INDUSTRY PROBLEM-1

PORT SCANS AND NETWORK ATTACKS USING SCAPY

What is a Port Scan?

A port scan is a common technique hackers use to discover open doors or weak points in a network. A port scan attack helps cyber criminals find open ports and figure out whether they are receiving or sending data. It can also reveal whether active security devices like firewalls are being used by an organization.
When hackers send a message to a port, the response they receive determines whether the port is being used and if there are any potential weaknesses that could be exploited.
Businesses can also use the port scanning technique to send packets to specific ports and analyse responses for any potential vulnerability. They can then use tools like IP scanning,Scapy, network mapper (Nmap), and Netcat to ensure their network and systems are secure.

Port scanning can provide information such as:

1. Services that are running
2. Users who own services
3. Whether anonymous logins are allowed
4. Which network services require authentication

How to Prevent Port Scan Attacks?

Port scanning is a popular method cyber criminals use to search for vulnerable servers. They often use it to discover organizations' security levels, determine whether businesses have effective firewalls, and detect vulnerable networks or servers. Some TCP methods also enable attackers to hide their location.

Cyber criminals search through networks to assess how ports react, which enables them to understand the business's security levels and the systems they deploy.
Preventing a port scan attack is reliant on having effective, updated that is in line with the evolving threat landscape. Businesses also require strong security software, port scanning tools, and security alerts that monitor ports and prevent malicious actors from reaching their network. Useful tools include IP scanning, Nmap, and Netcat.

What is Scapy?

Scapy is a Python interpreter that enables you to create, forge, or decode packets on the network, to capture packets and analyse them, to dissect the packets, etc. It also allows you to inject packets into the network. It supports a wide number of network protocols and it can handle and manipulate wireless communication packets.

Scapy can be used to perform the jobs done by many network tools, such as nmap, hping, arpscan, and tshark (the command line of wireshark).

1) TCP CONNECT SCAN

DESCRIPTION:

TCP connect scan is a three-way handshake between the client and the server. If the three-way handshake takes place, then communication has been established.

TCP connect scanning commonly involves establishing a full **connection**, and then subsequently tearing it down, and therefore involves sending a significant number of packets to each port that is **scanned**. Compared to other types of **scans**, a **TCP Connect scan** is slow and methodical. An adversary uses full TCP connection attempts to determine if a port is open on the target system. The scanning process involves completing a 'three-way handshake' with a remote port, and reports the port as closed if the full handshake cannot be established. An advantage of TCP connect scanning is that it works against any TCP/IP stack.

EXTENDED DESCRIPTION:

RFC 793 defines how TCP connections are established and torn down. TCP connect scanning commonly involves establishing a full connection, and then subsequently tearing it down, and therefore involves sending a significant number of packets to each port that is scanned. Compared to other types of scans, a TCP Connect scan is slow and methodical. This type of scanning causes considerable noise in system logs and can be spotted by IDS/IPS systems. TCP Connect scanning can detect when a port is open by completing the three-way handshake, but it cannot distinguish a port that is unfiltered with no service running on it from a port that is filtered by a firewall but contains an active service. Due to the significant volume of packets exchanged per port, TCP connect scanning can become very time consuming (performing a full TCP connect scan against a host can take multiple days). Generally, it is not used as a method for performing a comprehensive port scan, but is reserved for checking a short list of common ports.

PREREQUISITES:

The adversary requires logical access to the target network. The TCP connect Scan requires the ability to connect to an available port and complete a 'three-way-handshake' This scanning technique does not require any special privileges in order to perform. This type of scan works against all TCP/IP stack implementations.

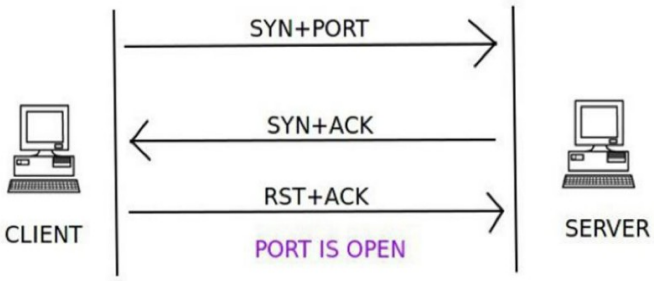
EXECUTION FLOW:

1. An adversary attempts to initialize a TCP connection with the target port.
2. An adversary uses the result of their TCP connection to determine the state of the target port. A successful connection indicates a port is open with a service listening on it while a failed connection indicates the port is not open.

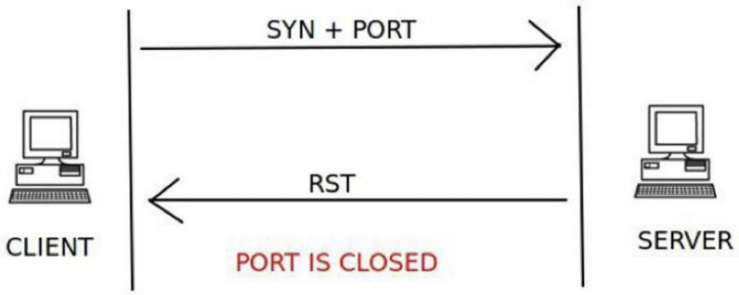
DRAWBACKS:

Exposure of Sensitive Information to an Unauthorized Actor.

STRUCTURE:



A client trying to connect to a server on port 80 initializes the connection by sending a TCP packet with the SYN flag set and the port to which it wants to connect (in this case port 80). If the port is open on the server and is accepting connections, it responds with a TCP packet with the SYN and ACK flags set. The connection is established by the client sending an acknowledgement ACK and RST flag in the final handshake. If this three-way handshake is completed, then the port on the server is open.



The client sends the first handshake using the SYN flag and port to connect to the server in a TCP packet. If the server responds with a RST instead of a SYN-ACK, then that particular port is closed on the server.

