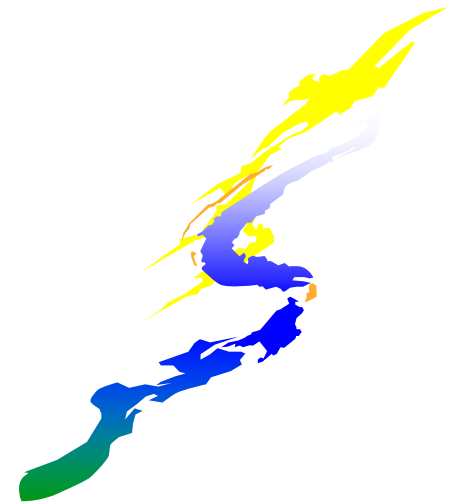


# Finding debugging clues in LE dumps



Thomas Petrolino  
[tapetro@us.ibm.com](mailto:tapetro@us.ibm.com)  
IBM Poughkeepsie  
Session 16158



# Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

- CICS®
- DB2®
- Language Environment®
- OS/390®
- z/OS®

\* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

\* All other products may be trademarks or registered trademarks of their respective companies.

## Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

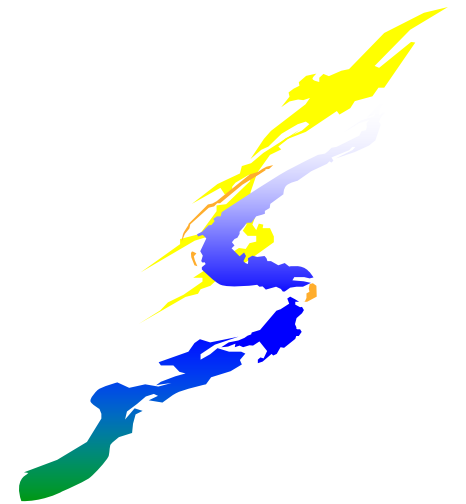
Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

# Agenda

---

- CSI Training
- The Crime is committed
- Collecting the Clues
- Analyzing the Crime Scene
- Back in the Lab
- Forcing LE to Spill the Beans
- Sources of Additional Information
- Appendix
  - Programs
  - Really Cool Lab Equipment

# Investigator training



# Major clues to look for (ABENDs)

- U4038 A severe (unhandled) error occurred.
- U4039 A severe (unhandled) error occurred AND LE took a system dump.
  
- U4083 Save area back chain error
- U4087 Error during condition processing
- U4093 Error during initialization
- U4094 Error during termination

# Major clues to look for (ABENDs)

- U4038/U4039
  - A severity 2 or greater condition has gone unhandled
  - Application will terminate (gracefully)
  - Clues will be gathered and collected if requested
    - TERMTHDACT controls amount and type of clues
    - CEEDUMP and/or DYNDUMP can be collected

# Major clues to look for (ABENDs)

- U40xx
  - Application will terminate immediately
  - TERMTHDACT does NOT control what clues are gathered
  - No CEEDUMP is generated
  - DYNDUMP can be used (3<sup>rd</sup> suboption) to collect clues

# Major clues to look for (messages)

- Message (and module) prefixes
  - CEE      CEL (but may be reporting errors elsewhere)
  - IGZ      COBOL
  - IBM      PL/I
  - FOR      Fortran (also AFH)
  - EDC      C/C++

