usage: project01.py [-h] [--types TYPES [TYPES ...]] [--nameserver NAMESERVER] [--output OUTPUT] domain

DNS Enumeration Tool

positional arguments:
  domain                Target domain to enumerate

options:
  -h, --help            show this help message and exit
  --types TYPES [TYPES ...]
                        DNS record types to query
  --nameserver NAMESERVER
                        Custom DNS nameserver to use
  --output OUTPUT       Output file to save results
```

A Successful Scan (The "Money Shot"):

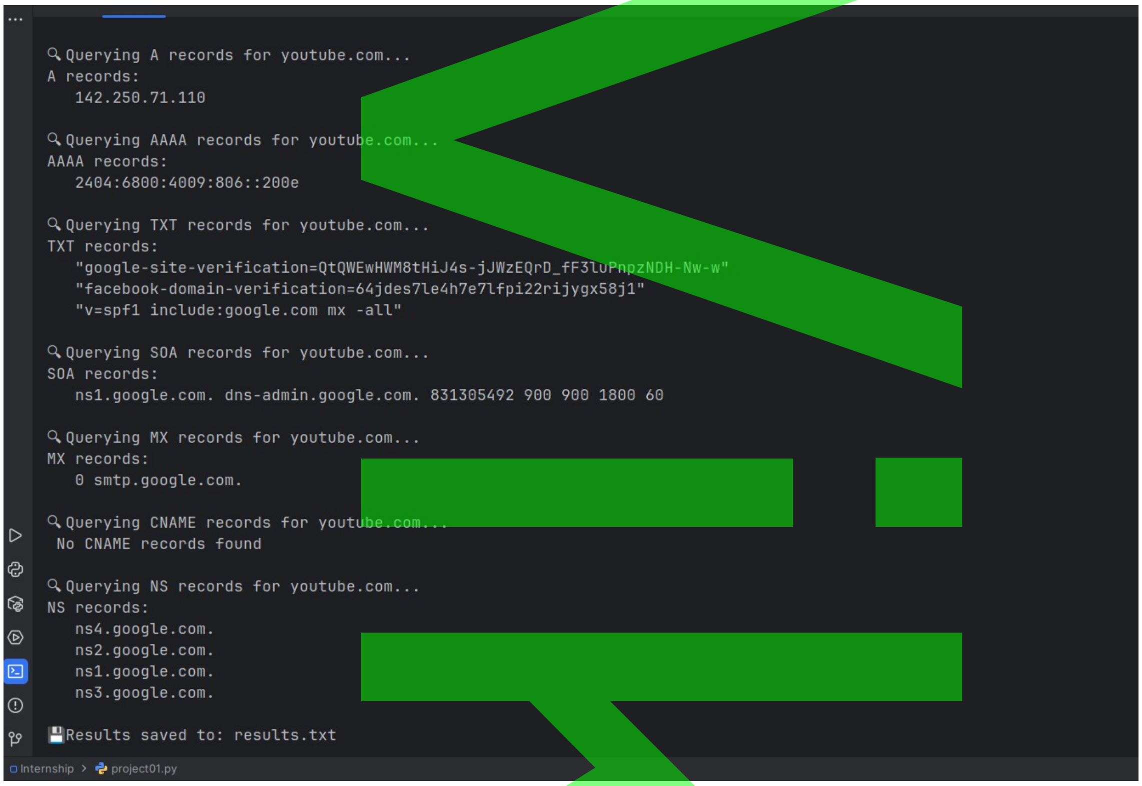
- Run: `python project01.py youtube.com` (or use `scanme.nmap`)



```
❯ python3 project01.py youtube.com
**** Starting DNS enumeration for: youtube.com ****
==================================================

🔍Querying A records for youtube.com...
A records:
   142.250.71.110

🔍Querying AAAA records for youtube.com...
AAAA records:
   2404:6800:4009:806::200e

🔍Querying TXT records for youtube.com...
TXT records:
   "facebook-domain-verification=64jdes7le4h7e7lfpi22rijygx58j1"
   "v=spf1 include:google.com mx -all"
   "google-site-verification=QtQWEwHWM8tHiJ4s-jJWzEQrD_fF3luPnpzNDH-Nw-w"

🔍Querying SOA records for youtube.com...
SOA records:
   ns1.google.com. dns-admin.google.com. 831305492 900 900 1800 60

🔍Querying MX records for youtube.com...
MX records:
   0 smtp.google.com.

