

Mejorando la seguridad de TCP/IP mediante aleatorización de parámetros de protocolo

Fernando Gont

Universidad Tecnológica Nacional
Facultad Regional Haedo

ekoparty security sonference

Noviembre 30 & Diciembre 1, Buenos Aires, Argentina



Acerca de....

- Miembro del Centro de Estudios de Informática (CEDI) de UTN/FRH, trabajando en el área de ingeniería de Internet.
- Participante activo de la IETF (Internet Engineering Task Force), participando en el proceso de estandarización de los protocolos de comunicaciones utilizados por la red Internet. Autor de cuatro internet-drafts adoptados oficialmente por la IETF para su futura publicación como RFCs.
- Durante algún tiempo fue miembro del equipo de desarrollo de OpenBSD, focalizando mi actividad en la pila TCP/IP.
- Realicé actividades de investigación en el área de seguridad de protocolos de comunicaciones para UK's NISCC (United Kingdom's National Infrastructure Security Co-ordination Centre).
- Actualmente trabajando para UK CPNI.
- Docente de una asignatura de la TSP dictada en UTN/FRH. Y ayudante de dos asignaturas de la carrera de Ingeniería Electronica dictada en UTN/FRH.



Agenda

- Introducción
- Revisión de algunos conceptos de TCP
- TCP Initial Sequence Numbers (ISN's)
- TCP port numbers
- TCP timestamps
- Algunas conclusiones

Introducción (I)

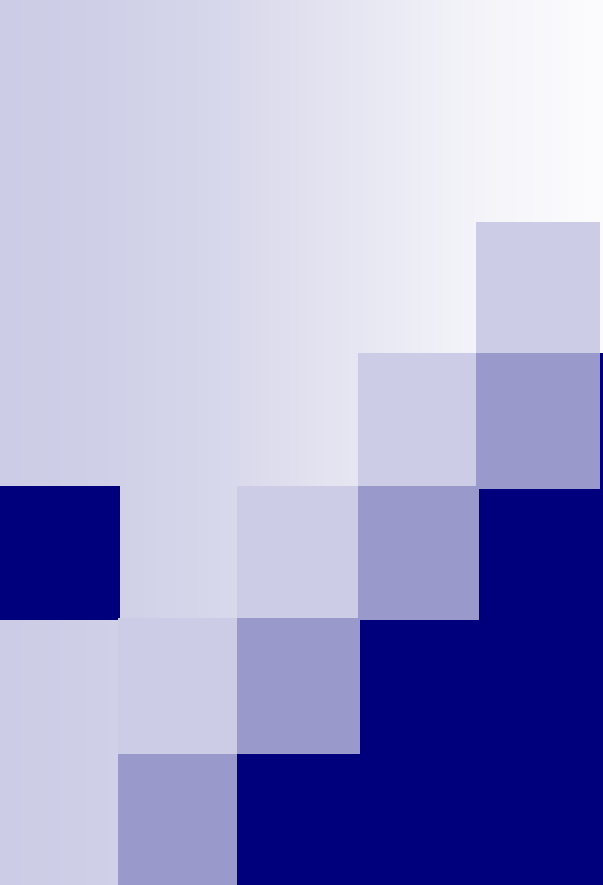
- Los protocolos TCP/IP fueron concebidos en para operar en un ambiente muy distinto al ambiente hostil en el que operan actualmente.
- Pese al “mito” de que fueron diseñados con el fin de operar en ambientes de guerra, su principal objetivo fue poder compartir los recursos de grandes servidores existentes en la red ARPANET.
- Estas consideraciones se reflejan directamente en las características de seguridad de los protocolos involucrados.
- Pese a que el diseño original no tiene en cuenta aspectos de seguridad, es posible mejorar la resistencia de los protocolos a una variedad de ataques, mediante pequeñas modificaciones compatibles con los estándares.

Introducción (II)

- Durante los últimos veinte años se han publicado una variedad de vulnerabilidades en los protocolos TCP/IP.
- Pese a que la respuesta general ante cuestiones de seguridad en protocolos de comunicaciones suele ser “use IPsec!”, existe una variedad de escenarios en los cuales esto puede ser no posible (ej., no implementación de IPsec, o dificultad para el establecimiento dinámico de SAs), o indeseable (ej., sistemas con baja capacidad de procesamiento, en los cuales simplemente se desea evitar ataques **triviales**).
- Asimismo, algunas tecnología (como SSL) dependen del correcto funcionamiento de la capa de transporte (ej., TCP) y la capa de Internet (ej., IP).
- Es ésta la motivación para intentar mejorar la seguridad de los protocolos “core”, sin recurrir a técnicas criptográficas.