See z/OS Language Environment Run-Time Messages



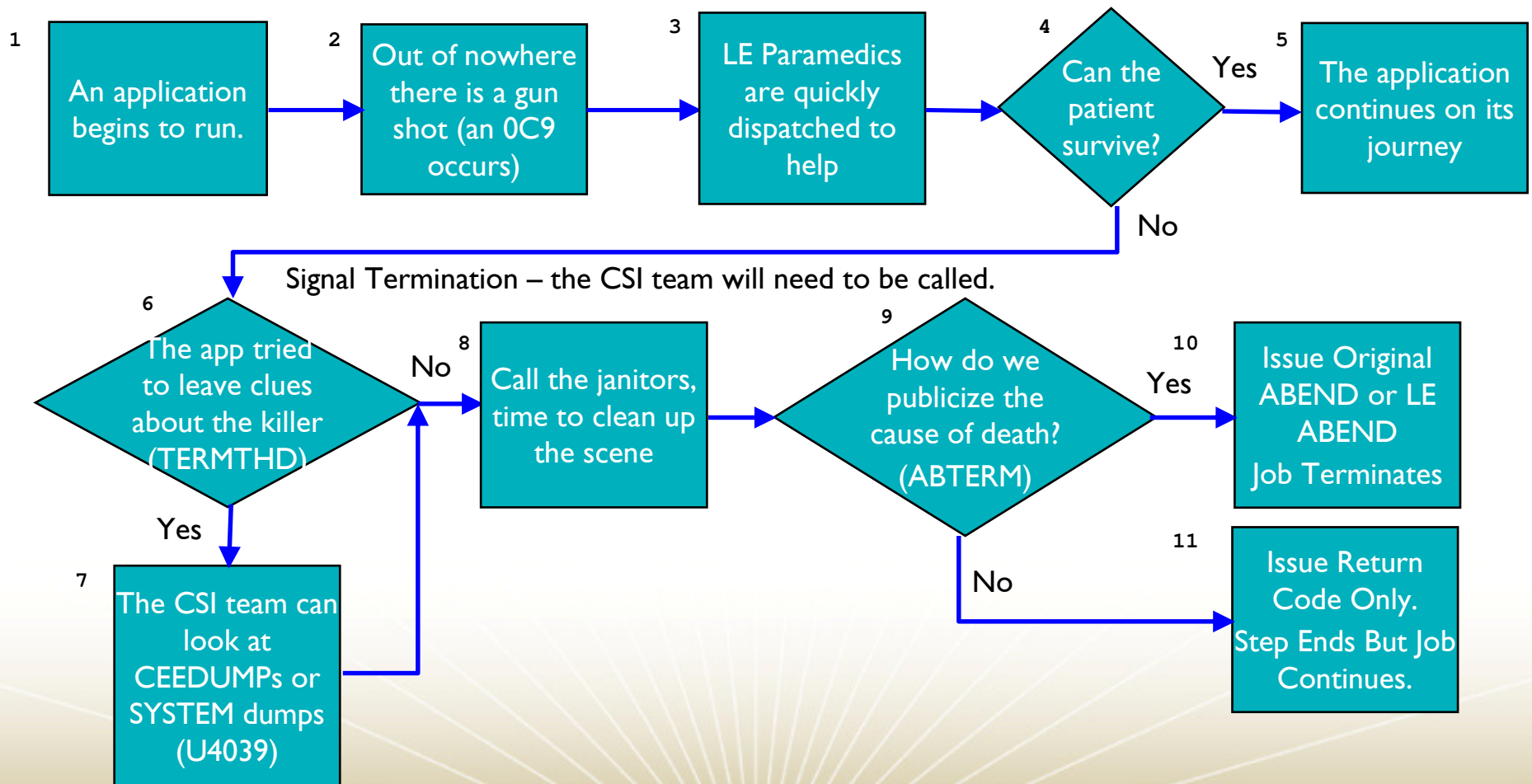
# Some DNA mapping

- Condition Token (Feedback Code)
  - Example: 00030C89 59C3C5C5 xxxxxxxx
    - 0003 | 0C89 | 59 | C3C5C5 | xxxxxxxx
      - 0003           Severity
        - 0000           Informational (I)
        - 0001           Warning (W)
        - 0002           Error (E)
        - 0003           Severe (S)
        - 0004           Critical (C)

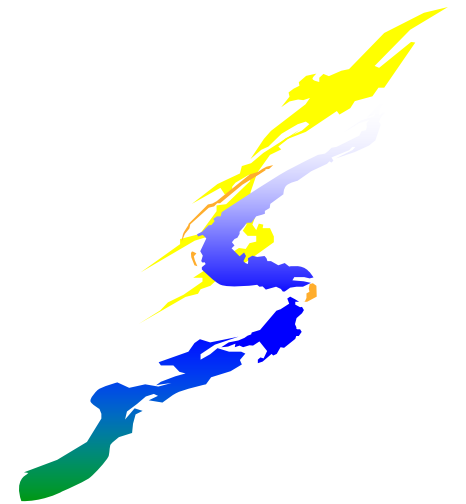
# Some DNA mapping

- Condition Token (Feedback Code)
  - Example: 00030C89 59C3C5C5 xxxxxxxx
    - 0003 | 0C89 | 59 | C3C5C5 | xxxxxxxx
      - 0003           Severity (**S**)
      - 0C89           Hex message number (**3209**)
      - 59             Flags (ignore)
      - C3C5C5        Hex (EBCDIC) facility ID (message prefix) (**CEE**)
      - xxxxxxxx     Instance specific info (internal)
  - This token represents message CEE3209S

# Timeline of a crime



**The crime is  
committed**

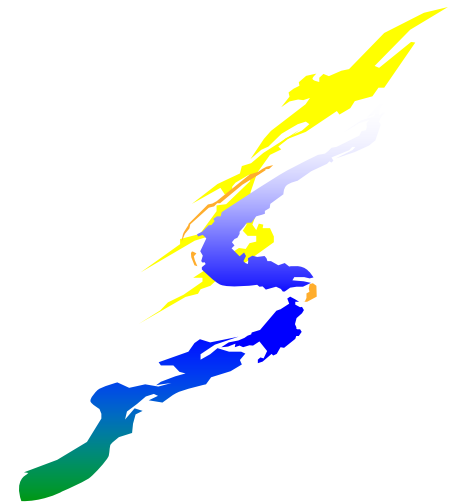


# The criminal application is written

- A COBOL program (COBOLED1) calls
- A 2<sup>nd</sup> COBOL program (COBOLED2) which calls
- A C program (CPROG3)
- CPROG3 divides by zero!

