Documentación Técnica.

Para implementar IRCBot se utilizaron las siguientes clases en el lenguaje Java.

IRC:

Implementa un cliente IRC mediante los siguientes métodos:

conect():

Este método es el encargado de conectarse al servidor IRC.

forThisBot(String cmd):

Determina si la instrucción en el canal IRC es para este bot, o para todos.
Donde una instrucción tiene la siguiente forma: <<{nickbot}>>
!@{comando} arg1 arg2 arg3 donde:

nickbot: es el nombre del bot que queramos que realize la instrucción.
Si queremos que todos los bots ejecuten la instrucción omitimos este parámetro.
{comando}: es el comando que se ejecutara.
arg1, arg2, arg3: son los argumentos al
comando.

Por ejemplo: Si queremos que el bot W7_abcdefghij_128 liste los archivos en C:
La instrucción quedaría asi:

<<W7_abcdefghij_128>> !@dir C:\\

Y si queremos que todos los bots listen los archivos en C:
La instrucción seria:

!@dir C:\\

init():

Este método se encarga de definir los valores que usara este bot para conectarse a IRC.
Aquí se define el puerto, los objetos de lectura y escritura y las credenciales de acceso.

joinChannel():

Realiza la conexión a un canal previamente definido.

quitIRC():

Termina la conexión IRC.

ReadLines():

Lee los mensajes del canal IRC, de esta forma obtiene las instrucciones que ejecutara

OSUtils:

Esta clase se encarga de definir identificadores para los bots, dependiendo de su sistema operativo, mediante los siguientes métodos:

getID():

Regresa un identificador único para el bot obteniéndolo de alguna característica de su sistema operativo o del Hardware.

getLinuxID():

Regresa un identificador unico para los bots de tipo Linux o Unix(BSD,etc).

getMACID():

Análogo para MACs.

getOSName():

Obtiene el nombre del sistema operativo sobre el cual corre el bot.

getWindowsID():

Obtiene un identificador único para bots tipo Windows.

isMac():

Regresa true si el bot es una MAC, falso en otro caso.

isUnix():

Análogo.

isWindows():

Análogo.

CommandParser:

Es la clase encargada de ejecutar las instrucciones pasadas por los usuarios.

parseCommand(String cmd):

Este método recibe una instrucción de la forma !@{command} y, dependiendo de command regresa un entero que servirá para identificar ese comando.

getParam(String cmd,int n):

Obtiene el n-esimo parámetro de una instrucción, es decir recibe una instrucción de la forma !@{command} file arg1 arg2 arg3 y regresa el argumento n.

Por ejemplo:

Si cmd= **!@**cifraByteXOR C:\Users\admin\Desktop\test.txt 4

Hacer getParam(cmd,2) regresa 4.

getParams(String cmd):

Regresa una lista con todos los parámetros de la introducción que recibe, estos se delimitan por medio de un espacio.

Ejemplo:

Si cmd="**!@**cifraByteXOR C:\Users\admin\Desktop\test.txt 4"

Hacer getParams(cmd) regresa [cifraByteXOR , C:\Users\admin\Desktop\test.txt , 4]

paramToInt(int n,String param):

Recibe un valor por default y un parámetro param.

Si param es un entero regresa param, sino regresa n.

Este método es utilizado para definir valores por default si es que el usuario ingresa valores no permitidos.

Ejemplo:

paramToInt(5,”8”) regresa 8

paramToInt(5,”cero”) regresa 5. **getFile(String cmd):**

Recibe un comando y regresa la ruta del archivo sobre el que actuara dicho comando.

Ejemplo:

Si cmd= ”**!@**cifraByteXOR C:\Users\admin\Desktop\test.txt 4”

Hacer getFile(cmd) regresa C:\Users\admin\Desktop\test.txt

doCommand(String cmd):

Recibe una cadena de la forma **!@command** file arg1 arg2 arg3

Detecta de que instrucción se trata usando el método parseCommand.

Obtiene los parámetros usando getParams()

Ejecuta el comando con los parámetros obtenidos si el metodo parseCommand regreso un numero distinto de -1.

Ejemplo:

cmd=!@dir C:\\

getParam regresa 12.

