Randomización de puertos TCP

Fernando Gont

UTN/FRH, Argentina

Jornada de Seguridad en Internet

15 de agosto de 2007, Montevideo, Uruguay

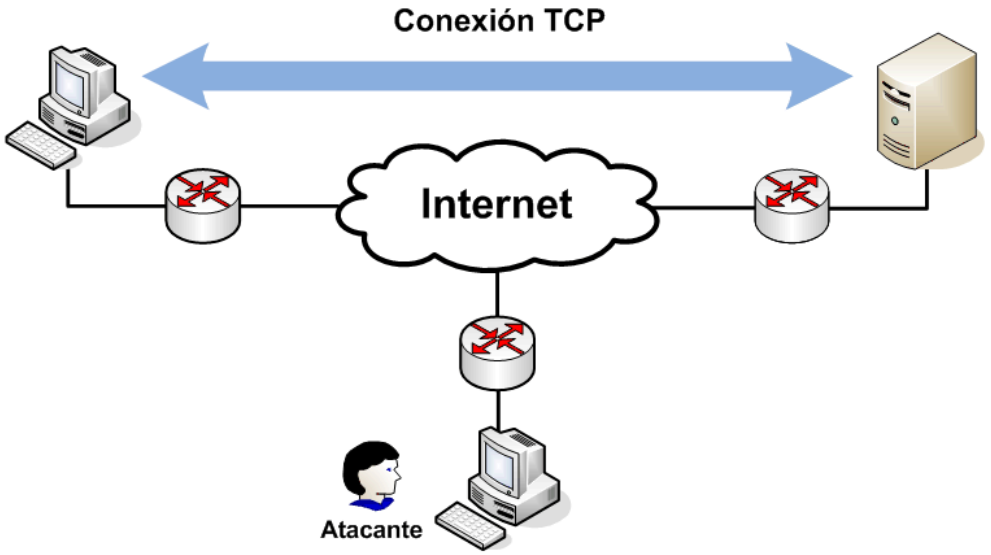
ANTEL TELECOMUNICACIONES

CSIRT Gestión de Incidentes de Seguridad Informática y de Telecomunicaciones

INTELIGENCIA EN TELECOMUNICACIONES

Ataques contra TCP

- En los últimos años se han divulgado dos familias de ataques “ciegos” contra TCP
  - “**Slipping in the Window**”: Ataques basados en segmentos TCP falsificados (NISCC vulnerability advisory #236929)
  - “**ICMP attacks against TCP**” : Ataques basados en paquetes ICMP falsificados (NISCC vulnerability advisory #532967)
- Estos ataques pueden ser realizados sin la necesidad de acceder a los paquetes pertenecientes a la conexión atacada.



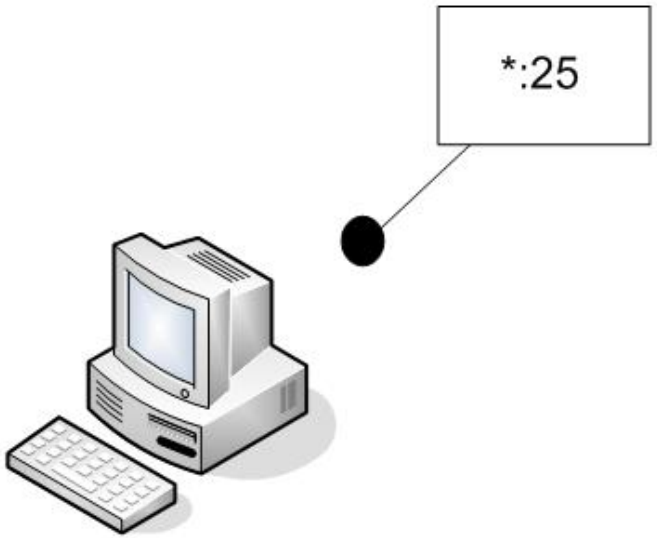
- Estos ataques requieren, mínimamente, que el atacante conozca o pueda adivinar el connection-id.

Selección de puertos efímeros

Utilización de los números de puerto

- Para poder permitir que varias instancias de comunicación utilicen los servicios de TCP al mismo tiempo, TCP posee “puertos”, encargados de posibilitar la demultiplexación de segmentos.
- Toda aplicación que desee poder recibir conexiones, deberá quedarse escuchando en algún puerto particular.

