CC4303-1 Redes
Profesor: José M. Piquér
Auxiliar: David Miranda
Estudiante: Andrés Calderón Guardia

Control 3

Redes

P1. TCP avanzado

P1.1. Protocolo de Conexión y fin

1. El funcionamiento del ataque SYN Flood consiste en que el atacante se aprovecha del protocolo de enlace que implementa la conexión TCP, dado que envia un alto volumen de paquetes SYN al servidor atacado, de esta forma el servidor debe responder a cada una de las solicitudes de conexión con paquetes SYN-ACK para luego dejar un puerto abierto con el propósito de recibir la confirmación (cuando un servidor deja una conexión abierta pero la máquina al otro lado no se conoce como conexión medio abierta), en donde finalmente el atacante explota este comportamiento al no enviar de vuelta ningún ACK, de modo que el servidor seguirá abriendo puertos hasta que no queden más disponibles y entonces este dejará de funcionar con normalidad.

   Para este problema existen distintas soluciones, la primera siendo la creación de reglas de filtrado robustas tal que con estas se pueda identificar y bloquear solicitudes SYN maliciosas basándose en comportamientos específicos o direcciones IP maliciosas conocidas previamente. Otro método consiste en reciclar las conexiones TCP medio abiertas más antiguas cuando se complete el backlog. También existe la posibilidad de aumentar directamente el tamaño del backlog. Y por último, está la opción de reducir el tiempo que espera el servidor a recibir un ACK tras enviar el paquete SYN-ACK.

   Entre los distintos métodos planteados, la que considero una mejor opción es combinar los tres últimos métodos, en primer lugar descartando el primer método ya que su efectividad puede depender de la complejidad del algoritmo utilizado para detectar estos comportamientos, y aunque se tenga un buen método siempre puede existir la posibilidad de que un atacante logre evadir estos filtros. Por ello es que considero una mejor opción combinar dichas soluciones, ya que estas por sí solas no son una solución tan efectiva pero en combinación se apoyan mutuamente, dado que al aumentar el backlog se da la posibilidad de que el método de reciclaje solo se llame de ser necesario al estar recibiendo una cantidad masiva de solicitudes, y por último, si se reduce el tiempo de espera para todas estas solicitudes es posible usar más efectivamente el backlog completo al tener disponible más rápido los puertos.

2. Las SYN Cookies funcionan tal que cuando el servidor responde con un paquete SYN-ACK a las solicitudes de conexión TCP, en uno de los parámetros de este paquete se designa un número de secuencia específico a dicho paquete calculado mediante esta técnica, la cual guarda información acerca de la solicitud inicial, luego se elimina la solicitud SYN a la que el servidor le respondió del backlog, de forma que no asigna memoria a esta conexión, dejando el puerto abierto y preparado para una nueva conexión, en la cual si se tiene una conexión legítima y recibe dicho ACK de vuelta, entonces ahora si se asigna memoria a la conexión, recuperando parte de la información dada por el SYN inicial en base a decodificar el número de secuencia entregado por el ACK.

   Esta solución en comparación a la planteada en la pregunta anterior se enfoca en resolver el problema de base más que aumentar la capacidad del servidor para manejar dichos ataques o implementar técnicas para utilizar la memoria de mejor manera. Por este motivo, elijo esta solución por sobre la primera para resolver el problema dada la robustez que ofrece ante dicho ataque por lo mencionado anteriormente.

3. Que estos números de secuencia sean predecibles, en este caso partiendo siempre en 0, genera problemas dado que para los atacantes les será más sencillo suplantar la identidad de usuarios al conocer los números de secuencia válidos de dichas conexiones TCP, de forma que podría enviar paquetes a ambos extremos de la conexión con fines maliciosos. Y una segunda razón es para tener protección contra paquetes antiguos duplicados, ya que al terminar una conexión pueden haber paquetes aún en tránsito tales que para una conexión posterior pueden confundirse con información real que fue enviada solo durante esta última, siendo que no fue el caso.

4. Uno no puede olvidarse de la conexión tras recibir el paquete FIN y responder con su ACK, porque al ser TCP hay que asegurarse de que este paquete llegue correctamente al otro extremo, ya que si nos olvidásemos de la conexión y se perdiese dicho ACK en el camino entonces una de las maquinas no tendría la confirmación necesaria para realizar el cierre.