Obtengo el archivo con getFile = C:\\

Ejecuto el comando.

execute(String cmd):

Este método es usado cuando el método parseCommand regresa -1, entonces se asume que es un comando nativo y se intenta ejecutar nativamente.

Por ejemplo:

cmd=!@notepad

En este caso getParam(cmd) regresa -1

Entonces se ejecuta este método que abre una nueva ventana del bloc de notas.

Si el comando no es reconocido de manera nativa el programa en el bot continua con su ejecución de manera normal.

Cipherx:

Es la clase encargada de cifrar los archivos pasados en los comandos utilizando los métodos:

cifraBinRot(String f,int k)

Toma la ruta del archivo y lo sobrescribe rotando su información binaria k posiciones.

descifraBinRot(String f,int k)

Toma la ruta del archivo cifrado y lo sobrescribe rotando su información binaria k posiciones.

cifraByteRot(String f,int k):

Análogo a nivel de bytes.

descifraByteRot(String f,int k)

Análogo a nivel de bytes.

cifraByteXOR(String f,int k):
Toma la ruta del archivo y lo sobrescribe haciendo xor k byte a byte.

descifraByteXOR(String f,int k)
Inversa al método anterior (de hecho hace exactamente lo mismo).

cifraByteRotXOR(String f,int rot,int k):
Toma la ruta de un archivo, un parámetro de rotación y una llave k para hacer XOR. Entonces sobrescribe el archivo después de rotarlo k posiciones y hacerle XOR con k.

descifraByteRotXOR(String f,int rot,int k)
Análogo al metido anterior.

cifraTransposicionInversa(String f)
Toma la ruta de un archivo y lo sobrescribe usando los bytes en el orden inverso.

descifraTransposicionInversa()
Análogo al método anterior.

getDESKey():
Regresa una llave para cifrar usando DES.

cifraArchivosDES(String f,SecretKey k)
Toma una ruta de archivo y una llave DES, sobrescribe el archivo con la información original cifrada usando DES.

descifraArchivosDES(String f,SecretKey k):
Análogo al método anterior.

fixPass(String p):
Toma una cadena de longitud variable y regresa los primeros 8 caracteres.

Archivos

Clase para manipular archivos: escribir, leer, descargar y eliminarlos.

read(File f):
Recibe la ruta de un archivo y regresa una cadena con el contenido de este.

readBytes(File f)
Recibe la ruta de un archivo y regresa un arreglo con los byte de este.

write(String filename,String content):
Recibe una ruta para el archivo, y una cadena, genera un nuevo archivo con la información de content en la ruta parámetro.

writeBytes(byte[] bytes,String filename):

Recibe una ruta para el archivo, y un arreglo de bytes, genera un nuevo archivo con la información del arreglo de bytes.

bytesToBase64(byte[] bytes):

Toma un arreglo de bytes y regresa la representación de este en base64.

base64ToBytes(String b64):

Toma una cadena en base64 y regresa el arreglo de bytes representado por la cadena.

download(String url,String file):

Toma una dirección de un archivo en internet y una ruta de archivo local.
Descarga el archivo url y lo guarda en la ruta local pasada como parámetro.

list(String ruta):

Toma una dirección local en el sistema regresa una lista de archivos en ese directorio.

delete(String f):

Toma la ruta de un archivo y lo elimina.

Utils

Esta clase define métodos auxiliares para las demás.

rotaBits(String archivo,int nshifts):

Toma la ruta de un archivo y el numero de posiciones a rotarlo a nivel de bits a la derecha.
Regresa un arreglo de bytes con los bits rotados.

desrotaBits(String archivo,int nshifts):

Análogo al anterior, pero rota los bits a la izquierda.

sendList(BufferedWriter writer,String result,String channel, String nick)

Este método es usado por la clase IRC.
Se implemento debido a que en IRC la longitud máxima de un mensaje es de 500 caracteres, entonces si el resultado de ejecutar un comando es mayor, el resto no podrá enviarse. En particular el problema surge del comando dir (si la carpeta tiene muchos archivos y estos tienen un nombre muy largo).

Para solucionar esto tenemos 2 opciones:
1) Enviar un archivo por cada linea. En este caso tendríamos que enviar n lineas, pero ocurre el riesgo de que el servidor IRC banee al bot por floodding.