See appendix for these programs

# Collecting the clues



# Major Sources of Evidence

- CEEDUMP
- LEDATA IPCS Verb Exit
- Other Language Environment-produced reports
  - Options Report
  - Storage Report
  - Heap Storage Diagnostics Report

# Major Sources of Evidence

- Getting useful information
  - Use Language Environment run-time option TERMTHDACT() to request Language Environment take a dump
    - DUMP           CEEDUMP with storage
    - TRACE         CEEDUMP with traceback only
    - **UADUMP   CEEDUMP, system dump via U4039**
    - UAONLY       No CEEDUMP, system dump via U4039
    - UATRACE     CEEDUMP (traceback) and system dump via U4039
    - (UAIMM)      *System dump via original error (only for debug purposes), also TRAP(ON,NOSPIE)*



# CEEDUMP (the picture)

- Formatted dump produced by Language Environment
  - Failure information, traceback, control blocks, heaps, run-time options report
- Written to CEEDUMP data set:
  - CEEDUMP DD if allocated
  - Dynamically allocated if not available based on CEEDUMP run-time option
    - `CEEDUMP(60,SYSOUT=*,FREE=END,SPIN=UNALLOC)`
- CICS written to CESE Transient Data Queue
  - With `TERMTHDACT(,CICSDDS)` written as part of the CICS dump data set (CICS transaction dump)



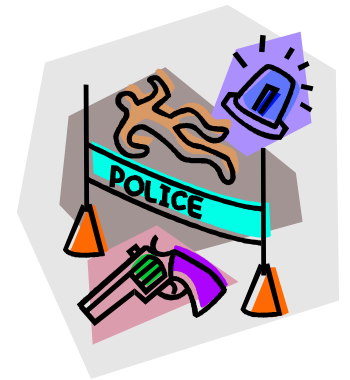
# CEEDUMP (the picture)

- Advantages
  - Immediately available and readable
  - Can provide a lot of information (with the right set of compile and run-time options)
- Disadvantages
  - Snapshot of the crime scene
    - May not contain all clues necessary to solve the crime



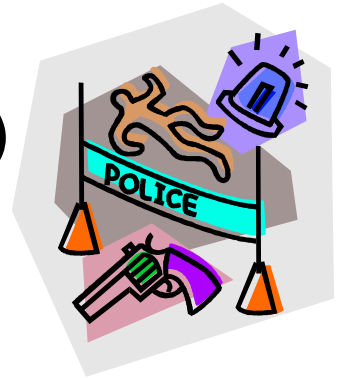
# IPCS readable dump (the body)

- IPCS support to format and analyze data in a system dump
- Options to generate numerous reports
  - CEEDUMP information and more
- System dump generated:
  - For an unhandled condition of severity 2 or greater
    - TERMTHDACT(UADUMP/UATRACE/UAONLY)
      - with SYSMDUMP DD
      - DYNDUMP(hlq,DYNAMIC,TDUMP)
  - When CEE3ABD is called by the application
  - Using system mechanisms (SLIP, Console Dump)



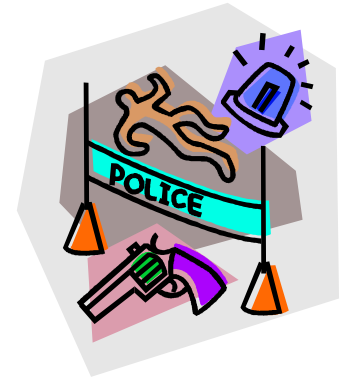
# IPCS readable dump (the body)

- DYNDUMP run-time option
  - DYNDUMP(hlq,U4039-ABEND,U40xx-ABEND)
    - hlq
      - \*USERID or \*USERID.hlq
      - \*TSOPREFIX or \*TSOPRE
        - (also \*TSOPREFIX.hlq or \*TSOPRE.hlq)
      - Up to 26 characters of an MVS data set name



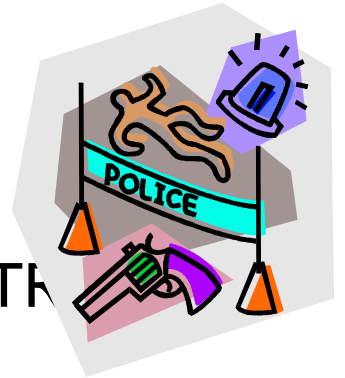
# IPCS readable dump (the body)

- DYNDUMP run-time option
  - DYNDUMP(hlq,U4039-ABEND,U40xx-ABEND)
    - U4039-ABEND
      - NODYNAMIC (default)
        - DYNDUMP turned off for U4039 ABENDs
      - DYNAMIC
        - DYNDUMP active for U4039 ABENDs if no SYSMDUMP, SYSUDUMP or SYSABEND DD.
      - FORCE
        - DYNDUMP active for U4039 ABENDs even with above DDs allocated
      - BOTH
        - You want it all!!!