Aleatorización de parámetros de protocolos

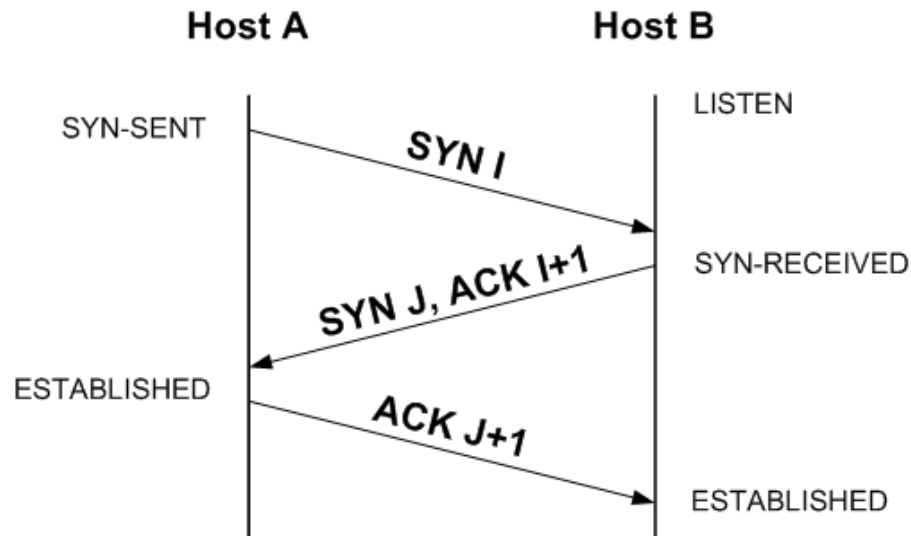
- Muchas de las vulnerabilidades descubiertas en los protocolos TCP e IP requerían que el atacante pudiera adivinar diversos parámetros de los protocolos (como por ejemplo números de puertos, números de secuencia, etc.).
- Algunas implementaciones intentaron mitigar muchas de estas vulnerabilidades mediante la simple aleatorización de dichos parámetros.
- Desafortunadamente, dichas modificaciones han tenido implicancias negativas en la interoperabilidad de los protocolos, y usualmente han sido motivadas por concepciones erróneas del funcionamiento de los protocolos.
- Mike Silbersack (FreeBSD) hizo una presentación muy interesante (“Improving TCP/IP Security through randomization without sacrificing interoperability”) en las conferencias EuroBSD 2005 y BSDCan 2006, en donde inició una discusión sobre el impacto de la aleatorización de diversos parámetros de los protocolos TCP/IP en la interoperabilidad de los mismos.
- La presentación dejó varias cuestiones abiertas. Y esta presentación hace las veces de “*follow-up*”.



Breve revisión de algunos conceptos básicos de TCP

Establecimiento de conexión

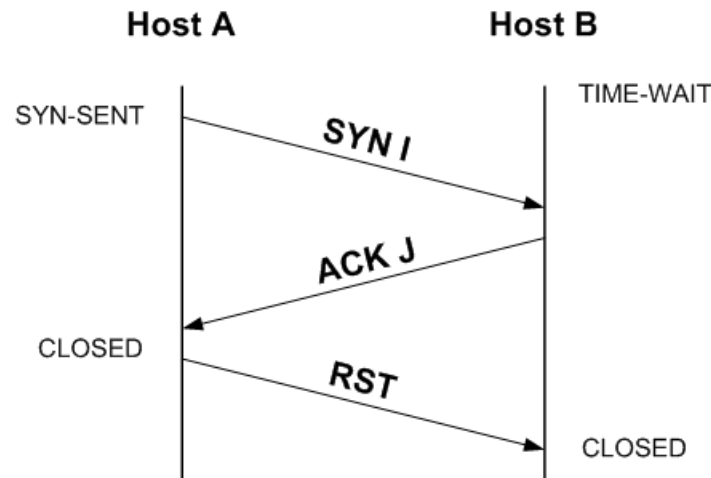
- El proceso de establecimiento de conexión consiste usualmente en el intercambio de tres segmentos.