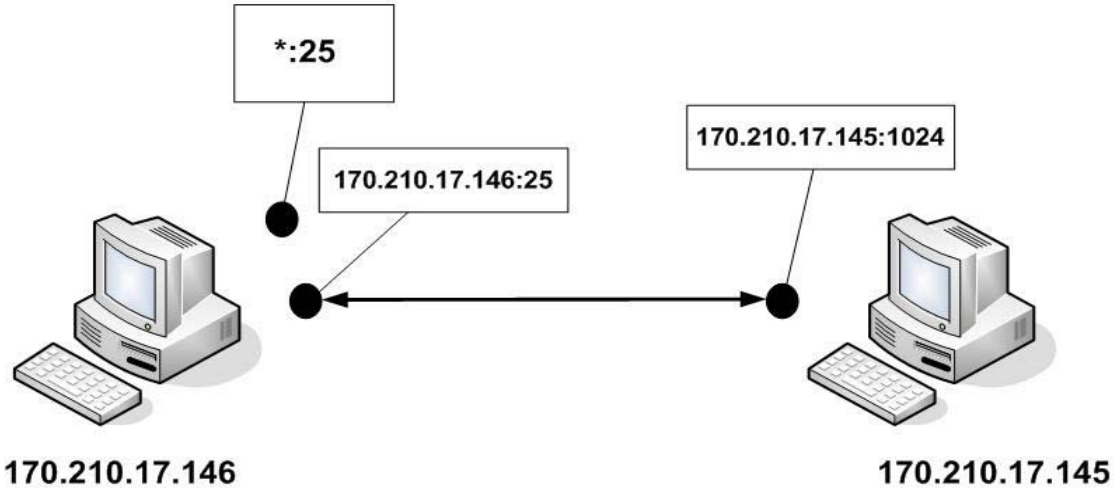
En el siguiente ejemplo, el sistema en cuestión se encuentra escuchando en el puerto 25:



Cada punto de conexion se denomina “socket”

Utilización de los números de puerto *(cont.)*

Si un sistema deseara conectarse al servidor de nuestro ejemplo, debería utilizar como “destiantion port” el número de puerto en el cual el servidor se encuentra escuchando, y utilizar como “source port” algún número de puerto que no estuviera utilizado actualmente.



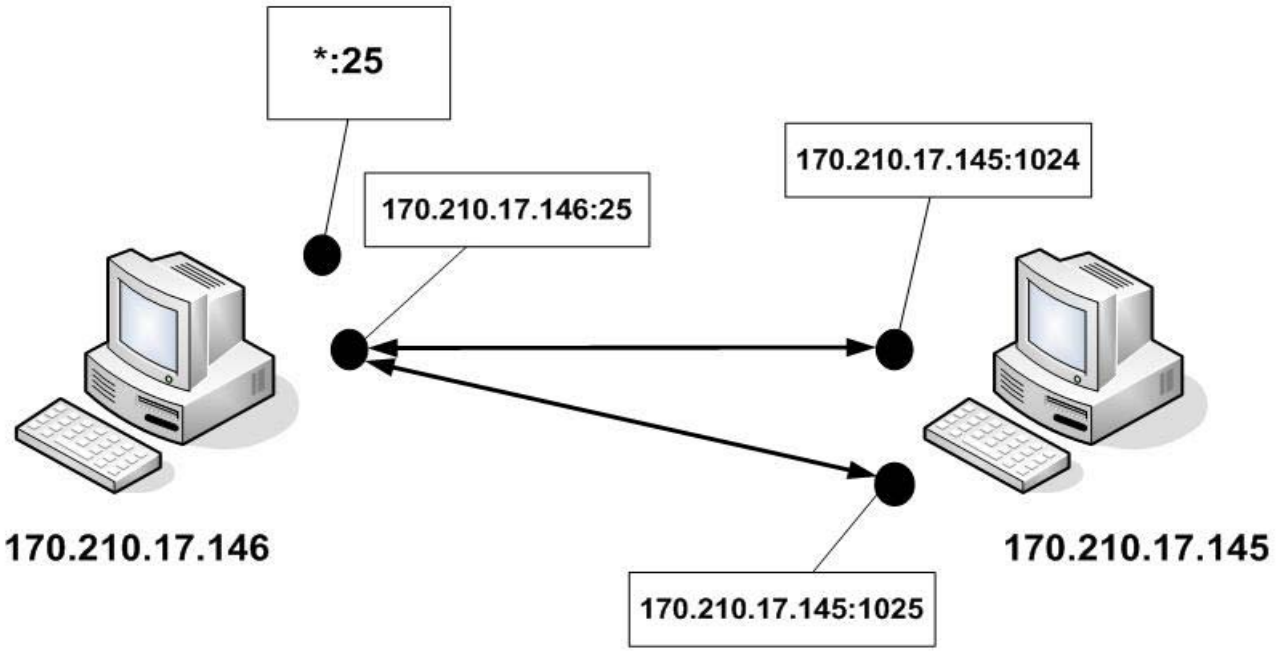
La conexión establecida estará únivocamente identificada en cada sistema como