SCRIPT:

```
import socket,sys,os
host = 'scanme.nmap.org'
ip = socket.gethostbyname(host)
open_ports =[]
start_port = 79
end_port = 82

def probe_port(host, port, result = 1):
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(0.5)
        r = sock.connect_ex((host, port))
        if r == 0:
            result = r
        sock.close()
    except Exception as e:
        pass

    return result

for p in range(start_port, end_port+1):
    sys.stdout.flush()
    print (p)
    response = probe_port(host, p)
    if response == 0:
        open_ports.append(p)
    if not p == end_port:
        sys.stdout.write('\b' * len(str(p)))

if open_ports:
    print ("Open Ports")
    print (sorted(open_ports))
else:
    print ("Sorry, No open ports found.!!")
```

OUTPUT:

```
C:\Users\91961\Desktop\scapy-master>connect.py
79
80
81
82
Open Ports
[80]

C:\Users\91961\Desktop\scapy-master>
```

2) TCP STEALTH SCAN

DESCRIPTION:

TCP stealth scan is similar to the TCP connect scan. The client sends a TCP packet with the SYN flag set and the port number to connect to. If the port is open, the server responds with the SYN and ACK flags inside a TCP packet. But this time the client sends a RST flag in a TCP packet and not RST+ACK, which was the case in the TCP connect scan. This technique is used to avoid port scanning detection by firewalls.

An adversary uses a SYN scan to determine the status of ports on the remote target. SYN scanning is the most common type of port scanning that is used because of its many advantages and few drawbacks. As a result, novice attackers tend to overly rely on the SYN scan while performing system reconnaissance. As a scanning method, the primary advantages of SYN scanning are its universality and speed.

EXTENDED DESCRIPTION:

RFC 793 defines the required behaviour of any TCP/IP device in that an incoming connection request begins with a SYN packet, which in turn must be followed by a SYN/ACK packet from the receiving service. For this reason, like TCP Connect scanning, SYN scanning works against any TCP stack. Unlike TCP Connect scanning, it is possible to scan thousands of ports per second using this method. This type of scanning is usually referred to as 'half-open' scanning because it does not complete the three-way handshake. The scanning rate is extremely fast because no time is wasted completing the handshake or tearing down the connection. This technique allows an attacker to scan through stateful firewalls due to the common configuration that TCP SYN segments for a new connection will be allowed for almost any port. TCP SYN scanning can also immediately detect 3 of the 4 important types of port status: open, closed, and filtered.

PREREQUISITES:

This scan type is not possible with some operating systems (Windows XP SP 2). On Linux and Unix systems it requires root privileges to use raw sockets.

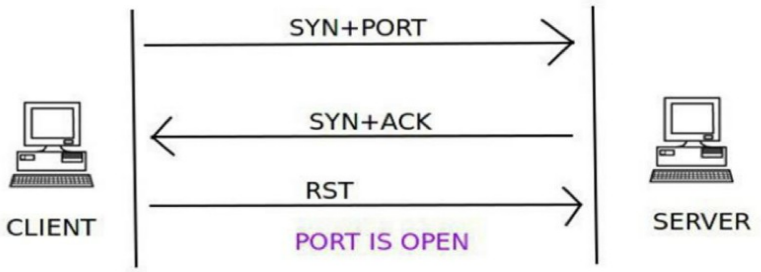
EXECUTION FLOW:

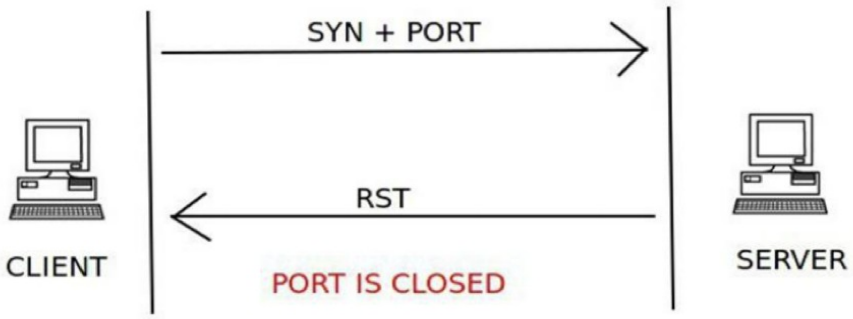
1. An adversary sends SYN packets to ports they want to scan and checks the response without completing the TCP handshake.
2. An adversary uses the response from the target to determine the port's state. The adversary can determine the state of a port based on the following responses. When a SYN is sent to an open port and unfiltered port, a SYN/ACK will be generated. When a SYN packet is sent to a closed port a RST is generated, indicating the port is closed. When SYN scanning to a particular port generates no response, or when the request triggers ICMP Type 3 unreachable errors, the port is filtered.

DRAWBACKS:

Exposure of Sensitive Information to an Unauthorized Actor.

STRUCTURE:


SYN+PORT
SYN+ACK
RST
CLIENT
SERVER
PORT IS OPEN


SYN + PORT
RST
CLIENT
SERVER
PORT IS CLOSED

SCRIPT:

```
import socket
from scapy.all import *

host = 'scanme.nmap.org'
ip = socket.gethostbyname(host)

openp = []
filterdp = []
common_ports = {79,80,81,82}
def is_up(ip):
    icmp = IP(dst=ip)/ICMP()
    resp = sr1(icmp, timeout=10)
    if resp == None:
        return False
    else:
        return True

def probe_port(ip, port, result = 1):
    src_port = RandShort()
    try:
        p = IP(dst=ip)/TCP(sport=src_port, dport=port, flags='S')
        resp = sr1(p, timeout=2) # Sending packet
        if str(type(resp)) == "<type 'NoneType'>":
            result = 0
        elif resp.haslayer(TCP):
            if resp.getlayer(TCP).flags == 0x12:
                send_rst = sr(IP(dst=ip)/TCP(sport=src_port,
dport=port, flags='AR'), timeout=1)
                result = 1
            elif resp.getlayer(TCP).flags == 0x14:
                result = 0
            elif (int(resp.getlayer(ICMP).type)==3 and
int(resp.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                result = 2
    except Exception as e:
        pass

    return result


if __name__ == '__main__':
    conf.verb = 0
    if is_up(ip):
        for port in common_ports:
            print (port)
            response = probe_port(ip, port)
            if response == 1:
                openp.append(port)

        if len(openp) != 0:
            print ("Open Ports:")
            print (openp)
        else:
            print ("Sorry, No open ports found.!!")

        if len(filterdp) != 0:
            print ("Possible Filtered Ports:")
            print (filterdp)
    else:
        print ("Host is Down")
```

OUTPUT:

```
C:\Users\91961\Desktop\scapy-master>stealth.py
80
81
82
79
Open Ports:
[80]

C:\Users\91961\Desktop\scapy-master>
```

3) **TCP XMAS scan**

Description:
An adversary uses a TCP XMAS scan to determine if ports are closed on the target machine. This scan type is accomplished by sending TCP segments with all possible flags set in the packet header, generating packets that are illegal based on RFC 793. The RFC 793 expected behaviour is that any TCP segment with an out-of-state Flag sent to an open port is discarded, whereas segments with out-of-state flags sent to closed ports should be handled with a RST in response. This behaviour should allow an attacker to scan for closed ports by sending certain types of rule-breaking packets (out of sync or disallowed by the TCB) and detect closed ports via RST packets.