🔍Querying CNAME records for youtube.com...
 No CNAME records found

🔍Querying NS records for youtube.com...
NS records:
   ns1.google.com.
   ns2.google.com.
   ns4.google.com.
   ns3.google.com.
```

The Output File:

- Run: `python project01.py youtube.com --output results.txt`



```
🔍Querying A records for youtube.com...
A records:
   142.250.71.110

🔍Querying AAAA records for youtube.com...
AAAA records:
   2404:6800:4009:806::200e

🔍Querying TXT records for youtube.com...
TXT records:
   "google-site-verification=QtQWEwHWM8tHiJ4s-jJWzEQrD_fF3luPnpzNDH-Nw-w"
   "facebook-domain-verification=64jdes7le4h7e7lfpi22rijygx58j1"
   "v=spf1 include:google.com mx -all"

🔍Querying SOA records for youtube.com...
SOA records:
   ns1.google.com. dns-admin.google.com. 831305492 900 900 1800 60

🔍Querying MX records for youtube.com...
MX records:
   0 smtp.google.com.

🔍Querying CNAME records for youtube.com...
 No CNAME records found

🔍Querying NS records for youtube.com...
NS records:
   ns4.google.com.
   ns2.google.com.
   ns1.google.com.
   ns3.google.com.

💾Results saved to: results.txt
```

- Then open `results.txt` in Notepad or `cat results.txt` in the terminal.

02.

Title: Multi-Threaded Subdomain Enumeration Tool

Objective The primary objective was to automate the discovery of subdomains (e.g., `dev.target.com`, `admin.target.com`) that are not publicly advertised. In a penetration test, this phase expands the scope of the audit, often revealing development servers or forgotten admin panels that are less secure than the main website.

Methodology & Code Logic The tool utilizes a "Dictionary Attack" approach:

- **Input:** Takes a target domain and a wordlist (a text file containing common subdomain names).
- **Parallel Processing:** Instead of checking one domain at a time, the script spins up a pool of worker threads.
- **Verification:** Each thread constructs a URL (`http://sub.domain.com`), sends a GET request, and analyzes the response code to determine if the site is active.

Key Code Segments

- *Concurrency Implementation:*

Python

```
with ThreadPoolExecutor(max_workers=self.threads) as executor:
    future_to_subdomain = {
        executor.submit(self.check_subdomain, subdomain): subdomain
        for subdomain in subdomains
    }
```

Explanation: This segment initializes the thread pool, allowing multiple network requests to happen at the exact same time, drastically speeding up the enumeration process.

- *The Verification Logic:*

  Python

```
response = self.session.get(url, timeout=self.timeout)
if response.status_code < 400:
    return url
```

Explanation: We strictly filter for status codes below 400. This ensures we don't report broken links (404) or server errors (500) as valid, discoverable assets.)

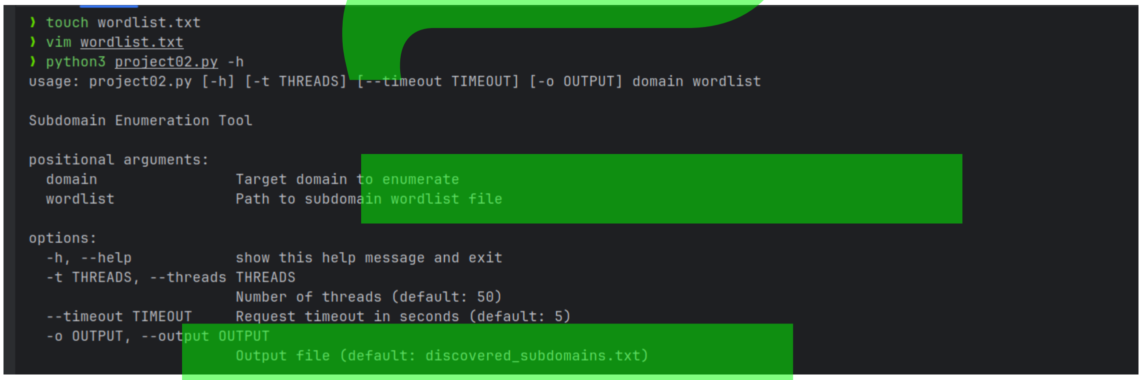
Screenshots

For this tool, we will need a "wordlist" file to make it work. Create a file named `wordlist.txt` and put these words inside it:

```
www
mail
remote
blog
webmail
server
help
secure
```

The Help Menu:

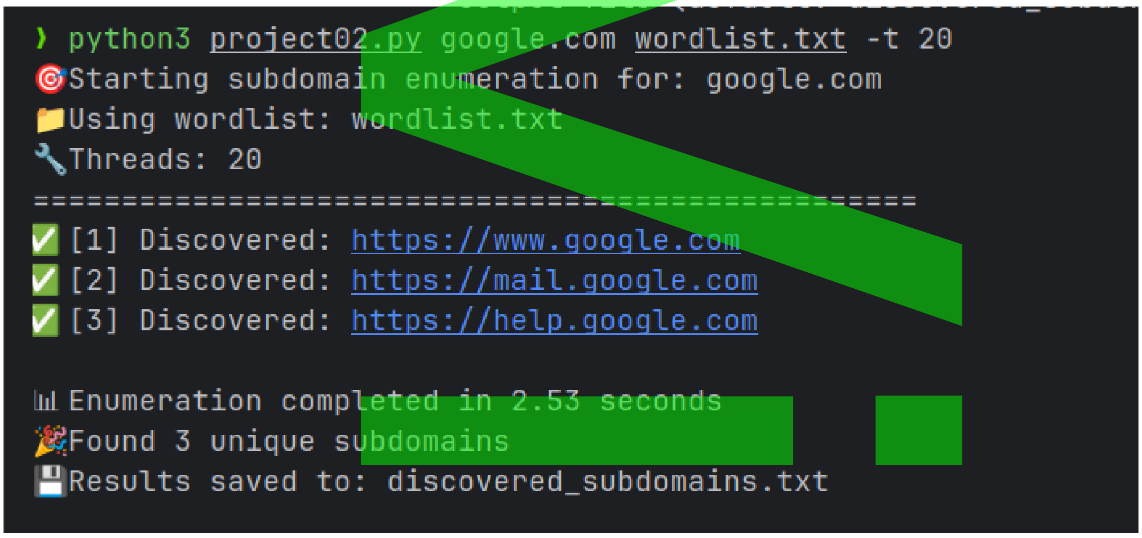
- Run: `python project02.py -h`



```
❯ touch wordlist.txt
❯ vim wordlist.txt
❯ python3 project02.py -h
usage: project02.py [-h] [-t THREADS] [--timeout TIMEOUT] [-o OUTPUT] domain wordlist

Subdomain Enumeration Tool

positional arguments:
  domain                Target domain to enumerate
  wordlist              Path to subdomain wordlist file

options:
  -h, --help            show this help message and exit
  -t THREADS, --threads THREADS
                        Number of threads (default: 50)
  --timeout TIMEOUT     Request timeout in seconds (default: 5)
  -o OUTPUT, --output OUTPUT
                        Output file (default: discovered_subdomains.txt)
```

The Active Scan:

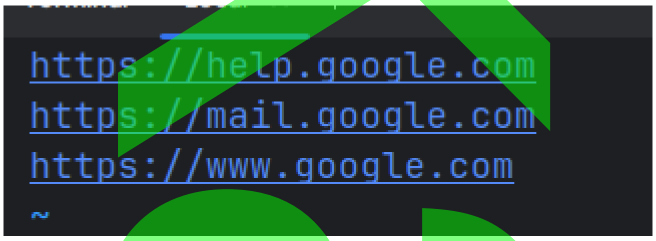
- Run: `python project02.py google.com wordlist.txt -t 20`



```
❯ python3 project02.py google.com wordlist.txt -t 20
🎯Starting subdomain enumeration for: google.com
📁Using wordlist: wordlist.txt
🔧Threads: 20
==================================================
✅ [1] Discovered: https://www.google.com
✅ [2] Discovered: https://mail.google.com
✅ [3] Discovered: https://help.google.com

📊Enumeration completed in 2.53 seconds
🎉Found 3 unique subdomains
💾Results saved to: discovered_subdomains.txt
```

The Results File:

- Open the generated `discovered_subdomains.txt`.



```
https://help.google.com
https://mail.google.com
https://www.google.com
~
```

03.

Title: PDF Encryption & Data Loss Prevention (DLP) Tool

Objective The objective of this tool is to enforce **Data Confidentiality**. in a corporate environment, sensitive reports (like penetration test results) must be encrypted before being shared via email or stored on shared drives. This tool automates that security control.

Methodology & Code Logic The script follows a "Stream-based" approach:

- **Stream Reading:** It opens the target file in binary read mode (`rb`).
- **Page Extraction:** It extracts page objects from the source to ensure no content is lost.
- **Cryptographic Wrapper:** It creates a new `PdfWriter` object and applies the `encrypt()` method, which wraps the file content in a security layer requiring a password key.
- **Binary Writing:** Finally, it writes the encrypted stream to a new file in binary write mode (`wb`).

Key Code Segments

- *The Encryption Implementation:*

  Python

