A Python-Based Port Scanning Tool

Index

1. Introduction and use cases

The Andromeda Scanner is a Python-based tool designed to scan open ports on a specified host. This tool is intended for cybersecurity professionals and enthusiasts who wish to identify open ports and associated services on a network. The tool includes functionalities for both single-port and multi-port scanning, with an option to use threading for faster performance.

1. Network Security Auditing

- **Objective**: Identify open ports on servers and network devices to ensure that only necessary services are accessible.
- **Scenario**: A cybersecurity team is tasked with auditing the security of a company's internal network. They use the Andromeda Scanner to scan critical servers and identify open ports, helping them to detect potential vulnerabilities, such as unnecessary services that could be exploited by attackers.
- **Outcome**: The team can close or secure unnecessary open ports, reducing the attack surface and improving overall network security.

2. Vulnerability Assessment

- **Objective**: Preemptively identify potential vulnerabilities by scanning for open ports and associated services.
- **Scenario**: A penetration tester is conducting a vulnerability assessment on a client's network. By using the Andromeda Scanner, they can identify all open ports and services running on the target machines. This information is crucial for identifying outdated or vulnerable services that might be targeted for exploitation.
- **Outcome**: The penetration tester can focus their efforts on exploiting known vulnerabilities in the services running on the identified open ports.

3. Incident Response

- **Objective**: Quickly assess the status of a network after a security breach to identify potentially compromised services.
- **Scenario**: Following a security breach, the incident response team needs to assess the current state of the network. The Andromeda Scanner is used to quickly scan key servers and devices to identify which services are currently exposed and might have been exploited by the attackers.
- **Outcome**: The team can prioritize securing or shutting down compromised services to prevent further damage.

4. Compliance Auditing

- **Objective**: Ensure compliance with security standards by verifying that only approved services are running on specified ports.
- **Scenario**: A financial institution must comply with industry regulations that mandate the security of their IT infrastructure. As part of a routine compliance audit, the IT team uses the Andromeda Scanner to verify that only approved services are running on the required ports across all servers.
- **Outcome**: The institution can demonstrate compliance with security standards and avoid potential fines or penalties.

5. Asset Discovery and Inventory

- **Objective**: Discover and inventory all active services on a network.
- **Scenario**: A company wants to perform an inventory of all the devices and services running on its network. By deploying the Andromeda Scanner, the IT team can scan the entire network, identify all active services, and create a detailed inventory that can be used for monitoring and management purposes.
- **Outcome**: The company gains visibility into its IT assets, enabling better management and security of the network.

6. Educational and Training Purposes

- **Objective**: Teach students or new cybersecurity professionals how to use port scanning tools and understand network security concepts.
- **Scenario**: In a cybersecurity training program, instructors use the Andromeda Scanner as a teaching tool to demonstrate how port scanning works. Students use the tool to scan test networks, learn about different services that run on various ports, and understand the importance of securing these services.
- **Outcome**: Students gain hands-on experience with port scanning, enhancing their understanding of network security and the tools used in the field.

7. Preparation for Penetration Testing

- **Objective**: Gather information about a target network before performing a full penetration test.
- **Scenario**: Before conducting a penetration test, a tester needs to perform reconnaissance to understand the target's network structure. The Andromeda Scanner is used to scan the target’s IP addresses, revealing open ports and running services, which helps in planning the penetration test.

- **Outcome**: The tester can develop a more targeted approach to penetration testing, increasing the chances of uncovering security vulnerabilities.

8. Post-Deployment Security Check

- **Objective**: Ensure that a newly deployed system is secure before it goes live.
- **Scenario**: After deploying a new server or service, the IT team uses the Andromeda Scanner to ensure that only the necessary ports are open and that no unexpected services are running. This final check is crucial before the system is made available to users.
- **Outcome**: The new system is deployed with minimal security risks, reducing the likelihood of future security incidents.

2. Features

- **Single Port Scanning**: Allows users to scan a specific port on a host.
- **Range Port Scanning**: Enables scanning of a range of ports sequentially.
- **Threaded Scanning**: Utilizes Python's threading module to perform faster scans across a range of ports.
- **Service Identification**: Identifies the services running on open ports.

3. Technical Specifications

3.1 Programming Language

The tool is written in Python 3, leveraging Python's built-in libraries such as socket, threading, and argparse.

3.2 Dependencies

No external libraries are required, making the tool easy to set up and use on any system with Python 3 installed.