Extended Description:
In addition to its relative speed when compared with other types of scans, its major advantage is its ability to scan through stateless firewall or ACL filters. Such filters are configured to block access to ports usually by preventing SYN packets, thus stopping any attempt to 'build' a connection. XMAS packets, like out-of-state FIN or ACK packets, tend to pass through such devices undetected. Because open ports are inferred via no responses being generated, one cannot distinguish an open port from a filtered port without further analysis. For instance, XMAS scanning a system protected by a stateful firewall may indicate all ports being open. Because of their obvious rule-breaking nature, XMAS scans are flagged by almost all intrusion prevention or intrusion detection systems.

Typical Severity:
Severity is Low.

Domain of Attack:
- Software
- Communication

Mechanism of Attack:
- Collect and Analyse information.

Resources Required:
This attack can be carried out with a network mapper or scanner, or via raw socket programming in a scripting language. Packet injection tools are also useful for this purpose. Depending upon the method used it may be necessary to sniff the network in order to see the response.

Execution Flow:
1. An adversary sends TCP packets with all flags set but not associated with an existing connection to target ports.
2. An adversary uses the response from the target to determine the port's state. If no response is received the port is open. If a RST packet is received then the port is closed.

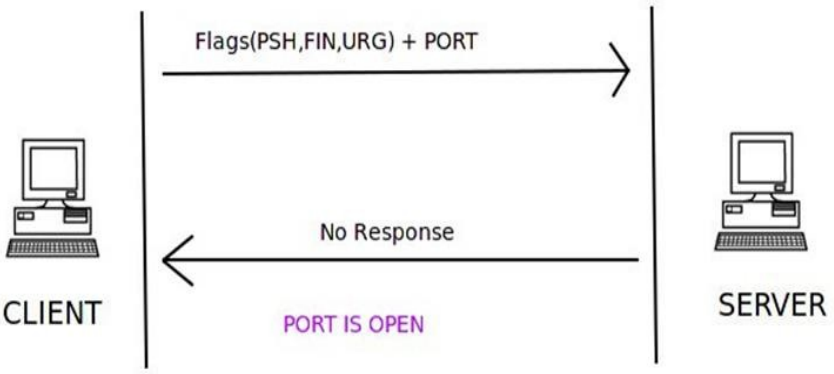
Weakness:

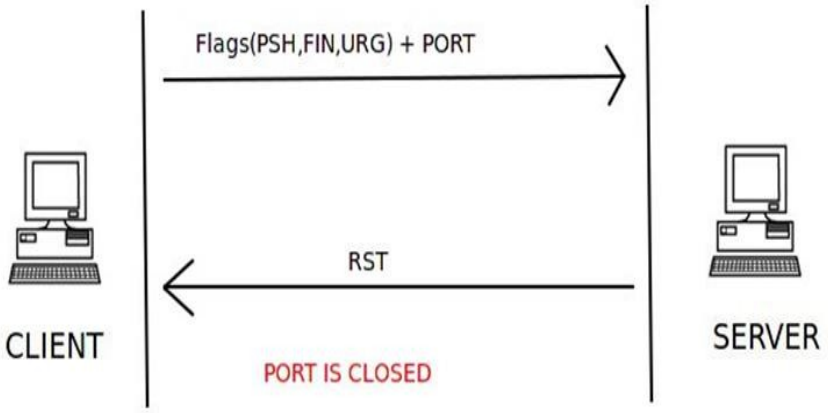
Exposure of Sensitive Information to an Unauthorized Actor

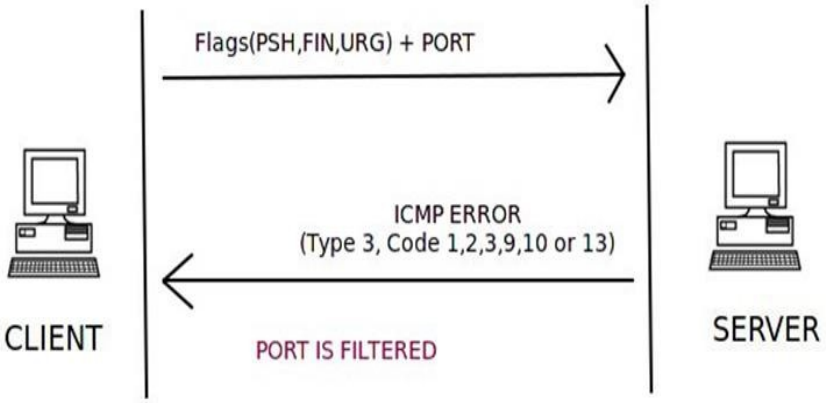
Consequences:

Scope	**Impact**
Confidentiality	Other
Confidentiality access control authorization	Bypass protection Mechanism Hide Activities
Availability	Unreliable Execution

Structure:


Flags(PSH,FIN,URG) + PORT
No Response
CLIENT
PORT IS OPEN
SERVER


Flags(PSH,FIN,URG) + PORT
RST
CLIENT
PORT IS CLOSED
SERVER


Flags(PSH,FIN,URG) + PORT
ICMP ERROR
(Type 3, Code 1,2,3,9,10 or 13)
CLIENT
PORT IS FILTERED
SERVER

Script:

xmas.py

```python
from scapy.all import *

host = 'http://scanme.nmap.org'
ip = socket.gethostbyname(host)

openp = []
filterdp = []
common_ports = { 79,80,81,82
                }
def is_up(ip):
    icmp = IP(dst=ip)/ICMP()
    resp = sr1(icmp, timeout=10)
    if resp == None:
        return False
    else:
        return True

def probe_port(ip, port, result = 1):
    src_port = RandShort()
    try:
        p = IP(dst=ip)/TCP(sport=src_port, dport=port, flags='FPU')
        resp = sr1(p, timeout=2) # Sending packet
        if str(type(resp)) == "<type 'NoneType'>":
            result = 1
        elif resp.haslayer(TCP):
            if resp.getlayer(TCP).flags == 0x14:
                result = 0
            elif (int(resp.getlayer(ICMP).type)==3 and int(resp.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                result = 2

    except Exception as e:
        pass

    return result


if __name__ == '__main__':
    conf.verb = 0
    if is_up(ip):
        for port in common_ports:
            print (port)
            response = probe_port(ip, port)
            if response == 1:
                openp.append(port)
            elif response == 2:
                filterdp.append(port)

        if len(openp) != 0:
            print ("Possible Open or Filtered Ports:")
            print (openp)
        if len(filterdp) != 0:
            print ("Possible Filtered Ports:")
            print (filterdp)
        if (len(openp) == 0) and (len(filterdp) == 0):
            print ("Sorry, No open ports found.!!")
    else:
        print ("Host is Down")
```