   La forma en que TCP resuelve el cierre de las conexiones es la siguiente, considerando el caso particular en que el cliente desea cerrar la conexión, pero el caso opuesto es análogo:
   1. El cliente que desea terminar la conexión envía un paquete FIN al servidor y se queda esperando a recibir un ACK y luego un FIN de vuelta.
   2. El servidor recibe FIN y entonces envía el ACK correspondiente, luego espera un momento y envía su FIN al cliente, quedando a la espera del ACK para este paquete.
   3. El cliente recibe el ACK correspondiente al FIN que envió, se queda esperando a recibir el FIN del servidor y cuando lo obtiene envía de vuelta un ACK.
   4. El servidor recibe dicho ACK y cierra su conexión.
   5. El cliente espera una cierta cantidad de tiempo antes de cerrar su conexión, con el fin de verificar que su ACK efectivamente llegó al servidor al no recibir retransmisiones en este periodo.

   En todos los pasos, al ser el protocolo TCP los paquetes vienen con números de secuencia asociados, de modo que si con este número se observa que un paquete no ha sido recibido o enviado entonces dicho extremo simplemente reenvía el paquete, manejando así la pérdida de paquetes en general.

5. Una extensión que haría resistente a este tipo de ataques el protocolo de la T2 sería implementar un MAC para la conexión entre cliente y servidor, de modo que al inicio de la comunicación sea posible generar esta clave y que sirva como método de autentificación entre ambas partes. Es posible lograr la creación de dicha clave entre dos extremos mediante algoritmos como el logaritmo discreto por lo que es una solución realista.

P1.2. Control de Flujo y Congestión

1. Las ventanas de envío y recepción permiten controlar la cantidad de datos en tránsito entre un emisor y un receptor, pero solo usando esto es posible enfrentarse a un problema en que el emisor envía una mayor cantidad de datos que los que puede manejar el receptor, pues el receptor pese a declarar un cierto tamaño de ventana podría no lograr manejar un flujo constante de grandes datos con dicha ventana. Por ello es necesario utilizar control de flujo para evitar sobrecargar al receptor, ya que con este es posible manejar el ritmo de envío con las capacidades del receptor dadas las estrategias que implementadas.

2. Achicando la ventana de recepción es posible mejorar la situación del receptor ya que este si será capaz de manejar los datos que reciba al haber un menor volumen de estos, pero hacer este cambio supone un problema dado que estará limitando el ancho de banda efectivo que podría estar ocupando resultando en la disminución de la capacidad de envío de paquetes en paralelo, ralentizando la conexión.

3. Al ser una simulación realizada en una misma maquina es que ambas partes procesan paquetes a la misma velocidad, de modo que no es necesario utilizar control de flujo ya que el emisor nunca podrá saturar al receptor.

4. Esto puede ser cierto, ya que es posible tener una situación en que la red está congestionada y entonces al cerrar dicha conexión se reinician los paquetes en vuelo y las ventanas pasan a tener

sus valores iniciales, de forma que ahora se puede rehacer la descarga pero con una red que se libró de la congestión.

5. Es posible que con este método se logre una mejora ya que utilizando una sola conexión puede darse el caso en que las retransmisiones generen mucho sobrecosto al estar enviando paquetes muy grandes y entonces bloquear el progreso del envío del resto del archivo, en contraste con múltiples conexiones, donde si se pierde un paquete no afecta al resto de envíos. Además, si la conexión es estable es posible que no se genere mucha congestión y por ende sea una mejor opción paralelizar el envío al disponer de un mayor ancho de banda efectivo, considerando que la ventana de congestión es independiente por cada conexión.