# IPCS readable dump (the body)

- DYNDUMP run-time option
  - DYNDUMP(hlq,U4039-ABEND,U40xx-ABEND)
    - U4039-ABEND
      - TERMTHDACT MUST be set to UADUMP, UATH, or UAOONLY to generate a U4039.
      - U4038 does not produce a dump!
      - Example: DYNDUMP(JMONTI,FORCE,TDUMP)



```
+CEE3798I ATTEMPTING TO TAKE A DUMP FOR ABEND U4039 TO DATA SET:  
JMONTI.D201.T1336225.JMONTI@B
```

```
IEA822I COMPLETE TRANSACTION DUMP WRITTEN TO  
JMONTI.D201.T1336225.JMONTI@B
```

```
+CEE3797I LANGUAGE ENVIRONMENT HAS DYNAMICALLY CREATED A DUMP.
```

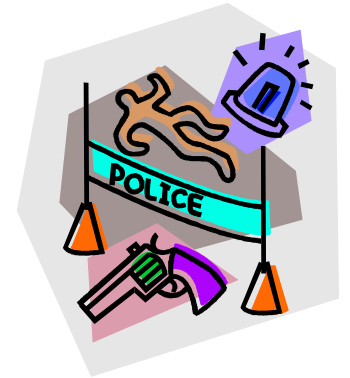
# IPCS readable dump (the body)

- DYNDUMP run-time option
  - DYNDUMP(hlq,U4039-ABEND,U40xx-ABEND)
    - U40xx-ABEND
      - TDUMP (Default)
        - DYNDUMP is active for all U40xx ABENDs (other than U4039) which request a dump.
      - NoTDUMP
        - DYNDUMP is not active for U40xx ABENDs



# IPCS readable dump (the body)

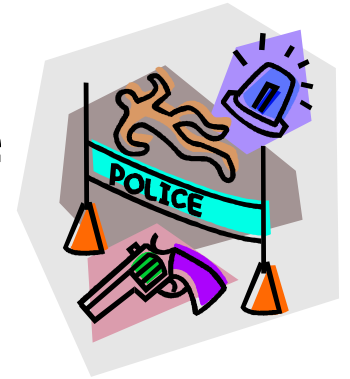
- DYNDUMP run-time option
  - Not honored for CICS
  - Use CEMT to request system dumps in CICS
    - CEMT SET TRD(40xx) SYS ADD



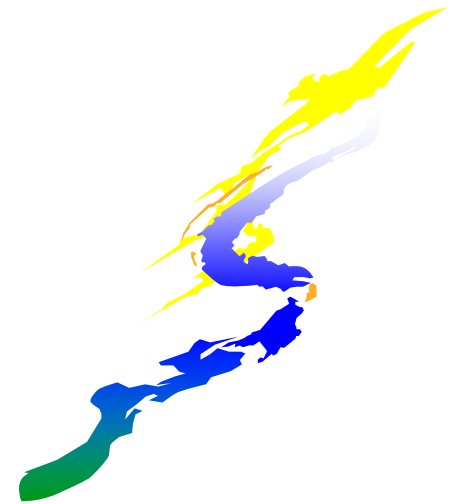


# IPCS readable dump (the body)

- Advantages
  - More complete picture of the crime scene
  - IPCS tools available for debugging
- Disadvantages
  - Additional skills required for analysis
  - Application programmers may not have access to system dumps and/or IPCS
  - Sometimes have to get your hands dirty (with bits and bytes)

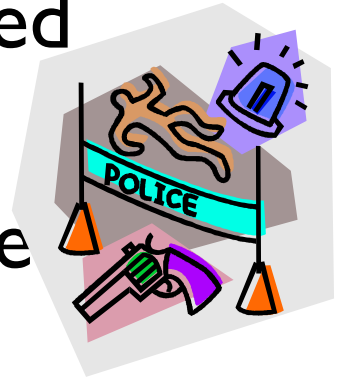


# False Leads!

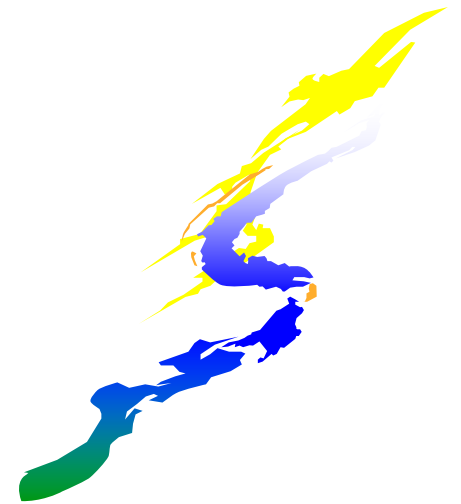


# Don't be fooled!

- Don't SLIP on Language Environment reissued ABEND (eg, 0C4)
  - Remember this is taking a picture after the crime scene has been cleaned up!



# Analyzing the Crime Scene



# Messages are the first clue

- Examine the Language Environment message file (usually SYSOUT)

CEE3209S The system detected a fixed-point divide exception (System Completion Code=0C9).

From compile unit POSIX.RTL.UT29.SRC(CPROG3) at entry point CPROG3 at statement 8 at compile unit offset +0000008C at entry offset +0000008C at address 219C60FC.

- Message tells us what the crime was and where it was committed!
  - A divide by zero occurred at offset x'8C' in CPROG3 (statement 8)!

