

# RECON

Fast TCP port scanner + basic service probe

Python script:

```
#!/usr/bin/env python3
```

```
"""
```

Simple TCP port scanner + banner probe.

Usage: python3 recon.py <target\_ip> [start\_port] [end\_port]

Example: python3 recon.py 192.168.56.101 1 1024

```
"""
```

```
import socket
```

```
import sys
```

```
import concurrent.futures
```

```
from datetime import datetime
```

```
TARGET = sys.argv[1] if len(sys.argv) > 1 else None
```

```
START = int(sys.argv[2]) if len(sys.argv) > 2 else 1
```

```
END = int(sys.argv[3]) if len(sys.argv) > 3 else 1024
```

```
TIMEOUT = 1.0
```

```
if not TARGET:
```

```
    print("Usage: python3 recon.py <target_ip> [start_port] [end_port]")
```

```
    sys.exit(1)
```

```
open_ports = []
```

```
def probe_port(port):
```

```
    try:
```

```
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

```
            s.settimeout(TIMEOUT)
```

```
            r = s.connect_ex((TARGET, port))
```

```
            if r == 0:
```

```
                try:
```

```
                    s.send(b'HEAD / HTTP/1.0\r\n\r\n')
```

```
                    banner = s.recv(1024).decode(errors='ignore').strip()
```

Jayesh R. Yadav

```

        except Exception:
            banner = ""
        return port, banner
    except Exception:
        pass
    return None

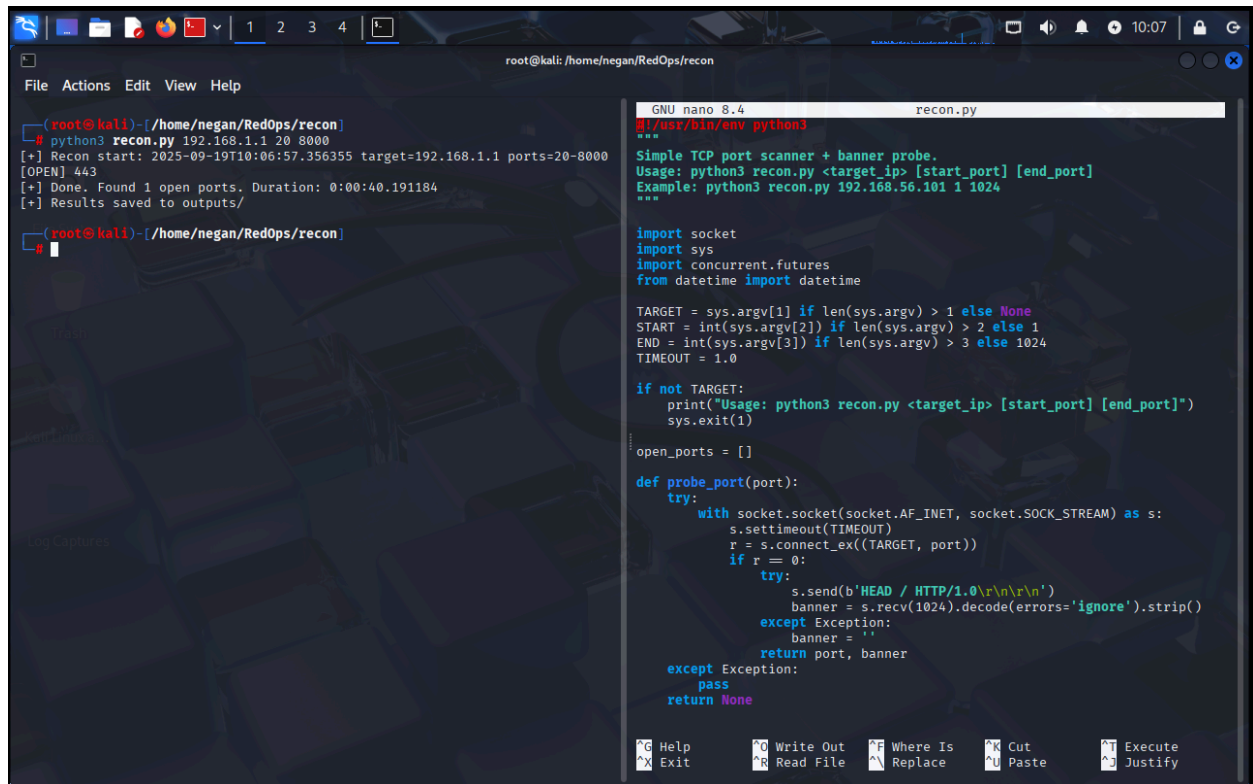
start = datetime.now()
print(f"[+] Recon start: {start.isoformat()} target={TARGET} ports={START}-{END}")

