Task 1

Penetration Testing Toolkit

PORT SCANNER:-

```
    # port_scanner.py
import socket
import sys
from datetime import datetime


def scan_port(target_host, port):
   """
   Attempts to connect to a specific port on the target host.
   Returns True if the port is open, False otherwise.
   """
   try:
      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
         socket.setdefaulttimeout(0.5) # Set a timeout for the connection attempt
         result = s.connect_ex((target_host, port))
         if result == 0:
            return True
         else:
            return False
   except socket.gaierror:
      print(f"[!] Error: Hostname {target_host} could not be resolved.")
      return None # Indicate an error occurred
   except socket.error as e:
      print(f"[!] Error: Could not connect to server. Reason: {e}")
      return None # Indicate an error occurred
```

```
    except KeyboardInterrupt:
        print("\n[*] Exiting Port Scanner Module.")
        sys.exit()


def scan_range(target_host, start_port, end_port):
    """
    Scans a range of ports on the target host.
    """
    print("-" * 60)
    print(f"Scanning target: {target_host}")
    print(f"Time started: {datetime.now()}")
    print("-" * 60)

    open_ports = []

    try:
        target_ip = socket.gethostbyname(target_host) # Resolve hostname to IP
        print(f"Target IP: {target_ip}")
        print("Scanning ports...")

        for port in range(start_port, end_port + 1):
            is_open = scan_port(target_ip, port)
            if is_open is True:
                print(f"[+] Port {port}/tcp is open")
                open_ports.append(port)
            elif is_open is None: # Handle errors from scan_port
                 print(f"[!] Error scanning port {port}. Skipping.")
```

```
        print("-" * 60)
        if open_ports:
            print("Scan Complete. Open ports found:")
            print(open_ports)
        else:
            print("Scan Complete. No open ports found in the specified range.")
        print("-" * 60)
        return open_ports

    except socket.gaierror:
        print(f"[!] Error: Hostname {target_host} could not be resolved.")
        return []
    except KeyboardInterrupt:
        print("\n[*] Exiting Port Scanner Module.")
        sys.exit()
    except Exception as e:
         print(f"[!] An unexpected error occurred: {e}")
         return []

# Example of how to call this module (could be done from main.py)
# if __name__ == "__main__":
#    target = "scanme.nmap.org" # Example target - USE WITH PERMISSION ONLY
#    scan_range(target, 1, 100)
```

BRUTEFORCER:-

```
    # brute_forcer.py
import itertools
import string
import time

# --- Placeholder for actual connection logic ---
def attempt_login(target, username, password):
    """
    Placeholder function: Replace this with actual logic
    to attempt a login using the provided credentials against
    a specific service (e.g., SSH, FTP, Web Form).

    Returns True if login is successful, False otherwise.
    """
    print(f"[*] Trying U: {username} P: {password}")
    # Example: Simulate a check (replace with real connection/auth)
    # import requests
    # response = requests.post(f"http://{target}/login", data={'user': username, 'pass': password})
    # if "Login successful" in response.text:
    #     return True

    # Simulate time delay
    time.sleep(0.1)

    # Simulate finding the correct password (FOR DEMO ONLY)
    if password == "pa55":
```

```
        print(f"[+] Success! Found credentials: {username}/{password}")
        return True
    return False
# --- End Placeholder ---

def generate_passwords(charset, min_len, max_len):
    """
    Generates password candidates based on charset and length range.
    """
    for length in range(min_len, max_len + 1):
        for guess in itertools.product(charset, repeat=length):
            yield "".join(guess)

def brute_force(target, username, charset=string.ascii_lowercase + string.digits, min_len=4,
max_len=6):
    """
    Attempts to brute-force a password for a given username and target.
    """
    print("-" * 60)
    print(f"Starting brute-force attack on {target} for user {username}")
    print(f"Charset: '{charset}', Length: {min_len}-{max_len}")
    print("-" * 60)

    password_generator = generate_passwords(charset, min_len, max_len)

    try:
        for password in password_generator:
            if attempt_login(target, username, password):
                # Stop if password found
                return (username, password)
```

```
        print("[!] Brute-force attempt finished. Password not found within the given parameters.")
        return None
    except KeyboardInterrupt:
        print("\n[*] Exiting Brute-Forcer Module.")
        return None
    except Exception as e:
        print(f"[!] An error occurred during brute-force: {e}")
        return None



# Example of how to call this module (could be done from main.py)
# if __name__ == "__main__":
#     # IMPORTANT: Only run against systems you OWN or have EXPLICIT PERMISSION to test.
#     target_host = "127.0.0.1" # Example: A local test service
#     user = "admin"
#     found_creds = brute_force(target_host, user, min_len=4, max_len=4) # Short params for quick demo
#     if found_creds:
#         print(f"\n[+] Credentials Found: {found_creds[0]}:{found_creds[1]}")
#     else:
#         print("\n[-] Failed to find credentials.")
```

MAIN SCRIPT:-

```
 # main.py
import argparse
from toolkit import port_scanner, brute_forcer # Assuming toolkit is a package

def main():
   parser = argparse.ArgumentParser(description="Modular Penetration Testing Toolkit")
   subparsers = parser.add_subparsers(dest='module', help='Available modules')

   # Port Scanner Arguments
   parser_ps = subparsers.add_parser('scan', help='Port Scanner Module')
   parser_ps.add_argument('target', help='Target host or IP address')
   parser_ps.add_argument('-p', '--ports', default='1-1024', help='Port range (e.g., 80, 22-25, 1-65535)')

   # Brute Forcer Arguments (Example)
   parser_bf = subparsers.add_parser('brute', help='Brute Forcer Module (Use Responsibly!)')
   parser_bf.add_argument('target', help='Target service host/IP')
   parser_bf.add_argument('-u', '--username', required=True, help='Username for brute-force attempt')
   # Add more options: password list, charset, length, protocol etc.

   args = parser.parse_args()

   if args.module == 'scan':
      # Basic port range parsing (can be made more robust)
      try:
         if '-' in args.ports:
```

```
            start, end = map(int, args.ports.split('-'))
            port_scanner.scan_range(args.target, start, end)
        else:
            port = int(args.ports)
            is_open = port_scanner.scan_port(args.target, port)
            if is_open:
                print(f"[+] Port {port}/tcp is open")
            elif is_open is False:
                print(f"[-] Port {port}/tcp is closed")
            # Handle is_open is None (error) if needed

    except ValueError:
        print("[!] Invalid port specification. Use format 'port' or 'start-end'.")
    except Exception as e:
        print(f"[!] Error during port scan execution: {e}")



elif args.module == 'brute':
    print("[!] WARNING: Execute brute-force attacks only on systems you have explicit permission to test.")
    # Call the brute_forcer function (implement specific logic)
    # brute_forcer.brute_force(args.target, args.username, ...) # Add other necessary args
    print(f"[*] Brute-force module called for {args.username}@{args.target}. (Implement actual logic)")
    # Example call to the conceptual brute_force function:
    # brute_forcer.brute_force(args.target, args.username)

else:
    parser.print_help()
```

```
if __name__ == "__main__":
    main()
```