  ```
  pdf_writer = PyPDF2.PdfWriter()
  # Copy all pages
  for page in pdf_reader.pages:
      pdf_writer.add_page(page)

  # Encrypt with password
  pdf_writer.encrypt(password)
  ```

  Explanation: This is the core security logic. We migrate the data to a new writer instance and apply the encryption method before the file is ever saved to the disk.

- *Binary File Handling:*

  Python

  ```
  with open(input_pdf, 'rb') as file:
      # ... processing ...
  with open(output_pdf, 'wb') as output:
      pdf_writer.write(output)
  ```

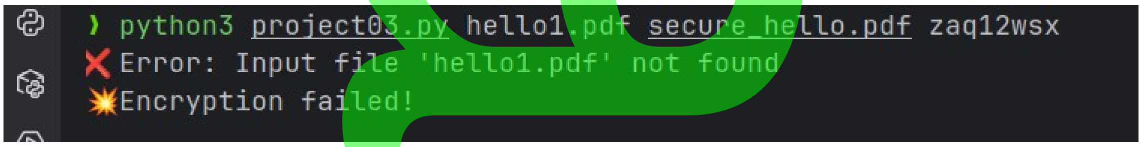
  Explanation: Using 'rb' (read binary) and 'wb' (write binary) is critical to prevent encoding errors that could corrupt the PDF structure.

Screenshots

To demonstrate this, we need a dummy PDF. Create a simple PDF named hello.`pdf` (you can just save a Word doc as PDF).

The Validation Error:

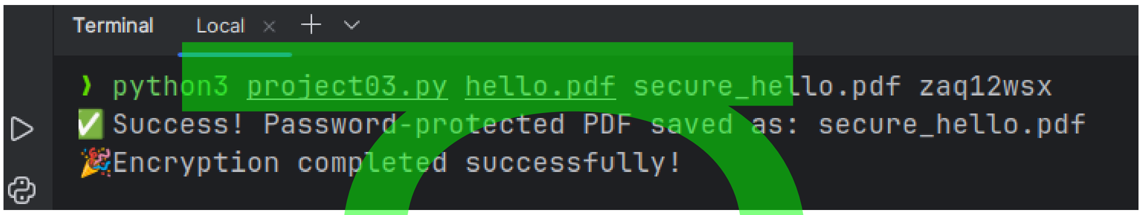
- Run: `python project03.py hello1.pdf secure_hello.pdf zaq12wsx`



```
❯ python3 project03.py hello1.pdf secure_hello.pdf zaq12wsx
❌Error: Input file 'hello1.pdf' not found
💥Encryption failed!
```

The Successful Encryption:

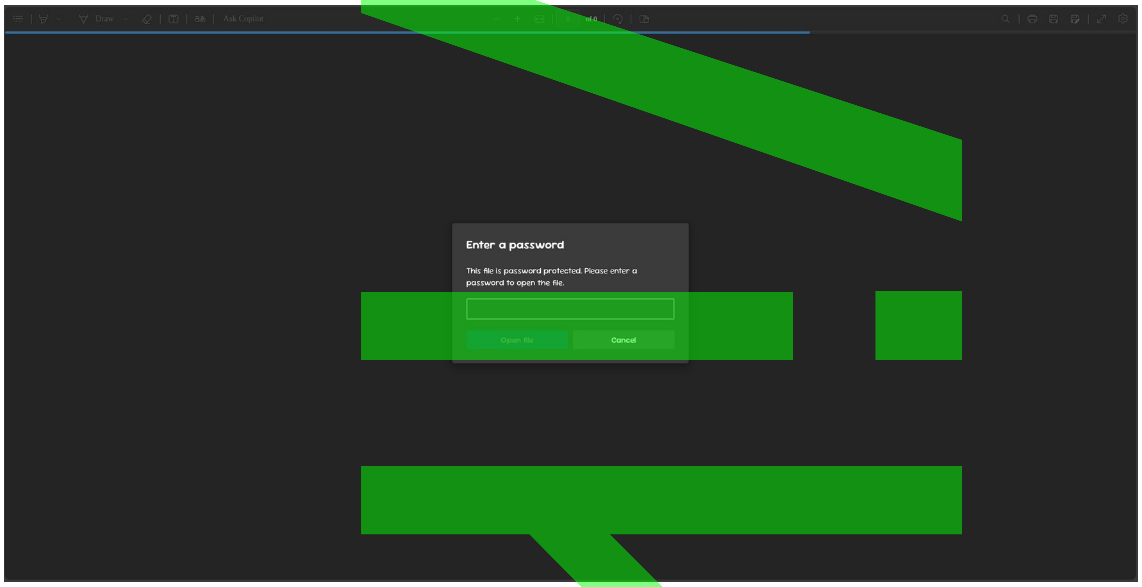
- Run: `python project03.py hello.pdf secure_hello.pdf SuperSecret123Pass`



