DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Ataques de Denegación de Servicio

Seguridad de la Información

Primer Cuatrimestre de 2020

Integrante	LU	Correo electrónico
Balboa, Fernando	246/15	fbalboa95@gmail.com
López Valiente, Patricio	457/15	patriciolopezvaliente@gmail.com
Mattenet Riva, Pedro	428/15	pmattenet@gmail.com
Raffo, Leandro	945/12	leandrojavr@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

1. Introducción

1.1. Resumen

El presente trabajo tratará sobre los ataques informáticos de denegación de servicio (conocidos como *DoS* por sus siglas en inglés). Los mismos están dirigidos a degradar de forma notable la performance de un servicio para dificultar su uso, o directamente volverlo inutilízable. El caso más frecuente de este tipo de ataques es contra aplicaciones web, para impedir que usuarios de una página puedan acceder a ella. En muchos casos, la denegación se produce de forma distribuida, es decir, se utiliza más de una computadora para llevar a cabo el ataque. Estos casos se conocen como denegación de servicio distribuido o *DDoS*.

Además de los conceptos y datos relevantes relacionados, este trabajo cuenta también con una implementación de un programa que permite realizar un ataque *DDoS* a un servidor web, utilizando la técnica conocida como *Slowloris* y la aplicación *Telegram* para coordinar a los dispositivos atacantes.

1.2. Código

El código de los experimentos asociados a este trabajo se encuentra adjunto en el repositorio. Para ejecutarlo, es necesario instalar *Python 3* y las librerías necesarias. El código que ejecuta cada dispositivo atacante se encuentra en el archivo *attackerslowloris.py*.

Simulando el atacante en *Telegram*, se encuentran dos ejecutables de *Python*. *userbot.py* simula ser el celular coordinador del ataque. Para utilizarlo, hay que ingresar un número de celular, y al ejecutarlo le pedirá un código de autorización que *Telegram* envía a la aplicación de celular (TW Factor). Este ejecutable tiene como función mandar un mensaje a un canal en donde se encuentran los equipos infectados.
Por otra parte, *bot.py* simula ser un celular infectado. Es un bot que forma parte de un grupo, y ejecuta el ataque al recibir la dirección de la víctima.
Estos deben tener acceso a internet para funcionar, dado que *Telegram* lo requiere.

2. Casos e impacto en la industria

2.1. Mitigacion y prevencion de Slowloris

Slowloris es categorizado como una herramienta de ataques baja y lenta, porque este genera un ritmo lento de pequeños volúmenes de trafico, por lo que se vuelve difícil de detectar para los software de seguridad estándar de mitigación de *DDoS*. Sin embargo, existen diversas formas de protegerse de este tipo de ataques.
Entre las más sencillas se encuentra utilizar webservers que por su diseño mismo son mucho más resistentes a estos ataques, como *Hiawatha*, *Cherokee*, *Cisco CSS*, etc. u otros como *nginx* o *lighttpd* que no se ven afectados por este tipo de ataques.

Si se utiliza alguno de los webservers vulnerables existen formas de mitigar el impacto de este ataque. En general, estas estrategias pueden dividirse en tres categorías:

- **Aumentar la disponibilidad del server**: aumentar el numero máximo de clientes que el server permite al mismo tiempo indefectiblemente aumentará el número de conexiones que el atacante necesitará crear para realizar el *DoS*. Esta estrategia no suele ser muy efectiva, ya que en la práctica el atacante puede escalar fácilmente la cantidad de requests (sobretodo si el ataque es distribuido), independientemente del aumento en la capacidad del server.

- **Limitar la cantidad de requests entrantes**: restringir el acceso basándose en factores de uso ayuda a mitigar un Slowloris. Dentro de estas técnicas se encuentran diversos acercamientos como limitar el número máximo de conexiones que una misma IP puede hacer, restringir transferencias de baja velocidad, y limitar el tiempo máximo que un cliente tiene permitido estar conectado al server.

- **Protecciones basadas en la nube**: utilizar algún servicio que pueda funcionar como un proxy reverso, delegándole a este la responsabilidad de la protección del ataque, y de esta forma proteger el server.