- Una vez finalizado el “three-way handshake”, los números de secuencia (y demás parámetros) estará sincronizados, y se podrá comenzar la transferencia de información.

Colisiones de connection-id's

- Sin embargo, puede suceder que al realizarse una petición de conexión exista una encarnación previa de la misma conexión en el sistema remoto. En tal caso, la petición de conexión fallará.



- Claramente, esta situación (colisión de connection-id's) es indeseable, por lo cual, en la medida que sea posible, deberá ser evitada.



TCP Initial Sequence Numbers

TCP Initial Sequence Numbers

- RFC 793 menciona que los ISN's se han de resultar en una secuencia monótonamente creciente, a partir de un timer global, con el fin de evitar la superposición de espacios de secuencia. Desde ahí (año 1981), se ha asumido que la generación de ISN's de forma lineal es crucial para la confiabilidad de TCP (protección contra segmentos "viejos").
- Sin embargo, la verdadera protección contra segmentos viejos está provista por otros dos mecanismos que no tienen relación alguna con los ISN:
 - "Quiet time concept": Cuando se reinicia el sistema, se debe esperar $2 * MSL$ antes de transmitir segmentos TCP.
 - TIME-WAIT state: Cuando se finaliza una conexión TCP, el extremo que realizó el "active close" debe permanecer en el estado TIME-WAIT por $2 * MSL$, asegurando así que los segmentos "viejos" desaparezcan de la red.

TCP Initial Sequence Numbers (II)

- En la implementación tradicional BSD, el generador de ISN se inicializaba en 1 al iniciar el sistema, y se incrementaba en 64000 cada medio segundo, y en 64000 por cada conexión establecida.
- Basado en la suposición de que los ISNs son generados linealmente, BSD implementó una modificación al comportamiento estandar de TCP, con el fin de permitir altas tasas de petición de conexiones. Si se recibe un SYN para una conexión que se encuentra en el estado TIME-WAIT, entonces,
 - Si el ISN del SYN es mayor que el último número de secuencia recibido ($SEG.SEQ > RCV.NXT$), se remueve el TCB en estado TIME-WAIT, y se crea uno en el estado SYN-RECEIVED.
 - En caso contrario, se siguen las reglas del RFC 793.
- Sin embargo, es interesante mencionar que este “fix” fue motivado por el uso de los comandos r^* , es decir, para conexiones cortas, que transmiten **poca** información, y/o a una baja tasa de transferencia. En caso contrario, esta “heurística” falla.

Implicancias de seguridad

- Las implicancias de seguridad de la predictibilidad de ISNs fue descrita por primera vez por Morris en 1985, en el trabajo “*A Weakness in the 4.2BSD Unix TCP/IP Software*”.
- La explotación de dicha vulnerabilidad para spoofing the conexiones TCP fue ampliamente publicitada 10 años mas tarde por T. Shimomura en “*Technical details of the attack described by Markoff in NYT*” (el famoso ataque de K. Mitnick).
- Asimismo, una gran cantidad de ataques “ciegos” contra TCP dependen de la facilidad para adivinar números de secuencia TCP.

Aleatorización de ISN's

- Algunas implementaciones (por ej., OpenBSD) decidieron simplemente aleatorizar la generación de ISN, con el fin de mitigar aquellos ataques basados en la predicción de números de secuencia. Lamentablemente, esto llevó a que en determinados escenarios, las peticiones de conexión fallaran, cuando antes esto no sucedía.
- S. Bellovin propuso, en RFC1948, un algoritmo para la selección de ISNs, que permitir que la heurística de las implementaciones BSD pueda seguir funcionando con éxito. Dicho algoritmo propone la selección de ISNs de acuerdo a la expresión:
$$\text{ISN} = M + F(\text{localhost}, \text{localport}, \text{remotehost}, \text{remoteport}, \text{secret})$$
- Esta función permite que los ISN para conexiones a un determinado host sean generados linealmente, a partir de un offset aleatorio, separando así los espacios de números de secuencia.
- Considerando tanto los aspectos de seguridad como de interoperabilidad, es aconsejable implementar un esquema como el propuesto en RFC1948, y **no** un esquema de aleatorización “trivial”.