{IP origen, puerto origen, IP destino, puerto destino}

Utilización de los numeros de puerto

(continuado)

Imaginemos ahora que otra aplicación perteneciente al mismo sistema deseara establecer otra conexión al puerto 25. El cliente utilizaría algún otro número de puerto que no estuviera utilizando en ese momento. Y, de ese modo, el escenario podría ser el siguiente:



Rango de los puertos efímeros (I)

La Internet Assigned Number Authority (IANA) tradicionalmente ha hecho el siguiente uso de los puertos TCP:

- Well Known Ports: 0 a 1023
- Registered Ports: 1024 a 49151
- Dynamic and/or Private Ports: 49152 a 65535

Es decir, se ha utilizado para los puertos efímeros el rango 49152-65535

Rango de puertos efímeros (II)

La mayoría de los sistemas eligen sus “puertos efímeros” de un subespacio de todo el espacio de puertos disponible. En muchos casos, el rango elegido difiere de aquél elegido por la IANA.

Sistema operativo	Puertos efímeros
Microsoft Windows	1024 - 4999
Linux kernel 2.6	1024 - 4999
Solaris	32768 - 65535
AIX	32768 - 65535
FreeBSD	1024 - 5000
NetBSD	49152 - 65535
OpenBSD	49152 - 65535

Algoritmos de selección de puertos efímeros

- Al seleccionar un puerto efímero, debe asegurarse que el connection-id resultante (client address, client port, server address, server port) no esté actualmente en uso.
- Si existe en el sistema local una conexión activa con dicho connection-id, se procederá a seleccionar otro puerto efímero, con el fin de salvar el conflicto.
- Sin embargo, es imposible para el sistema local poder detectar si existe alguna instancia de comunicación activa en el **sistema remoto** utilizando dicho connection-id.
- En caso que el puerto efímero seleccionado resultara en un connection-id que estuviera en uso en el sistema remoto, se dice que se ha producido una “colisión de puertos”.
- Las colisiones de puertos pueden resultar en corrupción de información o bien fallos en el establecimiento de conexión. Por ello, deben ser evitadas.

Algoritmo tradicional BSD (Algoritmo #1)

```
next_ephemeral = 1024;  /* init., could be random */
count = max_ephemeral - min_ephemeral + 1;

do {
    port = next_ephemeral;

    if (four-tuple is unique)
        return next_ephemeral;

    if (next_ephemeral == max_ephemeral)
        next_ephemeral = min_ephemeral;
    else
        next_ephemeral_port++;

    count--;
} while (count>0);

return ERROR;
```

Secuencia generada por el Algoritmo #1

Nr.	IP:port	min_ephemeral	max_ephemeral	next_ephemeral	port
#1	128.0.0.1:80	1024	65535	1024	1024
#2	128.0.0.1:80	1024	65535	1025	1025
#3	170.210.0.1:80	1024	65535	1026	1026
#4	170.210.0.1:80	1024	65535	1027	1027
#5	128.0.0.1:80	1024	65535	1028	1028

Características del Algoritmo #1

- Es el implementado en la gran mayoría de las pilas TCP/IP
- Es simple
- Posee una frecuencia de reuso de puertos aceptable (aunque mayor que la necesaria)
- **Produce una secuencia de puertos efímeros trivialmente predecible**.

Randomización de puertos

Randomización de puertos TCP

- Resulta lógico intentar mitigar la realización de ataques ciegos dificultando la habilidad del atacante de adivinar los valores {client IP, client TCP port, server IP, server TCP port}.
- De estos valores, el único valor que en principio es posible variar arbitrariamente es el client port.
- Sin embargo, en este aspecto la maýoría de las implementaciones facilitan la tarea del atacante más de lo necesario.
  - Utilizan para los puertos efímeros un rango de puertos muy reducido
  - El algoritmo de selección de puertos efímeros produce una secuencia trivialmente predecible (Algoritmo #1)

Características de un buen algoritmo de randomización de puertos

- Minimizar la predictibilidad de los números de puerto utilizados para futuras conexiones salientes.
- Minimizar la frecuencia de reuso de puertos (es decir, evitar “colisiones”).
- Evitar conflictos con aplicaciones que dependen de la utilización de números de puertos específicos (por ejemplo, evitar utilizar para los puertos efímeros números de puerto como el 80, el 100, el 6667, etc.)