3.3 Command-Line Arguments

- **Host**: The IP address or domain name of the target.
- **Ports**: The range of ports to scan (e.g., 1-1024).
- **Threads**: Optional flag to enable threaded scanning for faster execution.

3.4 Example Usage

python andromeda_scanner.py 192.168.1.1 -p 20-80 -t

This command scans ports 20 to 80 on the host 192.168.1.1 using threading.

4. Implementation Details

4.1 Single Port Scanning

The tool can be used to scan a single port using the scan_single_port() function. This function attempts to connect to the specified port on the host and returns whether the port is open or closed.

4.2 Multi-Port Scanning

The scan_multiple_ports() function scans a range of ports sequentially, collecting a list of open ports.

4.3 Threaded Port Scanning

For improved performance, the scan_with_threads() function performs the scan using multiple threads. Each thread handles the scanning of a specific port, significantly reducing the time required for large ranges.

4.4 Service Identification

The get_service_name() function maps open ports to their standard services, enhancing the usefulness of the scan results by providing additional context.

5. Banner and Branding

The tool includes a custom ASCII banner that gives the Andromeda Scanner a unique identity when executed. This can be customized to fit the user's needs.

6. Usage Instructions

1. **Install Python 3**: Ensure Python 3 is installed on your system.
2. **Download the Script**: Obtain the andromeda_scanner.py script.
3. **Run the Script**: Use the command line to execute the script with the desired options.

4. **Analyze the Results**: Review the output to identify open ports and associated services.

7. License

The Andromeda Scanner is released under the MIT License, allowing users to freely use, modify, and distribute the software.

8. Author

Erick Marín Monge
Cybersecurity Specialist and Developer

9. Appendix

9.1 Complete Code Listing

```
import socket
import threading
import argparse


# Banner corregido
BANNER = r"""
ANDROMEDA SCANNER
"""


# Function to scan single port
def scan_single_port(host, port):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(1)
    try:
        s.connect((host, port))
        print(f"Puerto {port} está abierto ({get_service_name(port)})")
    except socket.error as err:
        print(f"Puerto {port} está cerrado. Error: {err}")
```

```
    finally:
        s.close()

# Function to scan a range of ports
def scan_multiple_ports(host, port_range):
    open_ports = []
    for port in port_range:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(1)
        try:
            s.connect((host, port))
            open_ports.append(port)
        except socket.error:
            pass
        finally:
            s.close()
    return open_ports

# function to scan a range of ports using threads
def scan_with_threads(host, port_range):
    open_ports = []
    threads = []

    def scan_port(host, port):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(1)
        try:
            s.connect((host, port))
            open_ports.append(port)
        except socket.error:
```

```
            pass
        finally:
            s.close()

    for port in port_range:
        t = threading.Thread(target=scan_port, args=(host, port))
        threads.append(t)
        t.start()

    for t in threads:
        t.join()

    return open_ports

# function to obtain the name and service associated with a port
def get_service_name(port):
    try:
        return socket.getservbyport(port)
    except:
        return "Unknown Service"

# function to manage the command line arguments
def parse_arguments():
    parser = argparse.ArgumentParser(description="Escáner de Puertos en Python")
    parser.add_argument("host", help="La dirección IP o nombre de dominio del host a escanear")
    parser.add_argument("-p", "--ports", help="El rango de puertos a escanear, ejemplo: 1-1024", default="1-1024")
    parser.add_argument("-t", "--threads", help="Utilizar hilos para el escaneo", action="store_true")
    return parser.parse_args()
```

```
# main block to execute functions
if __name__ == "__main__":
    print(BANNER)  # Mostrar el banner al inicio

    args = parse_arguments()  # Ahora se llama a la función después de que esté definida

    host = args.host
    port_range = range(int(args.ports.split('-')[0]), int(args.ports.split('-')[1]) + 1)

    if args.threads:
        print("\nEscaneando múltiples puertos con hilos...")
        open_ports = scan_with_threads(host, port_range)
    else:
        print("\nEscaneando múltiples puertos secuencialmente...")
        open_ports = scan_multiple_ports(host, port_range)

    if open_ports:
        print(f"Puertos abiertos en {host}:")
        for port in open_ports:
            print(f"Puerto {port} ({get_service_name(port)})")
    else:
        print("No se encontraron puertos abiertos.")
```

Tool in action example

9.2 Future Enhancements

- Adding support for UDP port scanning.
- Implementing advanced service detection techniques.
- Integrating with logging tools for detailed scan reports.

10. Contact Information

For support or inquiries, please contact Erick Marín Monge at erick.marin-95@protonmail.com