TCP ephemeral ports

TCP ephemeral ports

- 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, y requieren (como mínimo) que el atacante conozca o pueda adivinar el connection-id (client IP, client port, server IP, server port).
- Mas allá de las posibles soluciones específicas para estas vulnerabilidades, resulta lógico mitigar las mismas dificultando la tarea del atacante para adivinar el connection-id.
- De los cuatro valores que componen al connection-id, el único que en principio puede elegirse arbitrariamente es el puerto TCP del cliente, o el puerto “efímero” de la conexión.

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 (por ejemplo, una conexión TCP en el estado TIME-WAIT).
- En caso que el puerto efímero seleccionado resultara en un connection-id en uso en el sistema remoto, se dice que se ha producido una “colisión de connection-id’s”, y el intento de conexión usualmente fallará.
- En consecuencia, la frecuencia de reuso de puertos debe ser minimizada.

Rango de puertos efímeros

IANA asigna a los puertos efímeros el rango 49152-65535. Sin embargo, la mayoría de los sistemas eligen sus “puertos efímeros” de un subespacio de todo el espacio de puertos disponible que, en la mayoría de los casos, 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 |

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 tradicional BSD (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.**

Aleatorización de puertos TCP efímeros

- Un buen algoritmo de aleatorización de puertos TCP efímeros debería:
 - Minimizar la predictibilidad de los números de puerto utilizados para futuras conexiones salientes.
 - Minimizar la frecuencia de reuso de puertos (es decir, evitar las “colisiones” de connection-id’s).
 - 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.)
- Asimismo, y por razones obvias, el rango de los puertos efímeros debería ser maximizado.

Algoritmo de aleatorización 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 aleatorización de puertos cuando el número de conexiones salientes por unidad de tiempo excede un determinado valor.

Un mejor algoritmo de aleatorización (Algoritmo #3) (Intro)

- Nuestra propuesta (“Port Randomization”, Larsen, M. y Gont, F.) es seleccionar los puertos TCP efímeros mediante una función similar a la propuesta por Steven Bellovin para los ISN:

$$\text{Port} = M + F(\text{local_IP}, \text{remote_IP}, \text{remote_port}, \text{secret_key})$$

- De este modo, se disminuye la predictibilidad de los puertos efímeros (separando el espacio de números de puerto), manteniendo baja la frecuencia de reutilización de puertos.