Rango de puertos efímeros

- Se debería utilizar para los puertos efímeros el rango 1024-65535
- Para evitar la utilización de determinados puertos, se podría utilizar un array de bits que permita indicar, para el mencionado rango, cuáles puertos se deben utilizar, y cuáles no.
- Esto, en conjunto con un buen algoritmo de randomización de puertos, dificulta la tarea del atacante de “adivinar” los puertos efímeros utilizados por el cliente.

Algoritmo de randomización básico (Algoritmo #2)

```
next_ephemeral = min_ephemeral + random()
                    % (max_ephemeral - min_ephemeral + 1)

count = max_ephemeral - min_ephemeral + 1;

do {
    if(four-tuple is unique)
            return next_ephemeral;

    if (next_ephemeral == max_ephemeral)
        next_ephemeral = min_ephemeral;
    else
        next_ephemeral_port++;

    count--;
} while (count > 0);

return ERROR;
```

Características del Algoritmo #2

- Ha sido implementado en OpenBSD y FreeBSD
- Produce una secuencia de puertos efímeros muy difícil de predecir
- La frecuencia de reuso de puertos puede ser mucho mayor que la del algoritmo BSD (Algoritmo #1)
- Usuarios de los mencionados sistemas operativos han reportado problemas de interoperatividad. En consecuencia, FreeBSD incorporó un hack que deshabilita la randomización de puertos cuando el número de conexiones salientes por unidad de tiempo excede un determinado valor.

Algoritmo de randomización avanzado (Algoritmo #3) (Intro)

- A finales de los ’90, S. Bellovin introdujo un mecanismo de randomización para los Números de Secuencia Inicial (ISN), que generaba una secuencia dificil de adivinar para un atacante que estuviera “fuera del camino” de los paquetes, y que era monotónicamente creciente en el tiempo.
- El ISN se seleccionaba como resultado de la función:
  - ISN = M + F(localhost, localport, remotehost, remoteport)
  - M: es un contador que se incrementa en el tiempo
  - F: Funcion de hash
- Esta función separa el “espacio de números de secuencia”
- Nuestra propuesta es utilizar una función similar para la randomización de puertos.

Algoritmo de randomización avanzado (Algoritmo #3)

```
next_ephemeral = 1024;  /*init., could be random */

offset = F(local_IP, remote_IP, remote_port, secret_key);

do {
    port = min_ephemeral + (next_ephemeral + offset)
              % (max_ephemeral - min_ephemeral + 1);
    next_ephemeral++;

    if(four-tuple is unique)
        return port;

    count--;

} while(count > 0);

return ERROR;
```