2)Esta forma soluciona el problema original y soluciona el posible baneo por floodding.
Se obtiene el resultado del comando en una cadena.
Se divide esta cadena en grupos de 500 caracteres y se envían estos grupos uno por mensaje, de esta manera, en lugar de enviar n lineas, ahora se envían (m/500) + (m%500) mensajes

donde m es el numero de caracteres que tiene el resultado, evitando de esta manera el posible baneo por floodding.

Lógica de Implementación

El bot lo primero que hace es conectarse a IRC y unirse al canal utilizando la clase IRC, poniéndose a escuchar el canal con el comando readLines, especificamente:

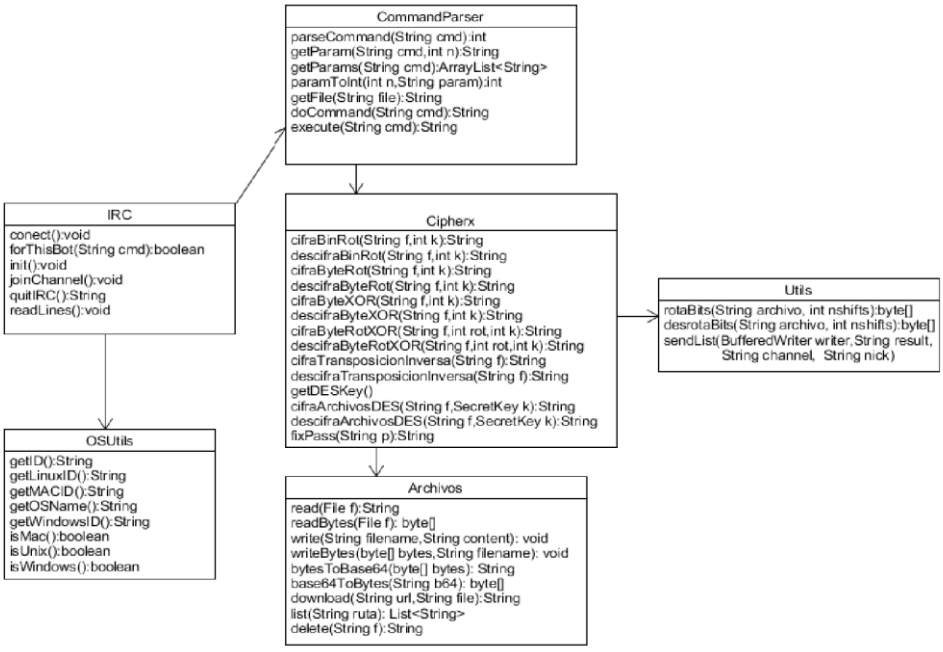
[1] El bot inicia ejecutando el método init de la clase IRC.
[2] Se conecta al servidor IRC
[3] Se une al canal
[4] Escucha el contenido del canal
[5] Analiza las lineas: contestando si recibe un ping y ejecutando el comando si va dirigido a el (esto se verifica utilizando el método forThisBot).
[6] Si es un comando para el, lo ejecuta utilizando doCommand de la clase CommandParser.[7] Continua haciendo esto hasta que recibe el comando quit.

Características de Implementación:

Agregar nuevos comandos es muy flexible ya que,no implica cambiar el código de los anteriores.
Para agregar un nuevo comando solo es necesario:

Definir el método en la clase correspondiente.
Agregar su firma (identificador) al arreglo cmds del método parseCommand.
Agregar la llamada al método en el switch del método doCommand.

A continuación se muestra un diagrama con las clases implementadas para IRCBot (el diagrama de clases para BotAdmin es un subdiagrama con la clase IRC).



Proyecto 3
Criptografía y Seguridad
Facultad de Ciencias,UNAM
Gonzalo Vázquez Cruz
413038357

Cosas por implementar:
Agregar persistencia.
Agregar un enmascaramiento.
Agregar comandos: killProcess,DNSPoisoning,etc.

Fuentes Consultadas: http://www.codejava.net/coding/file-encryption-and-decryption-simple-example https://en.wikipedia.org/wiki/Padding_%28cryptography%29
http://stackoverflow.com/questions/1385866/java-run-as-administrator