Un mejor algoritmo de aleatorización (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;
```

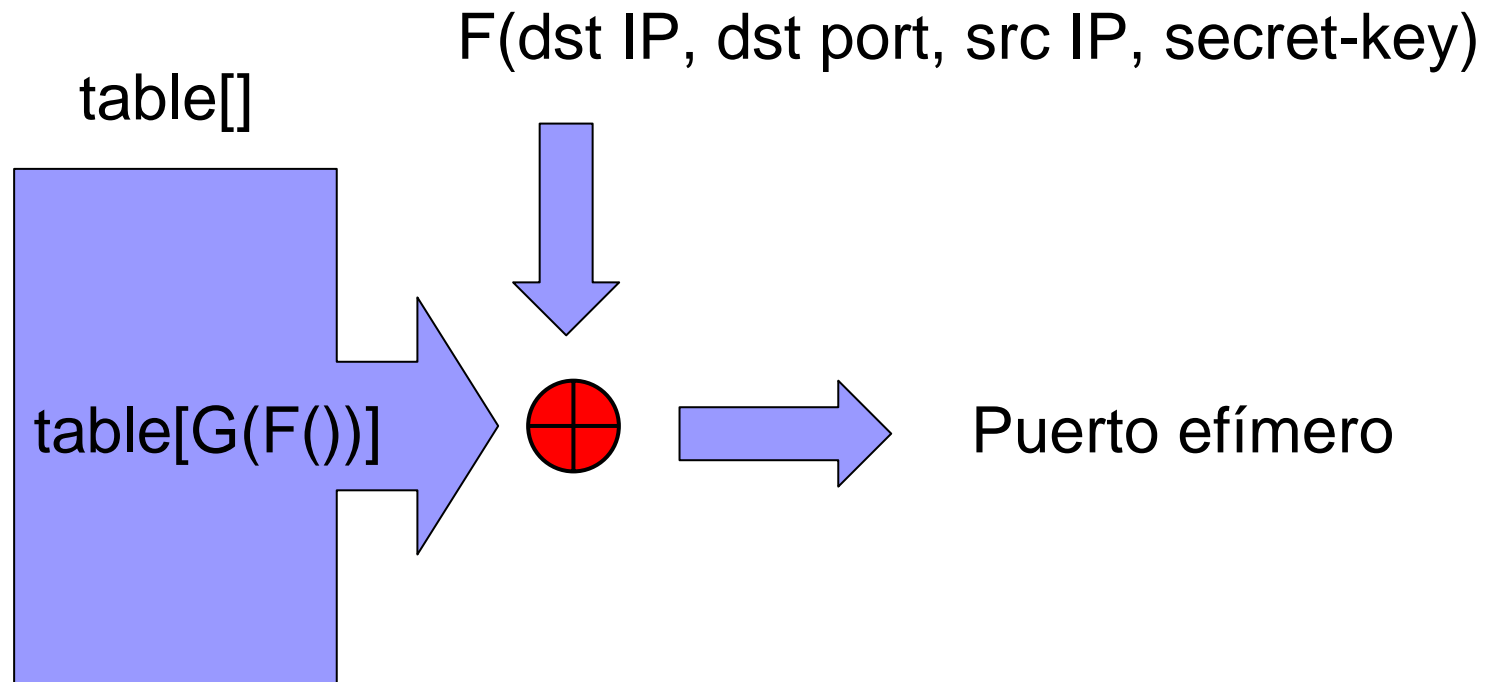
Secuencia producida por el Algoritmo #3

| 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 puede reducir 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 para 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



TCP timestamps

TCP Timestamps

- Las opciones TCP timestamp fueron introducidas por RFC 1323 con dos propósitos:
 - Medir el Round-Trip Time de la conexión
 - Proteger a las conexiones de segmentos “viejos”, en caso de que se utilicen grandes ventanas TCP (PAWS).
- Desde el punto de vista de PAWS, los TCP timestamps vienen a ser una extensión del número de secuencia.
- Para permitir la función de PAWS, los timestamps deben ser monotónicamente crecientes. El RFC 1323 sugiere una frecuencia para los timestamps de entre 1/1s y 1/1/ms.
- Basándose en esta premisa, algunas implementaciones (ej., Linux) permiten la destrucción del estado TIME-WAIT si el SYN entrante tiene un timestamp mayor al último recibido (es decir, un fix similar al de los BSD para el caso de los ISN).

Implicancias de seguridad de los TCP timestamps

- Para inyectar segmentos TCP en una conexión que utilice timestamps, el atacante tendrá que adivinar un timestamp válido. Por ello cuanto más difícil sea predecir los timestamps, más difícil será realizar ataques “ciegos” contra conexiones TCP.
- Asimismo, los TCP timestamps tienen proveen dos vectores previamente inexistentes:
 - Si se logra inyectar un segmento con un timestamp válido, pero mayor al recibido por ese TCP hasta el momento, futuros segmentos legítimos serán descartados (es decir, la conexión se “congelará”).
 - Si el origen de la secuencia de timestamps se inicializa a un valor fijo cada vez que se reinicia el sistema, el mismo revelará el uptime del sistema en cuestión.

Aleatorización de timestamps

- Algunas implementaciones de TCP (por ej., OpenBSD) decidieron aleatorizar los timestamps, con el fin de mitigar los vectores anteriormente descritos.
- Sin embargo, esta decisión tiene un impacto negativo en la interoperabilidad de los protocolos:
 - Se rompe la función de PAWS
 - Se rompe la “optimización” de Linux (y otros).
- Para mitigar los vectores mencionados, sin afectar la interoperabilidad de los protocolos, una posibilidad es generar los timestamps de acuerdo a una expresión del tipo (RFC1948):
$$TS = M + F(\text{localhost}, \text{localport}, \text{remotehost}, \text{remotepport}, \text{secret})$$



Armando el rompecabezas

Efecto de la implementación conjunta de todos los
mecanismos propuestos

Procesamiento de SYNs entrantes

- Si existe una encarnación previa de la misma conexión en el estado `TIME_WAIT`, entonces:
 - Si la encarnación previa utilizaba timestamps, aplicar la optimización propuesta para los timestamps. Si el timestamp del SYN fuera igual al último recibido por la encarnación previa, realizar el chequeo de ISN del SYN (fix de BSD).
 - Si la encarnación previa no utilizaba timestamps, pero el SYN entrante incluye uno, entonces permitir el establecimiento de la conexión.
 - Si ni la encarnación previa ni la nueva utilizan timestamps, entonces simplemente aplicar el chequeo de ISNs al SYN (fix de BSD).

Mejoras en la mitigación de ataques ciegos

- Con las modificaciones descritas, se mejora notablemente la resistencia de TCP a ataques ciegos:
 - Se hace más dificultoso predecir el connection-id (gracias a la aleatorización de puertos efímeros)