P1.3. Manejo de Timeouts

1. Con timeouts muy pequeños es posible que nunca se logre recibir un ACK de vuelta para el paquete correspondiente dada la latencia del receptor, y entonces la conexión quede bloqueada ya que la ventana del emisor nunca avanzaría.
2. La conexión puede ser muy lenta al esperar mucho más de lo necesario para reenviar paquetes, y por ende, las retransmisiones generarían un sobrecosto mucho mayor del necesario en la conexión, por lo que para conexiones con pérdidas notables la red sería mucho más lenta de lo que podría ser.
3. Consideraría mejor elegir un timeout muy grande ya que por lo menos no bloquearía el avance de las ventanas para receptores con una latencia muy alta, permitiendo que cualquier emisor sea capaz de interactuar con este, además de que esta solo genera problemas notables para conexiones con altas pérdidas.
4. TCP utiliza este algoritmo dado que es una buena forma de estimar el valor del round-trip time (RTT) el cual es un valor necesario de obtener para distintos protocolos, ya que solo calcula la diferencia de tiempo entre enviar un paquete y recibir su ACK para paquetes que no han sido retransmitidos, pues si estos se considerasen provocaría el cálculo de RTTs ambiguos ya que no se sabe si el ACK correspondía al paquete inicial o alguna de sus retransmisiones, en cuyo caso es mejor descartarlo para eliminar dicha incerteza.
5. Utilizar dichos timestamps es una buena idea ya que es útil para obtener el RTT independientemente de si hubieron retransmisiones o no ya que se elimina la ambigüedad que había previamente al incluir estos valores en los paquetes, e incluir esta información requiere poco espacio adicional al ser solo un número de una cantidad definida de bytes y puede mejorar la eficiencia de los protocolos al no depender de un temporizador por cada paquete enviado para obtener dicho valor.
6. Dicho esquema va a ser de la siguiente manera, considerando modificar solo las partes implicadas en el timeout y RTT:
    1. Al enviar un paquete por primera vez este contador empieza en 0. Si se llega a un timeout este contador aumenta en 1 y se reenvía el paquete, generando un temporizador por retransmisión.
    2. El receptor devuelve el respectivo ACK junto al contador del paquete recibido.
    3. Cuando el emisor recibe este ACK calcula el RTT en base al temporizador asociado al contador recibido en este paquete y descarta el resto.

P2. Redes IP

P2.1. Reserva de Recursos

La asignación dinámica de recursos fue definitivamente una de las principales razones por las cuales el internet fue un éxito, ya que esta diferencia le permitió al internet ser más eficiente con el uso de los recursos al adecuarse a la demanda instantánea, maximizando el uso compartido y reduciendo costos, lo cual resulta especialmente útil en conexiones asimétricas o intermitentes, no como con la telefonía en donde el ancho de banda y la memoria permanecen ocupados incluso en momentos de silencio durante las llamadas.

Además de esto también se presenta el beneficio de que dado que en el internet hay distintos tipos de demandas, como lo puede ser la transferencia de archivos, servicios de streaming, entre otros, asignar memoria dinámicamente para cada caso permite una mayor flexibilidad para soportar una amplia gama de servicios.

Por último, esta decisión hizo posible que el internet escalara masivamente, ya que reservar espacio estáticamente sería inviable para una red global como el internet de hoy en día.

Esta simple decisión fue la causa de múltiples factores de los cuales se pudo aprovechar el internet para triunfar ante la red telefónica.

P2.2. IPv6

1. Muchas redes y servicios aún operan únicamente sobre IPv4. Si una empresa usa solo IPv6, no podrá comunicarse directamente con estos servicios que utilizan IPv4.

2. El principal problema con solo utilizar IPv6 es la falta de soporte para IPv4, lo cual puede resolverse mediante mecanismos de traducción entre estos dos como lo es NAT64, el cual le permite a hosts IPv6 que puedan comunicarse adecuadamente con servidores IPv4 dada la incompatibilidad.

   Por otra parte, existe un lado más limitante respecto a que todo el software y hardware del que dispone la empresa debe ser capaz de soportar IPv6.

3. Considerando que se debe utilizar IPv6 sería más conveniente tener una red privada que las utilice, dado que de esta forma no necesitarían utilizar mecanismos de traducción entre dispositivos de la red, aunque si en caso de que se quiera comunicar con estas fuera de dicha red, mientras que si fuese pública necesitarían usarlos para cada uno cuando se requiera realizar una conexión con un servidor IPv4, además de que al ser privada mantendrían una mejor seguridad de la red de la empresa, algo por lo general deseable para las mismas.

P2.3. Máscara de Red

1. Tener esta información directamente facilitaría identificar rápidamente la red a la que pertenece una dirección, sin necesidad de consultar tablas de rutas o máscaras externas, lo cual provoca a su vez que sea posible identificar redes locales fácilmente.