with concurrent.futures.ThreadPoolExecutor(max_workers=200) as ex:
    futures = {ex.submit(probe_port, p): p for p in range(START, END+1)}
    for fut in concurrent.futures.as_completed(futures):
        res = fut.result()
        if res:
            port, banner = res
            open_ports.append((port, banner))
            print(f"[OPEN] {port} {banner}")

end = datetime.now()
print(f"[+] Done. Found {len(open_ports)} open ports. Duration: {end-start}")
# Save to outputs
import json, os
os.makedirs("../outputs", exist_ok=True)
with open("../outputs/recon_{}.json".format(TARGET), "w") as f:
    json.dump({"target": TARGET, "ports": open_ports, "scanned_range": [START, END],
"started": start.isoformat()}, f, indent=2)
print(f"[+] Results saved to outputs/")

```

Working :



The screenshot shows a Kali Linux desktop environment. In the foreground, a terminal window displays the execution of a Python script named `recon.py`. The command executed is `python3 recon.py 192.168.1.1 20 8000`. The output shows the script successfully scanned the target IP, found one open port (443), and saved the results to an output file. In the background, a nano editor window shows the source code of `recon.py`. The script is a simple TCP port scanner that takes a target IP, a start port, and an end port as arguments. It uses the `socket` module to establish connections and the `datetime` module for timing. The script prints the usage and example usage at the top. The main logic is in the `probe_port` function, which attempts to connect to the target IP on the specified port and returns the port and banner if successful.

```
GNU nano 8.4 recon.py
#!/usr/bin/env python3
"""
Simple TCP port scanner + banner probe.
Usage: python3 recon.py <target_ip> [start_port] [end_port]
Example: python3 recon.py 192.168.56.101 1 1024
"""

import socket
import sys
import concurrent.futures
from datetime import datetime

TARGET = sys.argv[1] if len(sys.argv) > 1 else None
START = int(sys.argv[2]) if len(sys.argv) > 2 else 1
END = int(sys.argv[3]) if len(sys.argv) > 3 else 1024
TIMEOUT = 1.0

if not TARGET:
    print("Usage: python3 recon.py <target_ip> [start_port] [end_port]")
    sys.exit(1)