 - Se hace más dificultoso predecir los números de secuencia (gracias a lo propuesto por S. Bellovin en RFC 1948)
 - Se hace mas dificultosa la predicción de TCP timestamps (de acuerdo al esquema de aleatorización propuesto).

Mejoras en interoperabilidad

- Se reducen las colisiones de connection-id's.
- Incluso si existieran colisiones de connection-id's, la optimización realizada mediante TCP timestamps permitirá una alta tasa de establecimiento de conexiones.
- Si dicha tasa fuera muy alta, y el último timestamp de la conexión coincidiera con el del SYN entrante, el ISN del SYN entrante nos brindaría otra oportunidad para aceptar la conexión.

Posibles escenarios de falla

- En escenarios en los cuales existe una variedad de sistemas detrás de un NAT realizando conexiones con un mismo servidor, las optimizaciones anteriormente propuestas podrían fallar:
 - Se podría incrementar la frecuencia de reuso de puertos, generándose colisiones de connection ID's
 - Se podrían solapar los espacios de números de secuencia (fallando la optimización de los BSD).
 - Se podrían solapar los espacios de los timestamps (fallando la optimización propuesta en esta presentación)
- Para evitar dichos inconvenientes, se sugiere la implementación de los algoritmos descritos, en el propio NAT (lo cual puede o no ser posible).
- En escenarios donde la identidad de distintos sistemas es “ocultada” detrás de un NAT, los esquemas del tipo RFC1948 pueden revelar información sobre la cantidad y/o identidad de los sistemas detrás del NAT.

Posibles mejoras

- Una posible modificación tendiente a posibilitar una tasa de conexiones más alta consistiría en adoptar para el estado TIME-WAIT (de usualmente 4 minutos), un valor mas adecuado.
 - Si consideramos que el estado TIME-WAIT debe representar $2 * MSL$, resulta lógico que el mismo sea función del RTT/RTO de la conexión.
 - Asimismo, es interesante notar que en la arquitectura TCP/IP no existe mecanismo alguno (como sí sucede en otros protocolos como Delta-t) para limitar el tiempo de vida **real** de los paquetes en la red.
 - Es decir, si uno es “purista”, el IP TTL **no** pone ninguna cota real a la vida de los paquetes en la red, y en conclusión el estado TIME-WAIT tampoco garantiza nada al respecto.
- Por ejemplo, el estado TIME-WAIT podría configurarse como de $10 * RTO$, $100 * RTO$, etc.



Conclusiones

Conclusiones

- Usualmente se asume que, debido a la antigüedad de los protocolos “core” de la suite TCP/IP, todas las implicancias negativas de seguridad del diseño de los mismos han sido resueltas, o solo pueden resolverse mediante uso de IPsec.
- Las vulnerabilidades publicadas incluso en los últimos cinco años parecen indicar lo contrario.
- Mediante la elección cuidadosa de determinados parámetros de los protocolos involucrados, se pueden mitigar una variedad de vulnerabilidades que, de lo contrario, son triviales de explotar.
- Muchos de los conceptos explicados en esta presentación pueden aplicarse a todos los protocolos de transporte existentes: TCP, UDP, SCTP, DCCP, etc.
- Lamentablemente, incluso al día de la fecha muchas pilas TCP no implementan contramedidas triviales como por ejemplo aleatorización de puertos.
- Pese a que el interés general suele centrarse en estudiar protocolos “nuevos”, lo cierto es que muchas veces, cuando “la Internet no anda” TM ó “la Internet anda mal” TM, la falla suele residir en protocolos viejos (los que **estamos usando!**)



Preguntas?



Información de contacto

Fernando Gont

fernando@gont.com.ar

Más información en:

<http://www.gont.com.ar>



Gracias!