```
❯ python3 project03.py hello.pdf secure_hello.pdf zaq12wsx
✅Success! Password-protected PDF saved as: secure_hello.pdf
🎉Encryption completed successfully!
```

The Proof (Most Important):

- Double-click `protected.pdf` to open it in a PDF viewer (like Chrome, Edge, or Adobe).


Enter a password

This file is password protected. Please enter a password to open the file.

04.

Title: PDF Password Auditing & Recovery Utility

Objective The objective of this project is to demonstrate the vulnerability of weak passwords. In a security audit, this tool is used to verify if employees are using easily guessable passwords (like "password123" or short combinations) that fail to meet compliance standards.

Methodology & Code Logic The script employs two distinct algorithms:

- **Dictionary Mode:** It reads a text file containing thousands of common passwords and attempts them one by one. This mimics an attacker using a database of breached credentials.
- **Brute-Force Mode:** It mathematically generates every possible combination of characters (e.g., `aa`, `ab`, `ac`...) up to a specific length.
- **Verification:** It attempts to open the PDF with the generated string. If `pikepdf` does not raise a `PasswordError`, the tool confirms the password is correct.

Key Code Segments

- *The Cracking Logic:*

  Python

```
try:
    with pikepdf.open(pdf_file, password=password) as pdf:
        return password
except pikepdf.PasswordError:
    return None
```

Explanation: We attempt to open the file stream with the candidate password. The library throws a specific exception if the password is wrong, which we catch to continue the loop.

- *The Permutation Logic (Brute Force):*

  Python

  ```
  from itertools import product
  for length in range(min_len, max_len + 1):
      for pwd_tuple in product(charset, repeat=length):
          passwords.append(''.join(pwd_tuple))
  ```

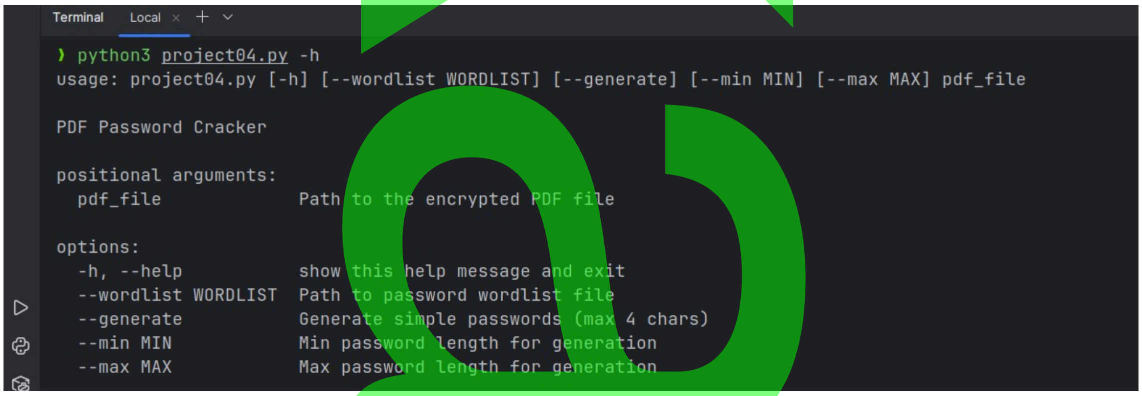
  Explanation: This segment dynamically generates every possible character combination, ensuring 100% coverage of the search space for the defined length.

Screenshots

To demonstrate this, we need the `protected pdf` , so we will use that created in Project 03 (i.e. secure_hello.pdf ; set the password to something simple like `abc` for this test).

The Help Menu:

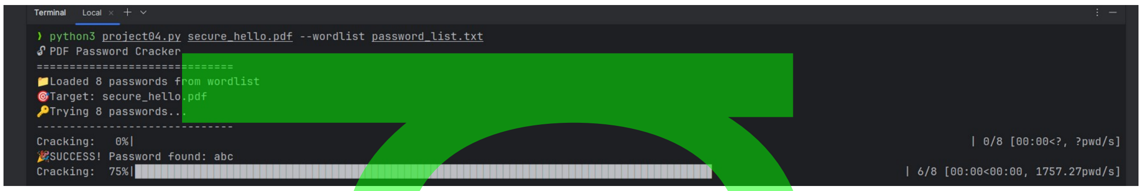
- Run: `python project04.py -h`