Output:

```
PORT — -zsh — 80×24
Last login: Mon May  2 16:19:45 on ttys000
adityasundarraj@Adityas-MBP PORT % Python3 xmas.py
WARNING: No IPv4 address found on anpi0 !
WARNING: No IPv4 address found on anpi2 !
WARNING: more No IPv4 address found on anpi1 !
80
81
82
79
Possible Open or Filtered Ports:
[80, 81, 82, 79]
adityasundarraj@Adityas-MBP PORT %
```

4) **TCP FIN scan**

Description:
An adversary uses a TCP FIN scan to determine if ports are closed on the target machine. This scan type is accomplished by sending TCP segments with the FIN bit set in the packet header. The RFC 793 expected behaviour is that any TCP segment with an out-of-state Flag sent to an open port is discarded, whereas segments with out-of-state flags sent to closed ports should be handled with a RST in response. This behaviour should allow the adversary to scan for closed ports by sending certain types of rule-breaking packets (out of sync or disallowed by the TCB) and detect closed ports via RST packets.

Extended Description:
In addition to its relative speed in comparison with other types of scans, the major advantage a TCP FIN Scan is its ability to scan through stateless firewall or ACL filters. Such filters are configured to block access to ports usually by preventing SYN packets, thus stopping any attempt to 'build' a connection. FIN packets, like out-of-state ACK packets, tend to pass through such devices undetected. FIN scanning is still relatively stealthy as the packets tend to blend in with the background noise on a network link.

Typical Severity:
Severity is Low.

Domain of Attack:
- Software
- Communication

Mechanism of Attack:
- Collect and Analyse information.

Resources Required:
This attack pattern requires the ability to send TCP FIN segments to a host during network reconnaissance. This can be achieved via the use of a network mapper or scanner, or via raw socket programming in a scripting language. Packet injection tools are also useful for this purpose. Depending upon the method used it may be necessary to sniff the network in order to see the response.

Execution Flow:
1. An adversary sends TCP packets with the FIN flag but not associated with an existing connection to target ports.
2. An adversary uses the response from the target to determine the port's state. If no response is received the port is open. If a RST packet is received then the port is closed.

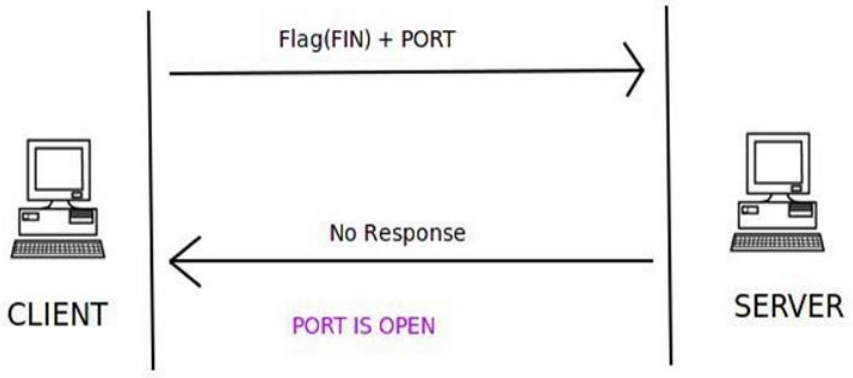
Weakness:

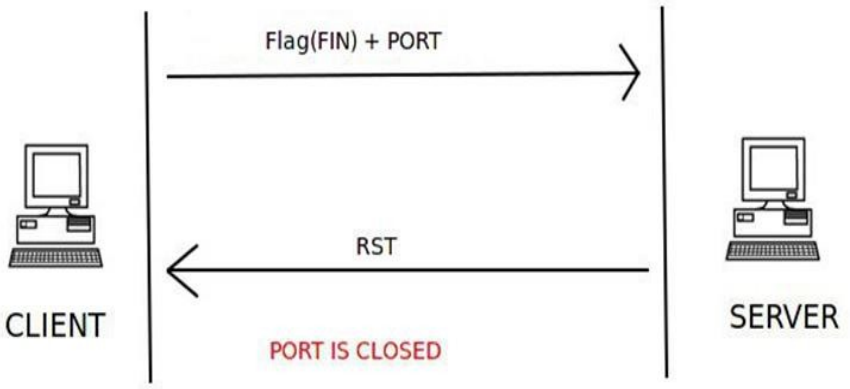
Exposure of Sensitive Information to an Unauthorized Actor

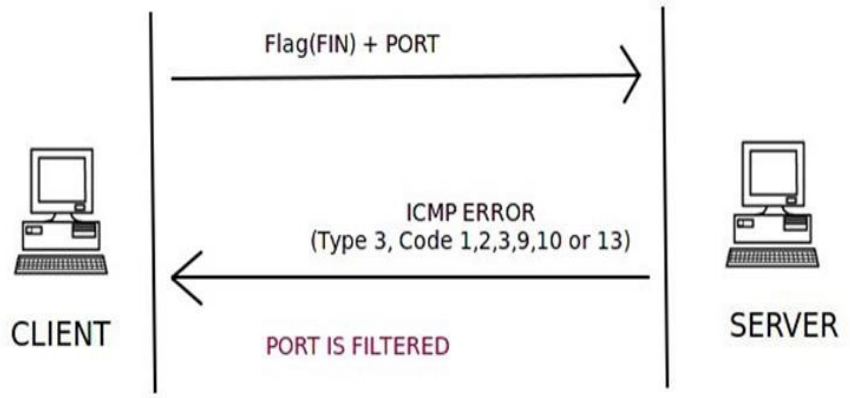
Consequences:

Scope	**Impact**
Confidentiality	Other
Confidentiality access control authorization	Bypass protection Mechanism Hide Activities

Structure:


Flag(FIN) + PORT
No Response
CLIENT
PORT IS OPEN
SERVER


Flag(FIN) + PORT
RST
CLIENT
PORT IS CLOSED
SERVER


Flag(FIN) + PORT
ICMP ERROR
(Type 3, Code 1,2,3,9,10 or 13)
CLIENT
PORT IS FILTERED
SERVER

Script:

fin.py — Free Mode

~/Desktop/PYTHON/PORT/fin.py — (no function selected)

```python
from scapy.all import *

host = 'http://scanme.nmap.org'
ip = socket.gethostbyname(host)

openp = []
filterdp = []
common_ports = { 79,80,81,82
                 }
def is_up(ip):
    icmp = IP(dst=ip)/ICMP()
    resp = sr1(icmp, timeout=10)
    if resp == None:
        return False
    else:
        return True

def probe_port(ip, port, result = 1):
    src_port = RandShort()
    try:
        p = IP(dst=ip)/TCP(sport=src_port, dport=port, flags='F')
        resp = sr1(p, timeout=2) # Sending packet
        if str(type(resp)) == "<type 'NoneType'>":
            result = 1
        elif resp.haslayer(TCP):
            if resp.getlayer(TCP).flags == 0x14:
                result = 0
            elif (int(resp.getlayer(ICMP).type)==3 and int(resp.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                result = 2

    except Exception as e:
        pass

    return result


if __name__ == '__main__':
    conf.verb = 0
    if is_up(ip):
        for port in common_ports:
            print (port)
            response = probe_port(ip, port)
            if response == 1:
                openp.append(port)
            elif response == 2:
                filterdp.append(port)

        if len(openp) != 0:
            print ("Possible Open or Filtered Ports:")
            print (openp)
        if len(filterdp) != 0:
            print ("Possible Filtered Ports:")
            print (filterdp)
        if (len(openp) == 0) and (len(filterdp) == 0):
            print ("Sorry, No open ports found.!!")
    else:
        print ("Host is Down")
```

L: 8 C: 29 Python Unicode (UTF-8) Unix (LF) Saved: 24/04/22, 4:44:36 PM 1,548 / 178 / 59 100%

Output:

```
[adityasundarraj@Adityas-MBP PORT % Python3 fin.py
WARNING: No IPv4 address found on anpi0 !
WARNING: No IPv4 address found on anpi2 !
WARNING: more No IPv4 address found on anpi1 !
80
81
82
79
Possible Open or Filtered Ports:
[80, 81, 82, 79]
adityasundarraj@Adityas-MBP PORT %
```

5) **TCP ACK scan**

Description:
An adversary uses TCP ACK segments to gather information about firewall or ACL configuration. The purpose of this type of scan is to discover information about filter configurations rather than port state. This type of scanning is rarely useful alone, but when combined with SYN scanning, gives a more complete picture of the type of firewall rules that are present.

Extended Description:
When a TCP ACK segment is sent to a closed port, or sent out-of-sync to a listening port, the RFC 793 expected behaviour is for the device to respond with a RST. Getting RSTs back in response to a ACK scan gives the attacker useful information that can be used to infer the type of firewall present. Stateful firewalls will discard out-of-sync ACK packets, leading to no response. When this occurs the port is marked as filtered. When RSTs are received in response, the ports are marked as unfiltered, as the ACK packets solicited the expected behaviour from a port. When combined with SYN techniques an attacker can gain a more complete picture of which types of packets get through to a host and thereby map out its firewall rule-set. ACK scanning, when combined with SYN scanning, also allows the adversary to analyse whether a firewall is stateful or non-stateful (described in notes). TCP ACK Scans are somewhat faster and more stealthy than other types of scans but often requires rather sophisticated analysis by an experienced person. A skilled adversary may use this method to map out firewall rules, but the results of ACK scanning will be less useful to a novice.

Typical Severity:
Severity is Low.

Domain of Attack:
- Software
- Communication

Mechanism of Attack:
- Collect and Analyse information.

Resources Required:
This attack can be achieved via the use of a network mapper or scanner, or via raw socket programming in a scripting language. Packet injection tools are also useful for this purpose. Depending upon the method used it may be necessary to sniff the network in order to see the response.

Execution Flow:
1. An adversary sends TCP packets with the ACK flag set and that are not associated with an existing connection to target ports.
2. An adversary uses the response from the target to determine the port's state. If a RST packet is received the target port is either closed or the ACK was sent out-of-sync. If no response is received, the target is likely using a stateful firewall.

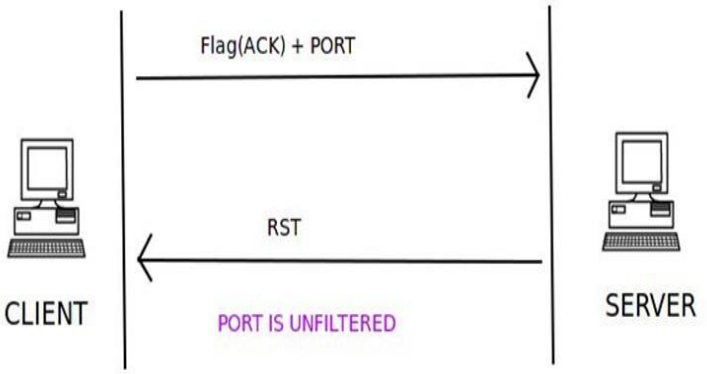
Weakness:

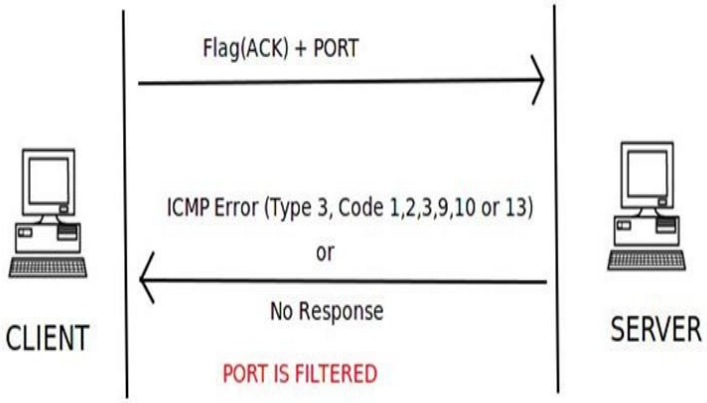
Exposure of Sensitive Information to an Unauthorized Actor

Consequences:

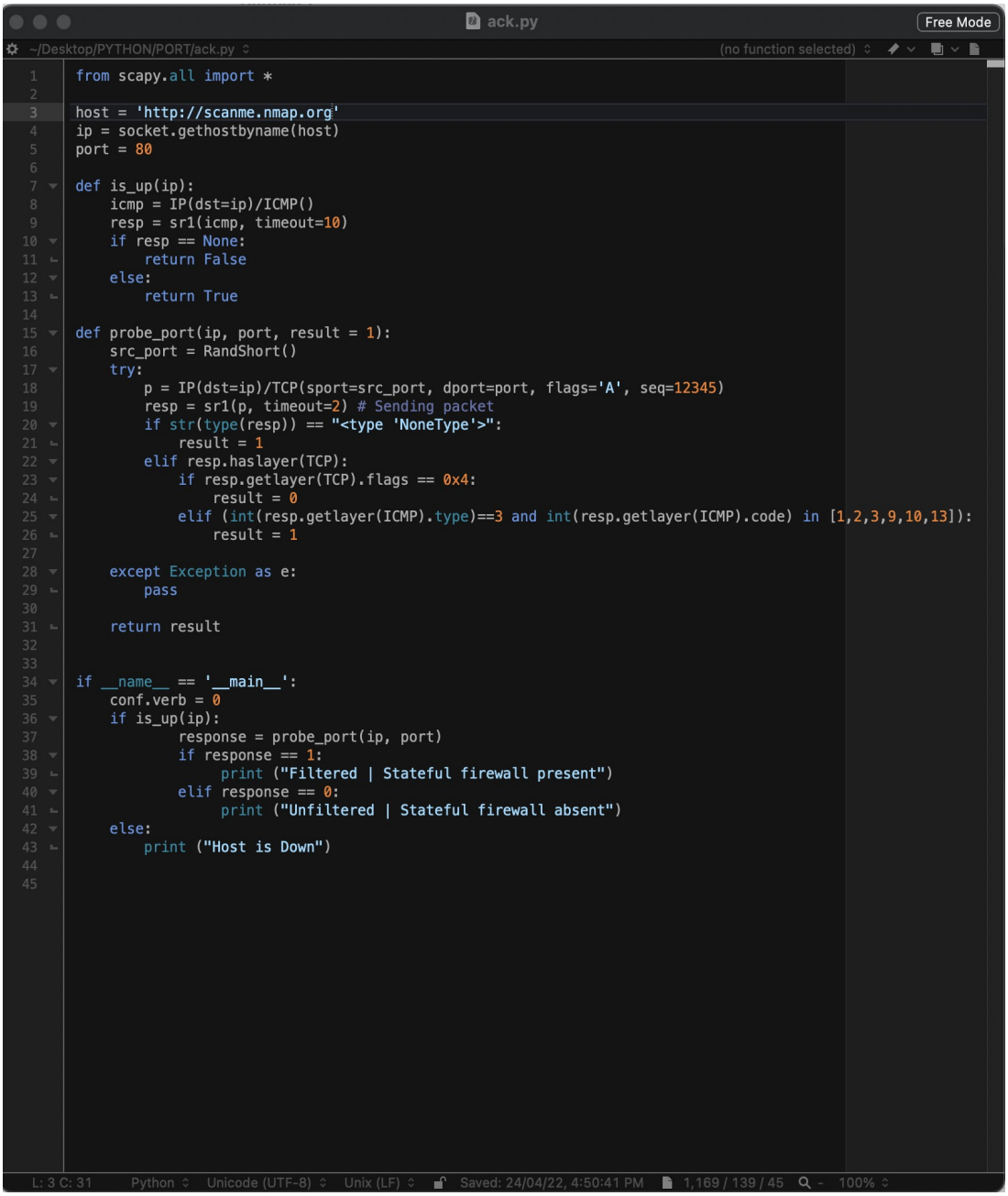
Scope	**Impact**
Confidentiality	Other
Confidentiality access control authorization	Bypass protection Mechanism Hide Activities

Structure:

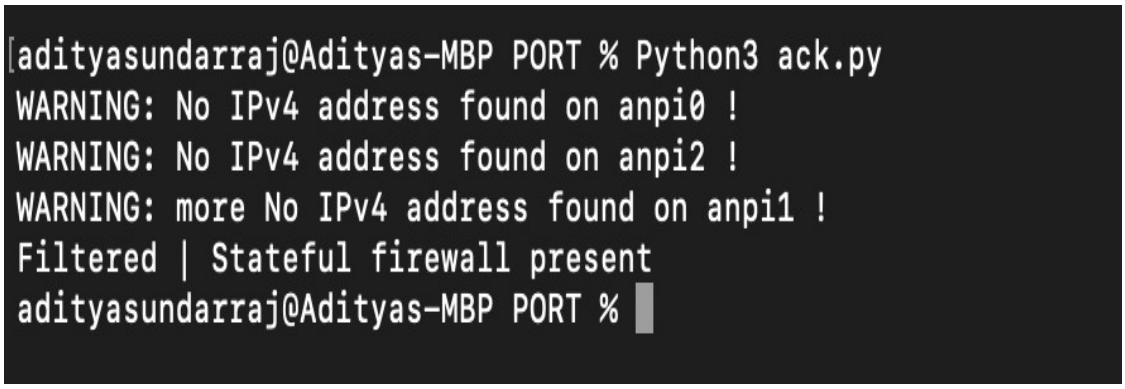

Flag(ACK) + PORT
RST
CLIENT
PORT IS UNFILTERED
SERVER


Flag(ACK) + PORT
ICMP Error (Type 3, Code 1,2,3,9,10 or 13)
or
No Response
CLIENT
PORT IS FILTERED
SERVER

Script:



Output:



6) NULL scan:

DESCRIPTION:

An adversary uses a TCP NULL scan **to determine if ports are closed on the target machine**. This scan type is accomplished by sending TCP segments with no flags in the packet header, generating packets that are illegal based on RFC 793.In a NULL scan, no flag is set inside the TCP packet. The TCP packet is sent along with the port number only to the server. If the server sends no response to the NULL scan packet, then that particular port is open An adversary uses a TCP NULL scan to determine if ports are closed on the target machine. This scan type is accomplished by sending TCP segments with no flags in the packet header, generating packets that are illegal based on RFC 793. The RFC 793 expected behaviour is that any TCP segment with an out-of-state Flag sent to an open port is discarded, whereas segments with out-of-state flags sent to closed ports should be handled with a RST in response. This behaviour should allow an attacker to scan for closed ports by sending certain types of rule-breaking packets (out of sync or disallowed by the TCB) and detect closed ports via RST packets.