# Getting a better picture

## The header section of the CEEDUMP

```
CEE3DMP V1 R12 0 Condition processing resulted in the unhandled condition 07/20/10 1:36:25 PM  
ASID: 0180 Job ID: JOB15785 Job name: JMONTI@B Step name: GO UserID: JMONTI
```

```
CEE3845I CEEDUMP Processing started.
```



# Getting a better picture

## The traceback section of the CEEDUMP

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit
1	CEEHDSP	+000041FA		CEEPLPKA	CEEHDSP
2	CPROG3	+0000008C	8	CPROG3	CPROG3
3	@@FECB	+00312B7C			
4	@@DB2C	+00000102		CEEEV003	
5	IGZCFCC	+000002C0		IGZCPAC	IGZCFCC
6	COBOLED2	+000004C0	28	COBOLSHR	COBOLED2
7	COBOLED1	+0000037A	10	COBOLSHR	COBOLED1

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	2199D768	217C8180	217C8180	+000041FA	20100319	CEL
2	2199D6D8	219C6070	219C6070	+0000008C	20070212	C/C++
3	2199D5E8	219BDE40	219BDE40	+00312B7C		LIBRARY
4	2199D560	21C504F0	21C504F0	+00000102		LIBRARY
5	2199D380	21922548	21922548	+000002C0	20100316	LIBRARY
6	2199D1D8	21702220	21702220	+000004C0	20100720	COBOL
7	2199D030	21700000	21700000	+0000037A	20100720	COBOL

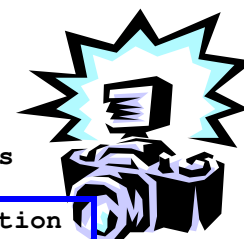
**Fully Qualified Names**

DSA	Entry	Program Unit	Load Module
2	CPROG3	POSIX.RTL.UT29.SRC (CPROG3)	CPROG3

Service Status  
HLE7770 Call

Exception

Call  
Call  
Call  
Call  
Call



**Status**  
**Why I left this entry point**

# Getting a better picture

## The condition information section of the CEEDUMP



### Condition Information for Active Routines

Condition Information for POSIX.RTL.UT29.SRC(CPROG3) (DSA address 2199D6D8)

CIB Address: 2199E060

Current Condition:

CEE0198S The termination of a thread was signaled due to an unhandled condition.

Original Condition:

CEE3209S The system detected a fixed-point divide exception (System Completion Code=0C9).

Location:

Program Unit: POSIX.RTL.UT29.SRC(CPROG3)

Entry: CPROG3 Statement: 8 Offset: +0000008C

Machine State:

```

ILC..... 0002      Interruption Code..... 0009
PSW..... 078D2400 A19C60FE
GPR0..... 00000000_00000000  GPR1..... 00000000_0000000A  GPR2..... 00000000_A1CD09BC
GPR3..... 00000000_219C60B8
GPR4..... 00000000_2199D2D8  GPR5..... 00000000_21F91A00  GPR6..... 00000000_21F92AC8
GPR7..... 00000000_219BDE40
GPR8..... 00000000_A19C63A8  GPR9..... 00000000_21F93368  GPR10.... 00000000_A19C6070
GPR11.... 00000000_A19C60A0
GPR12.... 00000000_21713B58  GPR13.... 00000000_2199D6D8  GPR14.... 00000000_00000000
GPR15.... 00000000_00000006
  
```

Storage dump near condition, beginning at location: 219C60EC

```

+000000 219C60EC D0848910 00011B01 41E00006 8EE00020 1DE0180F 5000D080 4400C1AC 47F03058
|.di.....&.....
  
```





# Getting a better picture

## The storage around regs section of the CEEDUMP



```
Storage around GPR15(21922548)
-0020 21922528 F0F3F1F6 F2F0F4F4 F0F0F0F3 F0C3F0F0 00074040 4040
+0000 21922548 47F0F028 00C3C5C5 000001E0 00000D78 47F0F001 0000
+0020 21922568 00000000 0000D500 90ECD00C 18BF1841 58A09058 58C0
```

### Local Variables:

```
18 01 WS-VARS          AN-GR
19 02 WS-COMP1        S9999 COMP      +00010
20 02 WS-COMP2        S9999 COMP      +00000
21 02 WS-COMP3        S9999 COMP      +00032
22 01 DYN-NAME        X(8) DISP      'CPROG3'
```

- To obtain Local Variables you must use the TEST compiler option

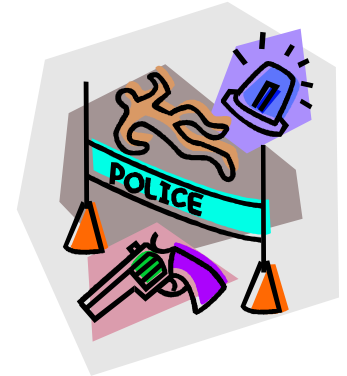


# Even more clues!

- CEEDUMP also contains
  - Run-time options report
  - COBOL working storage
  - Language Environment control blocks
  - Heap storage
  - Language specific information
  - Any much more
  