2. Sí, puede ser más eficiente dado que al poseer su respectivo prefijo explícitamente en la dirección no requiere de realizar la búsqueda de la máscara sobre la tabla de rutas, pasando de dicha complejidad de búsqueda a una constante, ahorrando tiempo de procesamiento.

3. Esto no es posible dado que puede darse una situación en el que un router deba revisar una ruta más específica que la indicada por el paquete ya que tiene reglas particulares para dicha ruta, de forma que en cambio necesite los bits indicados en su tabla. Y otro problema que se presenta es la optimización en espacio de la tabla, dado que sin estos bits no sería posible acotar la memoria utilizada cuando ocurra solapamiento, perdiendo la ventaja de agrupar rutas en el enrutamiento.

4. Implementando esto se presenta la problemática respecto al tamaño del header de dichos paquetes, ya que ahora necesitarán el espacio extra para almacenar esta información, generando problemas en conexiones con restricciones en el ancho de banda.

Un segundo problema es que realizar este cambio requeriría modificar varias implementaciones de estos protocolos para poder soportar el cambio, generando así incompatibilidades.

Por último, está el tema de no solo modificar los protocolos sino de también modificar la lógica de los routers, además de que delegar parte de esa lógica a los paquetes generaría problemas como se vio en la pregunta anterior.

5. Por lo dicho anteriormente es que se concluye que esto sería una mala idea, dado las complicaciones que conlleva implementar esta estrategia en un sistema actualmente funcional de la forma en que está hecho.

   Un segundo factor a tomar en cuenta aparte de esto último es que aunque se realizase, la mejora que proporciona es baja considerando que solo optimiza las búsquedas de máscaras, las cuales ya son eficientes en la actualidad por las estructuras de datos que utilizan como lo puede ser un map, por lo que este beneficio es pequeño en comparación al costo de transición.

P3. Ruteo

P3.1. Tablas de Rutas

1. Esto es así dado que garantiza que el paquete se dirija al destino más exacto posible dentro de las opciones disponibles, de forma que se minimiza la probabilidad de que este tome una ruta subóptima en la red.

2. Para realizar esta implementación asumiremos que cada entrada de `route_table` posee campos en los que almacena el prefijo (`prefix`), la máscara (`mask`) y una interfaz (`interface`), además de escribir directamente código en Python en vez de realizar pseudocódigo:

```
def find_route(IP_dest, route_table):
    for route in route_table:
        if (IP_dest & route.mask) == route.prefix:
            return route
    return None
```

   El ciclo `for` se asegura de recorrer las rutas desde las más específicas hasta las menos, mientras que el `if` verifica si las IPs coinciden según la máscara para así retornar la ruta con la cual rutear dicho paquete.

   Este algoritmo funciona ya que se encuentran primero las rutas más específicas, de modo que si hallo una coincidencia no puede existir una ruta más específica a la cual llegar después de esta.

P3.2. BGP-4

Considerando que la cantidad de Sistemas Autónomos no es tan grande en comparación a la de los prefijos, en teoría podría ser factible la idea de calcular mediante Dijkstra los caminos óptimos en un sistema centralizado, reduciendo la inconsistencia en decisiones locales entre Sistemas Autónomos, al evitar una elección de rutas subóptimas e inclusive potenciales bucles que podrían suceder entre estos, mejorando además el ruteo en conexiones a larga distancia.

Pese a esto, no es una solución perfecta ya que el sitio central tendría que ser capaz de recibir frecuentes actualizaciones en su red desde todos los Sistemas Autónomos, generando una alta carga de tráfico sobre este, además de que para cada una de estas actualizaciones se tendrían que recalcular rutas globales, dando lugar a un alto sobrecosto.

En contraste, el método implementado por BGP-4 es eficiente por la información local que comparte entre vecinos, mientras que un sistema centralizado tardaría bastante más en entregar dichos ruteos ya que necesita el tiempo para recibir las actualizaciones, recalcular las rutas óptimas y luego propagarlas a los routers, de forma que la latencia sería mucho mayor en comparación.

Además, un sistema centralizado no sería capaz de respetar todas las políticas de ruteo que presentan los distintos Sistemas Autónomos sin al menos incrementar enormemente la complejidad del cálculo.

Y por último, el mayor problema que presenta esta idea es que si este sitio central sufre algún error entonces se verá afectada toda la red global, quedando posiblemente bloqueada hasta que se resuelva.