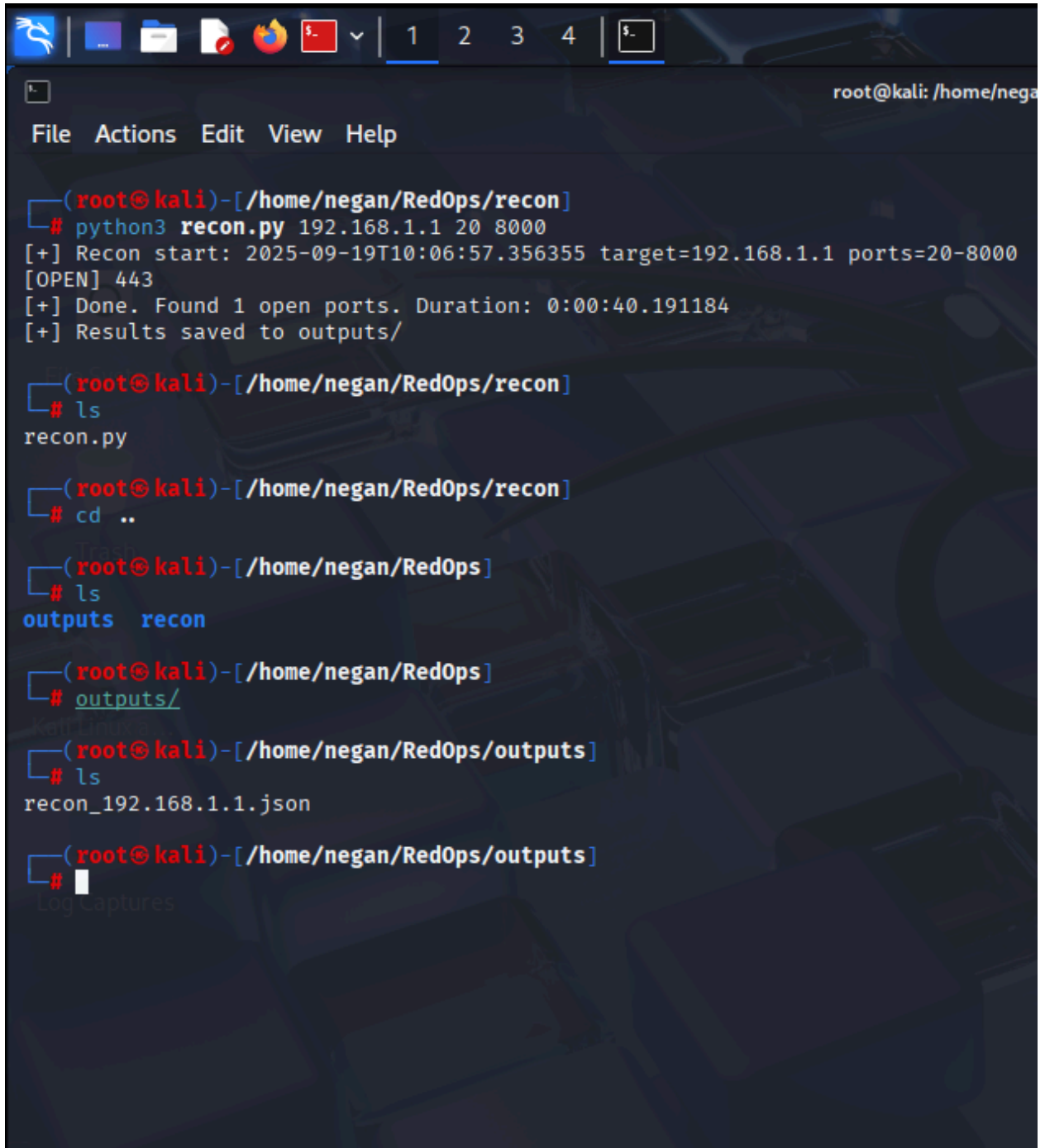
open_ports = []

def probe_port(port):
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.settimeout(TIMEOUT)
            r = s.connect_ex((TARGET, port))
            if r == 0:
                try:
                    s.send(b'HEAD / HTTP/1.0\r\n\r\n')
                    banner = s.recv(1024).decode(errors='ignore').strip()
                except Exception:
                    banner = ''
                return port, banner
    except Exception:
        pass
    return None

if __name__ == '__main__':
    with concurrent.futures.ThreadPoolExecutor() as executor:
        futures = [executor.submit(probe_port, port) for port in range(START, END + 1)]
        for future in futures:
            port, banner = future.result()
            if port and banner:
                open_ports.append(port)
                print(f"Found open port: {port} with banner: {banner}")
    print(f"Open ports: {open_ports}")
    with open('outputs.txt', 'a') as f:
        f.write(f"{datetime.now()} Open ports: {open_ports}\n")
```

Jayesh R. Yadav

Output :

A terminal window on a Kali Linux system. The window title is 'root@kali: /home/nega'. The terminal shows the execution of a Python script named 'recon.py' in the directory '/home/negan/RedOps/'. The script takes '192.168.1.1' as a target and '20 8000' as a port range. It reports finding 1 open port (443) and saving results to 'outputs/'. Subsequent commands show the user navigating to the 'outputs/' directory and listing the files, which includes 'recon\_192.168.1.1.json'.

```
(root@kali)-[/home/negan/RedOps/recon]
# python3 recon.py 192.168.1.1 20 8000
[+] Recon start: 2025-09-19T10:06:57.356355 target=192.168.1.1 ports=20-8000
[OPEN] 443
[+] Done. Found 1 open ports. Duration: 0:00:40.191184
[+] Results saved to outputs/

(root@kali)-[/home/negan/RedOps/recon]
# ls
recon.py

(root@kali)-[/home/negan/RedOps/recon]
# cd ..

(root@kali)-[/home/negan/RedOps]
# ls
outputs recon

(root@kali)-[/home/negan/RedOps]
# outputs/

(root@kali)-[/home/negan/RedOps/outputs]
# ls
recon_192.168.1.1.json

(root@kali)-[/home/negan/RedOps/outputs]
#
```