- But it is only a picture of the crime

# Examining the body

- IPCS Readable dumps
  - A bigger better picture of the crime
  - Use VERBX LE DATA
    - ‘Summary’                      general info and run-time options
    - ‘CEEDUMP’                      traceback similar to CEEDUMP
    - ‘CM’                                Condition management
    - ‘SM’                                Storage Management (Stacks/Heaps)
    - ‘HEAP’                            Heap data
    - ‘STACK’                          Stack data
    - ‘NTHREAD’                      traceback for all threads
    - ‘ALL’                                All the output



# Examining the body

- IP VERBX LEDATA 'CEEDUMP'

Traceback:

DSA	Entry	E	Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEHSDMP		+000000DA		CEEPLPKA	CEEHSDMP	HLE7770	Call
2	CEEHDSP		+00003EBA		CEEPLPKA	CEEHDSP	HLE7770	Call
3	CPROG3		+0000008C		CPROG3			Exception
4	@@FECEB							
			+00312B7A					Call
5	@@DB2C		+00000102		CEEEV003			Call
6	IGZCFCC		+000002C0		IGZCPAC	IGZCFCC		Call
7	COBOLED2		+000004C0		COBOLSHR	COBOLED2	.....%.	Call
8	COBOLED1		+0000037A		COBOLSHR	COBOLED1	.....%.	Call

DSA	DSA Addr	E	Addr	PU Addr	PU Offset	Comp Date	Compile Attributes
1	219A0860		217D8038	217D8038	+000000DA	20100319	CEL
2	2199D768		217C8180	217C8180	+00003EBA	20100319	CEL
3	2199D6D8		219C6070	219C6070	+0000008C	20070212	C/C++
4	2199D5E8		219BDE40	219BDE40	+00312B7A	*****	LIBRARY
5	2199D560		21C504F0	21C504F0	+00000102	*****	LIBRARY
6	2199D380		21922548	21922548	+000002C0	20100316	LIBRARY
7	2199D1D8		21702220	21702220	+000004C0	20100720	COBOL
8	2199D030		21700000	21700000	+0000037A	20100720	COBOL



# Examining the body

- IP VERBX LEDATA 'CM'

```

CIBH: 2170F410
+000000 EYE:CIBH BACK:00000000 FRWD:2199E570
+000010 PTR_CIB:2199E060 FLAG1:C5 ERROR_LOCATION_FLAGS:
+000018 HDLQ:00000000 STATE:00000000 PRM_DESC:00000000
+000024 PRM_PREFIX:00000000
+000028 PRM_LIST:2199E078 2199E140 2199E14C 2170FA5C
+000038 PARM_DESC:00000000 PARM_PREFIX:00000000
+000040 PARM_LIST:2199E13C 2199E060 2199E14C 2170FA5C FUN:00000067
+000054 CIB_SIZ:010C CIB_VER:0004 FLG_5:48 FLG_6:23
+00005A FLG_7:04 FLG_8:00 FLG_1:00 FLG_2:00 FLG_3:00
+00005F FLG_4:05 ABCD:940C9000 ABRC:00000009
+000068 OLD_COND_64:00030C89 59C3C5C5 (CEE3209S)
+000070 OLD_MIB:00000001 COND_64:00030C89 59C3C5C5 (CEE3209S)
+00007C MIB:00000001 PL:219C6018 SV2:2199D6D8
+000088 SV1:2199D6D8 INT:219C60FC MID:00000003
+000094 HDL_SF:21714600 HDL_EPT:A1918000 HDL_RST:00000000
+0000A0 RSM_SF:2199D6D8 RSM_POINT:219C60FE RSM_MACHINE:2170F858
+0000B0 COND_DEFAULT:00000003 Q_DATA_TOKEN:2170F548 FDBK:00000000
+0000BC ABNAME:..... BBRANCH_OFFSET:00000000
+000220 BBRANCH_STMTID:..... BBRANCH_STMTLEN:0000
  
```



# Examining the body

- IP VERBX LEDATA 'CM'
  - PSW and Regs at time of condition

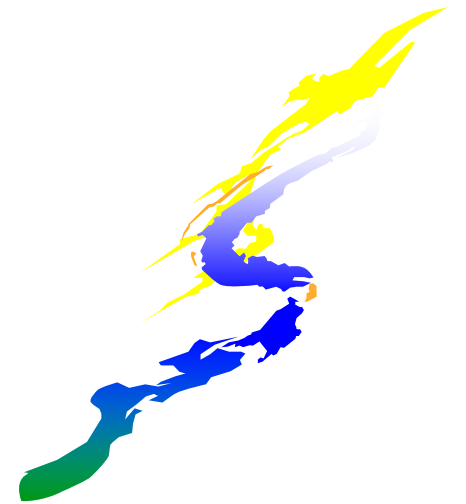
Machine State

```

+000248  MCH EYE:ZMCH
+000250  GPR00:00000000      GPR01:00000000A
+000258  GPR02:A1CD09BC      GPR03:219C60B8
+000260  GPR04:2199D2D8      GPR05:21F91A00
+000268  GPR06:21F92AC8      GPR07:219BDE40
+000270  GPR08:A19C63A8      GPR09:21F93368
+000278  GPR10:A19C6070      GPR11:A19C60A0
+000280  GPR12:21713B58      GPR13:2199D6D8
+000288  GPR14:00000000      GPR15:00000006
+000290  PSW:078D2400 A19C60FE
+000298  ILC:0002      IC1:00      IC2:09      PFT:00000000
  