Para los servers *Apache*, los cuales son muy populares en el mercado, y uno de los grandes afectados por este tipo de ataques, existen varios módulos que pueden mitigar *Slowloris*. Algunos de ellos son: *mod_limitipconn*, *mod_qos*, *mod_evasive*, *mod security*, *mod_noloris* y *mod_antiloris*. Todos ellos son sugeridos como recursos para reducir la probabilidad de éxito del ataque. Desde la versión 2.2.15, *Apache* ha marcado el módulo *mod_reqtimeout* como la solución oficial soportada por los desarrolladores.

2.2. Casos Reales

- **Sitios gubernamentales y afines iraníes en 2009:** durante una serie de protestas que ocurrieron en el país, se utilizo este tipo de ataque para denegar el servicio a varios sitios gubernamentales y portales de noticias oficialistas.
  Fue elegido *Slowloris*, ya que utilizar ataques de tipo *Flood* hubiera afectado también el acceso a internet de los protestantes.

- ***River City Media***: una variante de *Slowloris* fue usada para forzar a los servers de *Gmail* a mandar miles de mensajes en bulk al abrir miles de conexiones a la API de *Gmail* con requests de envío de mensajes, y luego completándolas todas a la vez.

3. Desarrollo

3.1. Conceptos

3.1.1. Servidores web

Para disponibilizar una página web, es necesario tener un servidor corriendo en una máquina y escuchando conexiones del protocolo *HTTP/S* en algún puerto (por lo general el 80/443). Dicho protocolo define el formato que deben tener los requests para ser válidos, incluyendo información como la versión, encabezados, el verbo *HTTP* o un cuerpo opcional.

Ante la llegada de un request, el servidor debe procesarlo y devolver una respuesta al cliente, consumiendo una cierta cantidad de recursos de la computadora. En las arquitecturas web más clásicas, el servidor retorna una respuesta en formato *HTML*. Si bien en los últimos años se ve una preferencia por las *Single Page Application (SPA)* con un backend que retorna texto en formato *JSON*, todavía son muy prevalentes aquellos que retornan *HTML* de forma directa. Al ser un lenguaje estructurado, el *HTML* es relativamente sencillo de parsear de forma programática para extraer información sobre la página actual o links a otras relacionadas a su dominio.

3.1.2. Caching

Se conoce como *caching* a la técnica de guardar un dato (por ejemplo, el resultado de un cómputo costoso) para poder devolverlo más rápido cuando es requerido. Muchas aplicaciones web utilizan este método en alguna capa para mejorar la performance del servidor.

Al realizar un ataque de denegación de servicio, es necesario tener en cuenta el *caching* a la hora de elegir qué recursos atacar. Podría suceder que la primera vez que se pide realizar una operación esta sea muy costosa, y por lo tanto un buen objetivo a explotar, pero si la aplicación utiliza una *cache* para guardar el resultado, pedidos subsecuentes no surtirán el mismo efecto deseado.

3.1.3. Tipos de DoS

El término ataque *DoS* se usa para abarcar un conjunto amplio de distintos tipos de ataques, todos con el fin de limitar la capacidad de responder *requests* de un servidor web, idealmente hasta al punto de inhabilitarlo. Originalmente, estos ataques se lograban a través de pedidos maliciosos, a partir de un *host* único propio del atacante.

En un principio, si un atacante tenía mayor ancho de banda que la víctima, este formato de ataque podía bastar para lograr que el ataque fuera exitoso, pero con el tiempo y la mejora en los servicios de los servidores, esto dejó de ser suficiente y así

surgieron los ataques *DDoS*. Estos simplemente consisten en los mismos principios, pero más elaborados, incluyendo el uso de una red de atacantes en comparación a un *Host* único, generalmente usando *Botnets*.

Este objetivo se puede cumplir de muchas maneras distintas, incluído algunas que aún actualmente no se conocen realmente. A continuación se presentan ejemplos a considerar en orden de complejidad, algunos en parte obsoletos, para poder comparar con el *Slow Loris*:

1. **UDP Flood**: inundar a una víctima con una gran cantidad de paquetes UDP dirigida a puertos azarosos donde haya una aplicación escuchando para eventualmente agotar los recursos del servidor .