```
❯ python3 project04.py -h
usage: project04.py [-h] [--wordlist WORDLIST] [--generate] [--min MIN] [--max MAX] pdf_file

PDF Password Cracker

positional arguments:
  pdf_file              Path to the encrypted PDF file

options:
  -h, --help            show this help message and exit
  --wordlist WORDLIST   Path to password wordlist file
  --generate            Generate simple passwords (max 4 chars)
  --min MIN             Min password length for generation
  --max MAX             Max password length for generation
```

The Dictionary Attack (Simulation):

- Create a file `passwords.txt` with random words and include the real password (`abc`) somewhere in the middle.

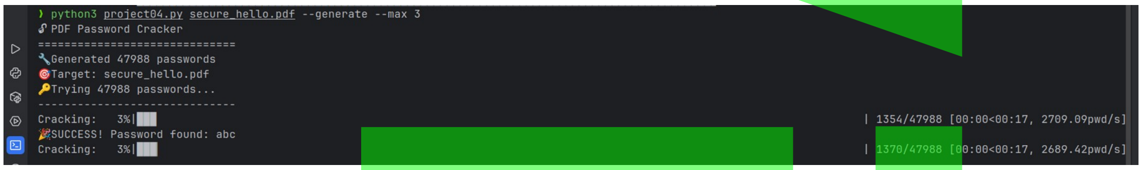


```
❯ python3 project04.py secure_hello.pdf --wordlist password_list.txt
🔓PDF Password Cracker
==============================
📁Loaded 8 passwords from wordlist
🎯Target: secure_hello.pdf
🔑Trying 8 passwords...
------------------------------
Cracking:   0%|                                                              | 0/8 [00:00<?, ?pwd/s]
🎉SUCCESS! Password found: abc
Cracking:  75%|████████████████████████████████████████████          | 6/8 [00:00<00:00, 1757.27pwd/s]
```

- Run: `python3 project04.py secure_hello.pdf --wordlist password_list.txt`
-

The Brute Force Attack:

- Run: `python project04.py secure_hello.pdf --generate --max 3`



```
❯ python3 project04.py secure_hello.pdf --generate --max 3
🔓PDF Password Cracker
==============================
🔧Generated 47988 passwords
🎯Target: secure_hello.pdf
🔑Trying 47988 passwords...
------------------------------
Cracking:   3%|███                                                        | 1354/47988 [00:00<00:17, 2709.09pwd/s]
🎉SUCCESS! Password found: abc
Cracking:   3%|███                                                        | 1370/47988 [00:00<00:17, 2689.42pwd/s]
```

05.

Title: Multi-Threaded Network Infrastructure Mapping & Banner Grabbing tool

Objective The objective of this tool is to perform **Active Reconnaissance**. Before a penetration tester can exploit a system, they must know what "doors" (ports) are open and what software is running behind them. This tool automates the discovery of the "Attack Surface."

Methodology & Code Logic The script operates on the TCP protocol:

- **Target Resolution:** Converts the hostname to an IPv4 address.
- **Threading Model:** Instead of checking Port 1, waiting, then Port 2, it spins up a "Pool" of workers (threads) to check hundreds of ports simultaneously.
- **TCP Handshake:** It attempts a full TCP connection (`socket.connect_ex`). If the return code is 0, the port is OPEN.
- **Data Extraction:** If open, it sends a request and listens for a "Banner"—a string of text the server sends back identifying itself.

Key Code Segments

- *The Banner Grabbing Logic:*

  Python

```python
try:
    sock.settimeout(0.5)
    banner = sock.recv(1024).decode(errors='ignore').strip()
except:
    pass
```

Explanation: This segment listens for the "Welcome Message" from the server. This data is critical for Version Detection, which helps hackers find specific exploits for that software version.

- *The Threading Implementation:*

  Python