```



# Back in the lab



# Who Took the Wild Branch...

## The Clue: The Breaking Event Address

- Available on z/Architecture machines with PER-3 facility
- 64-bit CPU register that is updated with the address of any instruction that causes a break in sequential instruction execution
- BEA register contents saved when interrupt occurs
- BEA value saved in the SDWA and EPIE
- LE uses this value to identify possible wild branch location
  - LE may also attempt to use linkage register (i.e., R14)



# Who Took the Wild Branch...

## CEEDUMP Output

```

:
Traceback:
  DSA   Entry      E Offset  Statement  Load Mod  Program Unit  Service  Status
  1     CEEHDSP     +000040EC CEEPLPKA   CEEHDSP    UK28165      Call
  2     OFFCLIFF    -20904128 CSICBBEA   OFFCLIFF    Exception
  3     LEAP        +0000031E 21         CSICBBEA   LEAP        Call
  4     WILD        +0000031E 9          CSICBBEA   WILD        Call

```



```

:
Condition Information for Active Routines
Condition Information for OFFCLIFF (DSA address 20998370)
  CIB Address: 20998E30
  Current Condition:
    CEE0198S The termination of a thread was signaled due to an unhandled condition.
  Original Condition:
    CEE3201S The system detected an operation exception (System Completion Code=0C1).
  Location:
    Program Unit: OFFCLIFF Entry: OFFCLIFF Statement: Offset: -20904128
    Possible Bad Branch: Statement: 35 Offset: +00000342

```

```

Machine State:
  ILC..... 0002      Interruption Code..... 0001
  PSW..... 078D0000 80000002
  GPR0..... 209B8214  GPR1..... 00000000  GPR2..... 209B8190  GPR3..... 20904400
  GPR4..... 20904160  GPR5..... 20994100  GPR6..... 00000000  GPR7..... 00FD8640
  GPR8..... 209B8210  GPR9..... 20994C78  GPR10.... 20904250  GPR11.... 20904324
  GPR12.... 209139C0  GPR13.... 20998370  GPR14.... A090446C  GPR15.... 00000000

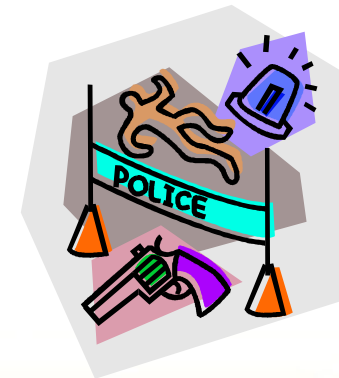
```

# Who Took the Wild Branch...

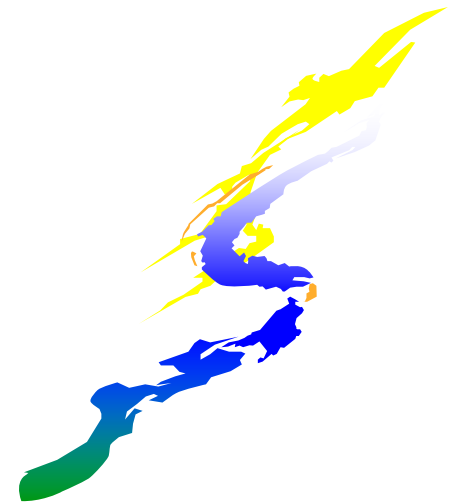
## LEDATA Output ('CM' or 'ALL')

```

CIBH: 209102A8
+000000 EYE:CIBH BACK:00000000 FRWD:20999340
:
+000070 OLD_MIB:00000000 COND_64:00030C81 59C3C5C5 (CEE3201S)
:
+0000BC ABNAME:..... BBRANCH_OFFSET:00000342
+000220 BBRANCH_STMTID:35 BBRANCH_STMTLEN:0002
Machine State
+000248 MCH_EYE:ZMCH
+000250 GPR00:209B8214 GPR01:00000000
+000258 GPR02:209B8190 GPR03:20904400
+000260 GPR04:20904160 GPR05:20994100
+000268 GPR06:00000000 GPR07:00FD8640
+000270 GPR08:209B8210 GPR09:20994C78
+000278 GPR10:20904250 GPR11:20904324
+000280 GPR12:209139C0 GPR13:20998370
+000288 GPR14:A090446C GPR15:00000000
+000290 PSW:078D0000 80000002
:
+0002EC INT_SF:00000000 FLAGS:40 EXT:00000000 BEA:2090446A
  