2. **ICMP (Ping) Flood**: análogo a UDP flood, con la diferencia en el protocolo. Los paquetes ICMP no consumen necesariamente tanto ancho de banda como el ejemplo anterior, pero corren el riesgo de saturar también al atacante, ya que la víctima responde a los requests con múltiples *Echo Replies*.

3. **SYN Flood**: explota una vulnerabilidad de diseño del *Three-way handshake* del protocolo TCP. El atacante envía gran cantidad de mensajes SYN, con el fin de pretender iniciar una comunicación, pero nunca los reconoce con un ACK. La víctima se ve obligada a esperar una cantidad extensa de tiempo hasta dar de baja la conexión, pero en el proceso, se va agotando el *pool* de conexiones que puede mantener abiertas.

4. **Ping of Death**: enviar paquetes de IP maliciosos, que sean de gran tamaño con el fin de causar a que la *Link Layer* de la red de la víctima colapse. La idea es que al separar el paquete IP en demasiados *frames* de Ethernet, la víctima al intentar reconstruir los paquetes caiga e un *Buffer Overflow* por tener que mantener los *frames* parciales en memoria.

5. **NTP Amplification**: un poco más refinado, este ataque de DDoS consiste en tomar el control de servidores NTP para hacer consultas en dicho protocolo a la víctima. Se lo denomina ataque de *Amplification* porque la naturaleza del ataque recae en que las respuestas NTP de las víctimas requieren mucho más ancho de banda, que las consultas del atacante.

6. **Zero-day DDoS Attacks**: abarca toda serie de ataques desconocidos que explotan vulnerabilidades que hasta el día de hoy no hayan sido parcheadas.

La diferencia particular que separa al ataque *Slow Loris* de algunos de los mencionados hasta ahora, en especial los de tipo *Flood*, es que no requiere generar una cantidad enorme de tráfico del lado del atacante. Como el principio detrás de *Slow Loris* consiste

en mantener una conexión abierta con un servidor víctima por la mayor cantidad de tiempo posible, no se trata de inundarlo de tráfico, sino de enviarle la mínima cantidad de bytes como para que el servidor no cierre la conexión por *timeout*. Dentro de las ventajas que esto tiene, es que requiere mucho menos cómputo del lado del atacante, y en el caso de usar agentes infectados como atacantes, es menos probable que estos detecten comportamiento sospechoso de sus dispositivos. A su vez, es más complicado para los *firewalls* y servidores identificar que un ataque se está llevando al cabo, ya que un request que se demora mucho en completar podría no ser un atacante, y tan solo un cliente con una conexión inestable.

3.1.4. Botnet

Un *bot* es un programa que ejecuta tareas, en general relativamente simples y repetitivas, de forma automatizada. En el contexto de ataques informáticos, los *bots* suelen ser utilizados para mandar spam, analizar y recolectar data sensible de servidores web o lanzar ataques de denegación de servicio distribuidos.

La forma más usual de lanzar un *DDoS* es a través de una *botnet*, que no es más que una red de *bots* coordinados. En general, estos programas corren en dispositivos previamente infectados (conocidos como *zombies* por el atacante por algún medio, y son coordinados por otro programa que les envía mensajes de comando y control (*C&C*). Actualmente, existen muchos dispositivos *IoT* conectados que pueden ser candidatos para correr un *bot* malicioso sin que el dueño del mismo esté al tanto.

Un ataque *DDoS* que utiliza una *botnet* tiene dos grandes ventajas. La primera es que la cantidad de requests que se pueden hacer al mismo tiempo depende de la cantidad de dispositivos que los hacen y de su poder de cómputo. Si bien los *zombies* pueden variar ampliamente en hardware, en general son muchos, lo que amplifica el poder del ataque en comparación a un *DoS* normal. La segunda es que es más difícil de detectar, dado que los pedidos vienen de muchas direcciones IP distintas y no es sencillo determinar cuáles son requests de usuarios reales y cuáles no.

3.2. Herramienta

3.2.1. Arquitectura general

Se asume que varios dispositivos conectados a Internet han sido infectados y un archivo ejecutable de la herramienta se encuentra disponible, y la aplicación *Telegram* instalada. Desde una máquina coordinadora se enviará un mensaje de control a los dispositivos infectados para que inicien el ataque. Este incluirá los parámetros a usar en la herramienta. Cuando sea recibido, un script lo procesará y ejecutará el código que inicia el ataque, que se describe en la siguiente sección.