  ```
  with concurrent.futures.ThreadPoolExecutor(max_workers=200) as executor:
      future_to_port = {
          executor.submit(scan_port, target_ip, port): port
          for port in range(start_port, end_port + 1)
      }
  ```

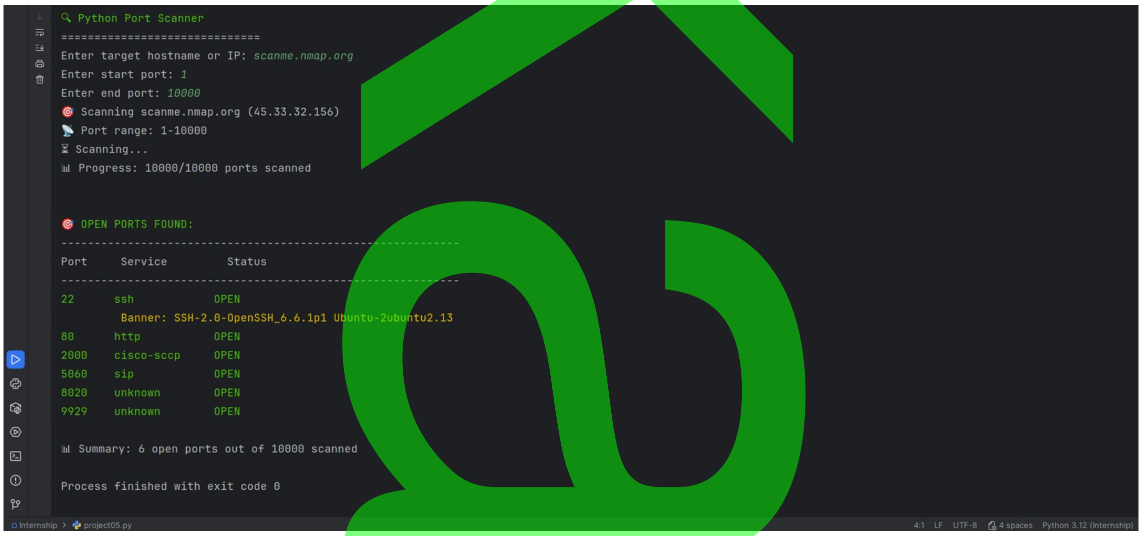
  Explanation: This code creates 200 parallel threads. Without this, scanning all 65,535 ports would take hours; with this, it takes minutes.

Screenshots

Crucial Note: Do not scan random websites (like Google or Facebook). It is illegal or against their Terms of Service. Use `scanme.nmap.org` (a site specifically made for testing scanners) or your own localhost.

The Input/Scan Screen:

- Run: `python project05.py`
- Enter Target: `scanme.nmap.org`



```
🔍 Python Port Scanner
==============================
Enter target hostname or IP: scanme.nmap.org
Enter start port: 1
Enter end port: 10000
🎯 Scanning scanme.nmap.org (45.33.32.156)
📡 Port range: 1-10000
⏳ Scanning...
📊 Progress: 10000/10000 ports scanned


🎯 OPEN PORTS FOUND:
------------------------------------------------------------
Port     Service          Status
------------------------------------------------------------
22       ssh              OPEN
         Banner: SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.13
80       http             OPEN
2000     cisco-sccp       OPEN
5060     sip              OPEN
8020     unknown          OPEN
9929     unknown          OPEN

📊 Summary: 6 open ports out of 10000 scanned

Process finished with exit code 0
```

Internship > project05.py
4:1 LF UTF-8 4 spaces Python 3.12 (Internship)

06.

Title: Offline Password Hash Auditing Tool

Objective The objective of this project is to simulate "Offline Password Attacks." When a database is breached, passwords are often stored as hashes. This tool tests whether those hashes can be reversed to reveal the original plaintext password, highlighting the need for "Salted" hashes and high-entropy passwords.

Methodology & Code Logic The tool works by "Forward Hashing":

- **Concept:** You cannot "decrypt" a hash. You must take a guess, hash the guess, and compare it to the stolen hash.
- **Input:** The target hash string and the algorithm type (e.g., MD5).
- **Generation:** The script generates candidates either from a wordlist or by mathematically creating every character combination (Brute Force).
- **Comparison:** `if md5(guess) == target_hash: return True`

Key Code Segments

- *Dynamic Hashing Wrapper:*

  Python

  ```
  hash_fn = getattr(hashlib, hash_type)
  # ... later in the code ...
  return hash_fn(password.encode()).hexdigest() == target_hash
  ```

  Explanation: Instead of writing separate functions for MD5, SHA1, etc., we use Python's reflection capabilities to select the algorithm at runtime based on the user's CLI argument.

- *The Brute Force Generator:*

  Python