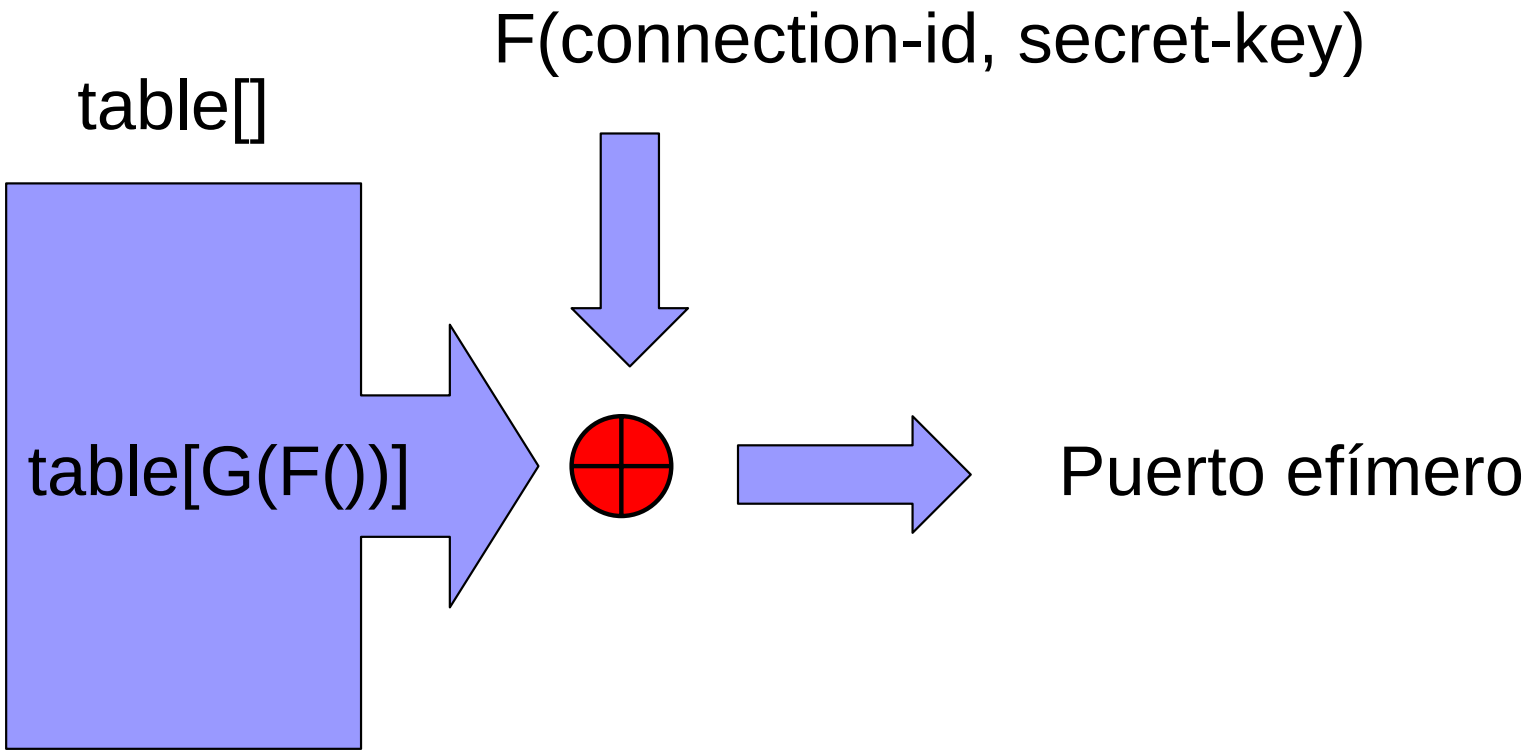
Secuencia producida por el Algoritmo #3

Nr.	IP:port	offset	min_ephemeral	max_ephemeral	next_ephemeral	port
#1	128.0.0.1:80	1000	1024	65535	1024	3048
#2	128.0.0.1:80	1000	1024	65535	1025	3049
#3	170.210.0.1:80	4500	1024	65535	1026	6550
#4	170.210.0.1:80	4500	1024	65535	1027	6551
#5	128.0.0.1:80	1000	1024	65535	1028	3052

Características del Algoritmo #3

- Implementado en el Linux Kernel
- Produce una secuencia muy difícil de predecir por terceros
- Posee una frecuencia de reuso de puertos igual a la del algoritmo tradicional BSD (Algoritmo #1)
- Su implementación es más compleja que la de los algoritmos anteriores

Algoritmo mejorado (Algoritmo #4)



- Se reduce la frecuencia de reuso de puertos mediante la separación del espacio de incrementos

Algoritmo mejorado (Algoritmo #4)

```
/* Initialization code */
for(i = 0; i < TABLE_LENGTH; i++)
    table[i] = random % 65536;

/* Ephemeral port selection */
offset = F(local_IP, remote_IP, remote_port, secret_key);
index = G(offset);
count = max_ephemeral - min_ephemeral + 1;

do {
    port = min_ephemeral + (offset + table[index])
                % (max_ephemeral - min_ephemeral + 1);
    table[index]++;
    count--;
      if(four-tuple is unique)
    return port;
} while (count > 0);
return ERROR;
```

Secuencia producida por el Algoritmo #4

Nr.	IP:port	offset	min_ephemeral	max_ephemeral	index	table[index]	port
#1	128.0.0.1:80	1000	1024	65535	10	1024	3048
#2	128.0.0.1:80	1000	1024	65535	10	1025	3049
#3	170.210.0.1:80	4500	1024	65535	15	1024	6548
#4	170.210.0.1:80	4500	1024	65535	15	1025	6549
#5	128.0.0.1:80	1000	1024	65535	10	1026	3050

Características del Algoritmo #4

- Se está trabajando en implementaciones apra FreeBSD y OpenBSD
- Produce una secuencia muy difícil de predecir por terceros
- Posee una frecuencia de reuso menor que la del algoritmo tradicional BSD (Algoritmo #1)
- Su implementación es más compleja que la de todos los algoritmos anteriores

Conclusiones

- La randomización de puertos es una medida proactiva para evitar ataques “ciegos” contra protocolos de transporte.
- Se puede aplicar a todos los protocolos de transporte existentes: TCP, UDP, SCTP, DCCP, etc.
- Lamentablemente, pocas implementaciones randomizan los puertos efímeros.
- De aquellas que lo hacen,muchas lo hacen incorrectamente, causando problemas de interoperabilidad.

Preguntas?

Información de contacto

Fernando Gont

fernando@gont.com.ar

Más información en:

http://www.gont.com.ar