3.2.2. Código atacante

La implementación del ataque *DDoS* propuesta por el equipo es de tipo *Slowloris*. El código está escrito en el lenguaje *Python* e incluye algunos métodos para dificultar su detección.

El programa crea una cantidad configurable de sockets y establece conexiones *HTTP* con el servidor víctima. Los recursos a atacar son seleccionados y priorizados mediante el módulo conocido como *Crawler*, descrito más adelante. A cada *URL* candidato se le asigna una prioridad para ser seleccionado al establecerse la conexión mediante el socket.

Cada request utiliza un header *User-Agent* seleccionado de forma aleatoria de una lista predefinida para simular que provienen de dispositivos diferentes. Además, aproximadamente un 30 % de las conexiones completarán el request de forma normal para que los pedidos maliciosos se confundan con los de usuarios reales. El 70 % restante intentará mantener la conexión abierta la mayor cantidad de tiempo posible. Para lograr esto, la estrategia adoptada consiste en enviar indefinidamente un header custom inútil que acata el formato requerido por una comunicación *HTTP*, hasta que el servidor decida cortar la conexión. Dicho header se envía cada cierto intervalo regular configurable (por default 30 segundos) para que la conexión no se corte por timeout.

3.2.3. Crawler

Un Crawler es una herramienta utilizada para navegar de forma automática a través de la Web y recolectar información sin necesidad de un browser común. Un caso de uso muy frecuente es el de motores de búsqueda, que utilizan mucho este tipo de herramientas. Los Crawlers también puede ser utilizados para los ataques de *DoS*, dado que permiten realizar una análisis previo de la víctima para recolectar datos que pueden ser útiles en el ataque.

Un Crawler es un bot cuyo trabajo consiste en hacer *scrapping* de páginas web, para encontrar datos o hipervínculos a otras páginas, sea en el código *HTML*, o el propio

Javascript, con la posibilidad de navegar a través de ellos. Por lo general sigue alguna lógica en particular, y analiza siguiendo criterios específicos, ya que el volumen de datos con los que trabaja (la web) es demasiado grande.

En el caso de este trabajo, el Crawler impementado se limita a navegar por recursos dentro del dominio al cual se busca atacar, y a medida que navega va categorizando cada pedido en función de la duración del mismo. Cada URL se registra en su forma canónica para evitar ciclar infinitamente por el dominio, por ejemplo, removiendo parámetros de URLs antes de marcarlas como visitadas.

El objetivo detrás de este análisis previo es encontrar armar un ranking de demora de todas las consultas que puede responder el servidor web. En este trabajo se asume la hipótesis de que el tiempo de respuesta a un request está directamente relacionado con la cantidad de recursos del servidor que se consumen para responderlo. Esto no es necesariamente cierto ya que hay muchos factores que pueden afectar esta variable, como por ejemplo la congestión de la red. A grandes rasgos, se interpreta el tiempo como el cómputo requerido del servidor, asociándolo a consultas internas no optimizadas, lecturas a disco, u otras operaciones que no necesariamente dependan del ancho de banda de la víctima. Con esta información adquirida, se busca entonces saturar la cantidad de conexiones de la víctima, priorizando atacar las URLs que el servidor tarda en responder.

Algunos detalles del proceso de implementación del Crawler que se incluyeron son los siguientes:

- **URLs canónicas:** ya que el espacio de búsqueda puede ser muy vasto, para mejorar la política de *selection*, se llevo el registro de páginas visitadas a partir de una versión canónica de las URLs. El espacio de URLs canónicas es mucho más acotado que el espacio completo de URLs adentro del dominio. Este proceso consistía en podar URLs donde solo difieren los parámetros.

- **Páginas Cacheadas:** hay casos de búsqueda donde el Servidor Web almacena en cache, las respuestas a los requests realizados previamente. Esto hizo completamente irrelevante algunos de los resultados del Crawler, ya que daba falsos positivos cuando se hallaban solicitudes que en un principio demoraban una buena cantidad de tiempo. La solución algo simplista, fue duplicar cada una de las consultas y evaluar si la diferencia de tiempo superaba una relación de 10:1. Donde esa comparación daba positiva, el request previamente almacenado, se descartaba.