  ```
  def password_generator(min_length, max_length, characters):
      for length in range(min_length, max_length + 1):
          for password in itertools.product(characters, repeat=length):
              yield "".join(password)
  ```

  Explanation: This generator uses `itertools.product` to create a stream of password candidates without storing them all in RAM, making the tool memory efficient.

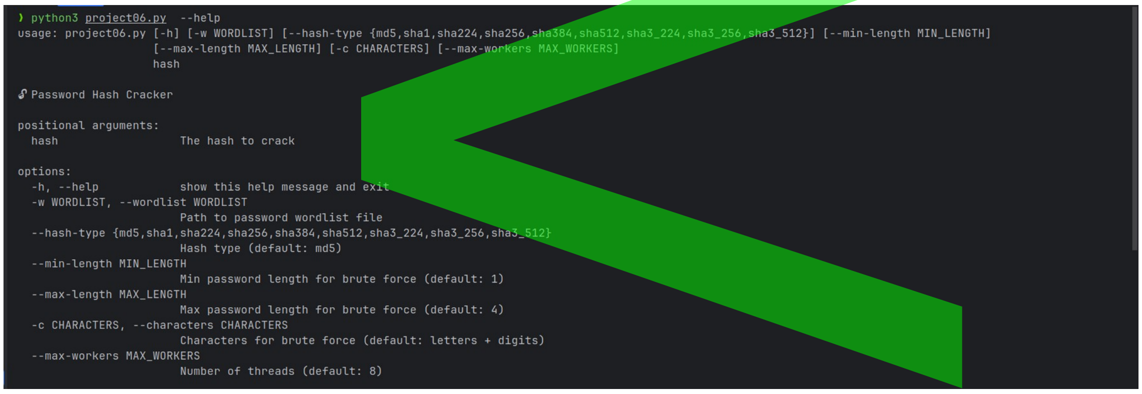
Screenshots

We first need to **generate a hash** to crack.

- Go to an online MD5 generator (or use Python: `import hashlib; print(hashlib.md5(b"abc").hexdigest())`).
- Let's use the hash for the password **"abc"**.
- **MD5 Hash for "abc":** `900150983cd24fb0d6963f7d28e17f72`

The Help Menu:

- Run: `python project06.py –help`

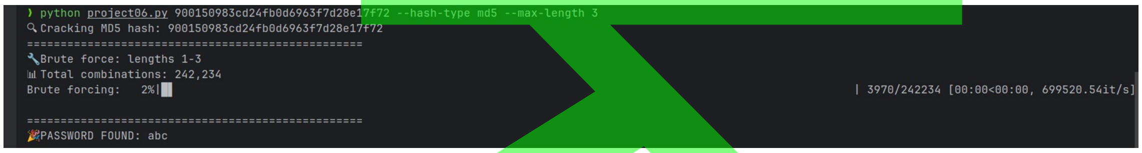


```
❯ python3 project06.py  --help
usage: project06.py [-h] [-w WORDLIST] [--hash-type {md5,sha1,sha224,sha256,sha384,sha512,sha3_224,sha3_256,sha3_512}] [--min-length MIN_LENGTH]
                    [--max-length MAX_LENGTH] [-c CHARACTERS] [--max-workers MAX_WORKERS]
                    hash

🔐 Password Hash Cracker

positional arguments:
  hash                  The hash to crack

options:
  -h, --help            show this help message and exit
  -w WORDLIST, --wordlist WORDLIST
                        Path to password wordlist file
  --hash-type {md5,sha1,sha224,sha256,sha384,sha512,sha3_224,sha3_256,sha3_512}
                        Hash type (default: md5)
  --min-length MIN_LENGTH
                        Min password length for brute force (default: 1)
  --max-length MAX_LENGTH
                        Max password length for brute force (default: 4)
  -c CHARACTERS, --characters CHARACTERS
                        Characters for brute force (default: letters + digits)
  --max-workers MAX_WORKERS
                        Number of threads (default: 8)
```

- Shows the list of supported algorithms (MD5, SHA1, SHA256, etc.).

Successful Brute Force (MD5):

- Run: `python project06.py 900150983cd24fb0d6963f7d28e17f72 --hash-type md5 --max-length 3`



```
❯ python project06.py 900150983cd24fb0d6963f7d28e17f72 --hash-type md5 --max-length 3
🔍Cracking MD5 hash: 900150983cd24fb0d6963f7d28e17f72
==================================================
🔧Brute force: lengths 1-3
📊Total combinations: 242,234
Brute forcing:   2%|█                                                                              | 3970/242234 [00:00<00:00, 699520.54it/s]

==================================================
🎉PASSWORD FOUND: abc
```

- *Why:* It will quickly find "abc" and show the "🎉 PASSWORD FOUND" message.

Note: This program is not yet suitable for huge string password.