Extended Description:

In addition to being fast, the major advantage of this scan type is its ability to scan through stateless firewall or ACL filters. Such filters are configured to block access to ports usually by preventing SYN packets, thus stopping any attempt to 'build' a connection. NULL packets, like out-of-state FIN or ACK packets, tend to pass through such devices undetected. Additionally, because open ports are inferred via no responses being generated, one cannot distinguish an open port from a filtered port without further analysis. For instance, NULL scanning a system protected by a stateful firewall may indicate all ports being open. Because of their obvious rule-breaking nature, NULL scans are flagged by almost all intrusion prevention or intrusion detection systems

DRAWBACKS:

Exposure of Sensitive Information to an Unauthorized Actor

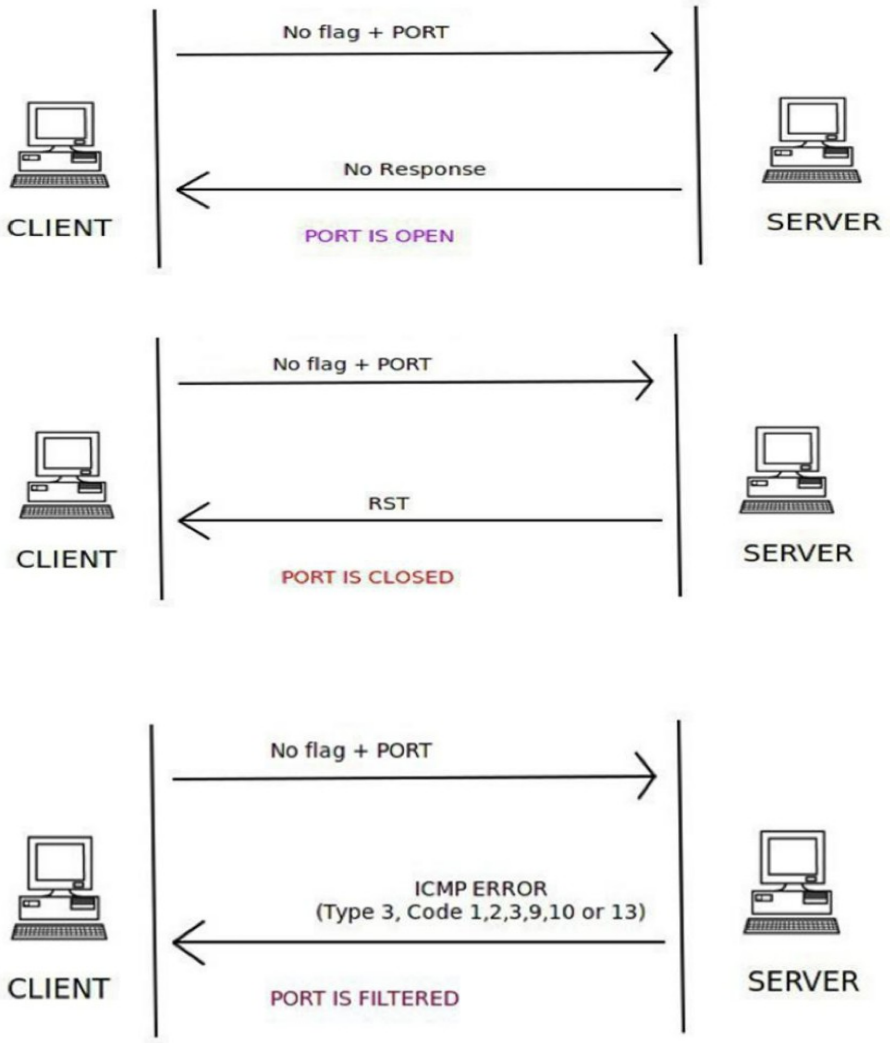
EXECUTION FLOW:

- An adversary sends TCP packets with no flags set and that are not associated with an existing connection to target ports.
- An adversary uses the response from the target to determine the port's state. If no response is received the port is open. If a RST packet is received then the port is closed.

CONSEQUENCES:

Scope	Impact
Confidentiality	Other
Confidentiality Access Control Authorization	Bypass Protection Mechanism Hide Activities

STRUCTURE:


No flag + PORT
No Response
CLIENT
SERVER
PORT IS OPEN
No flag + PORT
RST
CLIENT
SERVER
PORT IS CLOSED
No flag + PORT
ICMP ERROR
(Type 3, Code 1,2,3,9,10 or 13)
CLIENT
SERVER
PORT IS FILTERED

SCRIPT:

null.py

C: > Users > Dell > Downloads > null.py

```python
from scapy.all import *

host = 'http://scanme.nmap.org'
ip = socket.gethostbyname(host)

openp = []
filterdp = []
common_ports = { 79,80,81,82
                 }
def is_up(ip):
    icmp = IP(dst=ip)/ICMP()
    resp = sr1(icmp, timeout=10)
    if resp == None:
        return False
    else:
        return True

def probe_port(ip, port, result = 1):
    src_port = RandShort()
    try:
        p = IP(dst=ip)/TCP(sport=src_port, dport=port, flags='',timeout=10)
        resp = sr1(p, timeout=2) # Sending packet
        if str(type(resp)) == "<type 'NoneType'>":
            result = 1
        elif resp.haslayer(TCP):
            if resp.getlayer(TCP).flags == 0x14:
                result = 0
            elif (int(resp.getlayer(ICMP).type)==3 and int(resp.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                result = 2

    except Exception as e:
        pass

    return result
```

```python
if __name__ == '__main__':
    conf.verb = 0
    if is_up(ip):
        for port in common_ports:
            print (port)
            response = probe_port(ip, port)
            if response == 1:
                openp.append(port)
            elif response == 2:
                filterdp.append(port)

        if len(openp) != 0:
            print ("Possible Open or Filtered Ports:")
            print (openp)
        if len(filterdp) != 0:
            print ("Possible Filtered Ports:")
            print (filterdp)
        if (len(openp) == 0) and (len(filterdp) == 0):
            print ("Sorry, No open ports found.!!")
    else:
        print ("Host is Down")
```

OUTPUT:

```
Last login: Mon May  2 19:13:43 on console
adityasundarraj@Adityas-MBP PORT % Python3 null.py
WARNING: No IPv4 address found on anpi0 !
WARNING: No IPv4 address found on anpi1 !
WARNING: more No IPv4 address found on anpi2 !
80
81
82
79
Possible Open or Filtered Ports:
[80, 81, 82, 79]
adityasundarraj@Adityas-MBP PORT %
```

7) UDP scan:

DESCRIPTION:

UDP scanning methods involve **sending a UDP datagram to the target port and looking for evidence that the port is closed**. Open UDP ports usually do not respond to UDP datagrams as there is no stateful mechanism within the protocol that requires building or establishing a session.