Cabe aclarar de todas formas, que para el caso de un ataque *DoS* del tipo *Slow Loris*, no es necesariamente relevante tiempo de respuesta de un request HTTP. Esto se debe a que el objetivo no es inundar de pedidos a la víctima, sino mantener una gran cantidad de conexiones abiertas por la mayor cantidad de tiempo posible. Sin embargo,

la herramienta implementada también realiza requests reales en un intento de pasar desapercibida, por lo que atacar recursos no optimizados sigue aportándole valor al ataque.

3.2.4. Comunicación por Telegram

La herramienta implementada en este trabajo incluye programas para lograr la comunicación de comandos por *Telegram*, con el objetivo de crear una *botnet* para realizar el ataque *DDoS* propuesto. Existen dos tipos de bots de *Telegram*:

- **Bots normales:** estos son los bots que se crean mediante el bot conocido como @BotFather (provisto por *Telegram*), el cual genera un token único por cada bot. La aplicación identifica a estos como bots para el usuario final, y tienen algunas restricciones. Por ejemplo, no pueden unirse a canales ni hablar con otros bots, y sólo pueden responderle a usuarios que les hablan.
- **User bots:** *Telegram* permite crear aplicaciones sobre su API como *Telegram-Desktop* o las versiones de *Android/iOS*. De esta forma pueden crearse bots que a priori no pueden distinguirse de usuarios reales, ni tienen las restricciones de los bots normales.

Aunque estos Bots podrían crearse de forma automática con un Userbot, existe un límite de 20 Bots por número de teléfono, por lo cual se requerirían muchos números para generar una *botnet* de tamaño razonable.

En el esquema propuesto, un Userbot actúa como puente entre el atacante y la *botnet*. Esta a su vez es un conjunto de máquinas infectadas, cada una con su propio Bot normal. Cada Bot le avisa al Userbot o atacante su id para que quede registrado un mapa de la *botnet*.

Una vez finalizado el proceso de registro, el atacante de alguna manera le manda un mensaje al Userbot (por ejemplo a través de un socket de comunicación normal) y este a su vez ordena a los Bots normales que inicien el ataque mediante un comando por *Telegram*. Otra opción podría ser que el atacante realice estas acciones desde su celular y con la aplicación de *Telegram*. Una vez que conoce el Username o id de los Bots, este podría crear un canal al cual agrega a los Bots, y de esta forma puede enviarles el objetivo a atacar.

3.2.5. Testing

Para testear que el ataque realmente funciona, se utilizó un setup de dos máquinas virtuales conectadas en modo *host-only*, donde una es la víctima y la otra el atacante. Para la primera, se eligió la VM *Metasploitable2*, que al levantarse corre automáticamente un servidor web Apache vulnerable. El ataque se ejecutó desde la VM *Kali* hacia el

recurso */mutillidae/*. El resultado esperado es bastante inmediato. Luego de escanear el dominio, que es muy pequeño, el pool de conexiones se agotó rápidamente dado que no son muchas y el server dejó de responder, demostrando que el ataque fue exitoso.

El ataque fue probado muy brevemente contra otras páginas web más interesantes y también logró atacar la disponibilidad hasta el punto de volverlas inutilizables a los pocos segundos. El ataque terminó inmediatamente al cortar la ejecución del programa y los servidores pudieron volver a responder de forman normal.

Para probar el correcto funcionamiento de la comunicación por Telegram, los pasos son:

- Levantar el Userbot de *Telegram* e ingresar el número de teléfono para unirlo al canal. Luego de ingresarlo le va a pedir un código de confirmación que debería llegar por su app de Telegram.[*python3 userbot.py*]
- Levantar el Bot normal de *Telegram* para que escuche los mensajes del canal. [*python3 bot.py*]

Luego de ingresado estos 2 datos el Userbot va a conectar automáticamente la cuenta del usuario a un canal de *Telegram* donde previamente se agregó el Bot, y va a mostrar un shell con opciones para mandar el ataque.

Por último, se incluyen dos ejecutables del Bot, tanto para Linux como para Windows, ambos de de 64 bits. Estos son los archivos que se distribuirían para la infección.