An adversary engages in UDP scanning to gather information about UDP port status on the target system. UDP scanning methods involve sending a UDP datagram to the target port and looking for evidence that the port is closed. Open UDP ports usually do not respond to UDP datagrams as there is no stateful mechanism within the protocol that requires building or establishing a session. Responses to UDP datagrams are therefore application specific and cannot be relied upon as a method of detecting an open port. UDP scanning relies heavily upon ICMP diagnostic messages in order to determine the status of a remote port.

Extended Description:

During a UDP scan, a datagram is sent to a target port. If an 'ICMP Type 3 Port unreachable' error message is returned then the port is considered closed. Different types of ICMP messages can indicate a filtered port. UDP scanning is slower than TCP scanning. The protocol characteristics of UDP make port scanning inherently more difficult than with TCP, as well as dependent upon ICMP for accurate scanning. Due to ambiguities that can arise between open ports and filtered ports, UDP scanning results often require a high degree of interpretation and further testing to refine. In general, UDP scanning results are less reliable or accurate than TCP-based scanning.

PREREQUISITES:

The ability to send UDP datagrams to a host and receive ICMP error messages from that host. In cases where particular types of ICMP messaging is disallowed, the reliability of UDP scanning drops off sharply.

UDP port scan attack:

A port scan is **a common technique hackers use to discover open doors or weak points in a network**. A port scan attack helps cyber criminals find open ports and figure out whether they are receiving or sending data. It can also reveal whether active security devices like firewalls are being used by an organization.

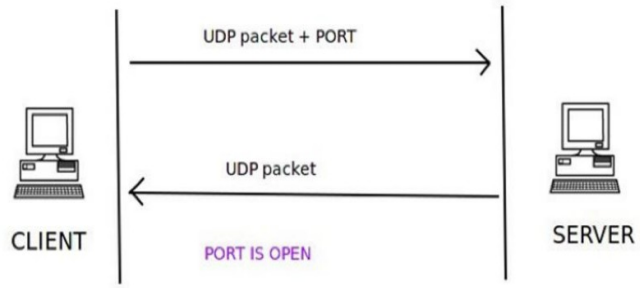
EXECUTION FLOW:

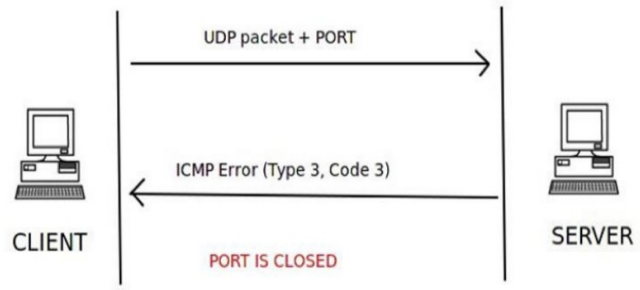
- An adversary sends UDP packets to target ports.
- An adversary uses the response from the target to determine the port's state. Whether a port responds to a UDP packet is dependent on what application is listening on that port. No response does not indicate the port is not open.

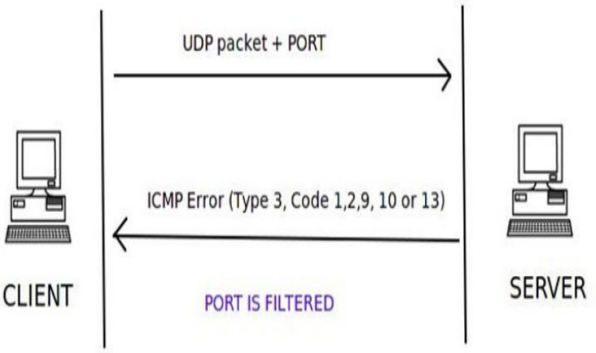
DRAWBACKS:

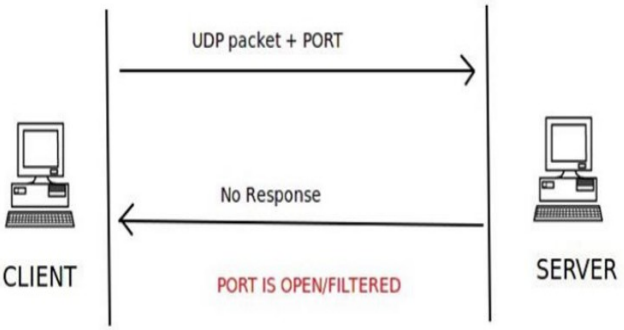
- The major drawback of UDP scan is **the scan is slow**.
- Since there is no response from the open port, the scanner has to resent the packet multiple times leading to the delay.

STRUCTURE:


UDP packet + PORT
UDP packet
CLIENT
SERVER
PORT IS OPEN


UDP packet + PORT
ICMP Error (Type 3, Code 3)
CLIENT
SERVER
PORT IS CLOSED


UDP packet + PORT
ICMP Error (Type 3, Code 1,2,9, 10 or 13)
CLIENT
SERVER
PORT IS FILTERED


UDP packet + PORT
No Response
CLIENT
SERVER
PORT IS OPEN/FILTERED

SCRIPT:

udp.py

```python
import sys
from scapy.all import *
destip="45.33.32.156"

a=79
b=82
ports=range(a,b)


openport=[]
closeport=[]
filterport=[]

for i in ports:
    p=sr1(IP(dst=destip)/UDP(sport=RandShort(), dport=i), flag= "",timeout=10)
    if(str(type(p))== "<type 'Nonetype'>"):
        filterport.append(i)
        continue
    if(p.haslayer(UDP)):
        openport.append(i)
    elif(p.haslayer(ICMP)):
        ic=p.getlayer(ICMP)
        if(int(ic.type == 3) and int(ic.code == 3)):
            closeport.append(i)

print ("Open ports in the given range are: ", openport)
print ("Closed ports in the given range are: ", closeport)
print ("Filtered ports in the given range are: ", filterport)
```

OUTPUT:

```
Last login: Mon May  2 19:17:47 on ttys000
adityasundarraj@Adityas-MBP PORT % Python3 udp.py
WARNING: No IPv4 address found on anpi0 !
WARNING: No IPv4 address found on anpi1 !
WARNING: more No IPv4 address found on anpi2 !
Begin emission:
Finished sending 1 packets.
...*
Received 4 packets, got 1 answers, remaining 0 packets
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Open ports in the given range are:  []
Closed ports in the given range are:  []
Filtered ports in the given range are:  [79, 80, 81]
adityasundarraj@Adityas-MBP PORT %
```