```



# Forcing LE to Spill the Beans



# Coercing Evidence from C-RTL

`perror()`: Prints string with errno message to stderr

`strerror()`: Returns message string for a given errno value

Environment variables to tell these functions to provide more information

- `_EDC_ERRNO_DIAG`
- `_EDC_ADD_ERRNO2`

# Coercing Evidence from C-RTL...

**`_EDC_ERRNO_DIAG = "x,y"`**

## **x**

- 0: Do not generate additional diagnostic information (default)
- 1: Call `ctrace()` – writes a traceback to CEEDUMP
- 2: Call `csnap()` – writes a condensed dump to CEEDUMP
- 3: Call `cdump()` – writes a CEEDUMP; sends snap dump to CEESNAP

## **y**

- List of errnos for which action “x” is to be taken, separated by commas
- If not specified, defaults to all errno values

# Coercing Evidence from C-RTL...

## **\_EDC\_ADD\_ERRNO2**

- Tells perror() and strerror() whether to append errno2 information to their output
- For strerror(121):
  - `_EDC_ADD_ERRNO2 = "0"` or unset:  
EDC5121I Invalid argument
  - With `_EDC_ADD_ERRNO2 = "1"`:  
EDC5121I Invalid argument. (errno2=0x0C0F8402)
- Default perror() behavior for z/OS V1.9 is to include errno2 value

### Interpret errno2 value using:

- BPXMTEXT TSO command (handles both USS and LE errno2 values)
- z/OS UNIX System Services Messages and Codes
- <http://www.ibm.com/servers/eserver/zseries/zos/le/assist/r9errno2.html>

# Sources of Additional Information

- z/OS Language Environment Debugging Guide
- z/OS Language Environment Programming Reference
- z/OS MVS IPCS Commands
- z/OS UNIX System Services Messages and Codes
- Web site
  - <http://www.ibm.com/servers/eserver/zseries/zos/le/>

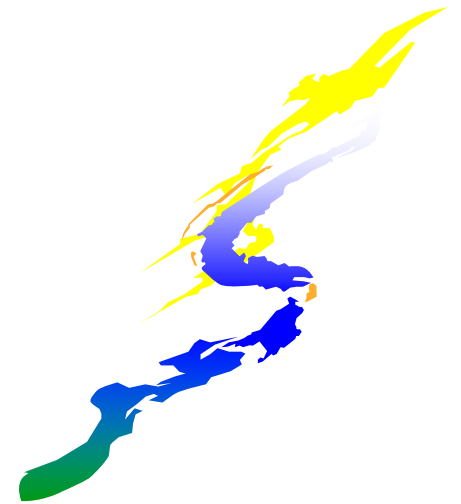


# Appendix





# Programs



# Program COBOLED1

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. COBOLED1.
```

```
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
```

```
01 WS-VARS.
```

```
    05 WS-COMP1 PIC S9(4) COMP-4 VALUE 1234.
```

```
PROCEDURE DIVISION.
```

```
MAIN-PROG.
```

```
    CALL "COBOLED2".
```

```
    STOP RUN.
```

```
END PROGRAM COBOLED1.
```

# Program COBOLED2

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. COBOLED2.
```

```
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
```

```
01 WS-VARS.
```

```
    05 WS-COMP1 PIC S9(4) COMP-4 VALUE ZEROES.
```

```
    05 WS-COMP2 PIC S9(4) COMP-4 VALUE ZEROES.
```

```
    05 WS-COMP3 PIC S9(4) COMP-4 VALUE ZEROES.
```

```
01 DYN-NAME PIC X(8).
```

```
PROCEDURE DIVISION.
```

```
MAIN-PROG.
```

```
    MOVE 32 TO WS-COMP3.
```

```
    MOVE 10 TO WS-COMP1.
```

```
    MOVE "CPROG3" TO DYN-NAME.
```

```
    CALL DYN-NAME.
```

```
    STOP RUN.
```

```
END PROGRAM COBOLED2.
```

# Program CPROG3

```
#pragma linkage(CPROG3, COBOL)
#include <stdio.h>
void CPROG3()
{
    int i, j, k;
    /* Compiler needs to be coerced to */
    /* divide by zero.                  */
    j = 5;
    k = 10;
    i = 6 / (k - j*2);
    return;
}
```

# Program WILD

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. WILD.
```

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
LINKAGE SECTION.  
PROCEDURE DIVISION.  
MAIN-PROG.  
    CALL "LEAP".  
    GOBACK.  
END PROGRAM "WILD".
```

# Program LEAP

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. LEAP.
```

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
LINKAGE SECTION.  
PROCEDURE DIVISION.  
MAIN-PROG.  
    CALL "OFFCLIFF".  
    GOBACK.  
END PROGRAM "LEAP".
```

# Program OFFCLIFF

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. OFFCLIFF.
```

```
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
```

```
01 FP USAGE IS PROCEDURE-POINTER.  
01 OBVIOUS PIC X(4) VALUE "OBVS".
```

```
LINKAGE SECTION.
```

```
PROCEDURE DIVISION.
```

```
MAIN-PROG.
```

```
    SET FP TO NULL.
```

```
    CALL FP.
```

```
    DISPLAY "SURVIVED THE JUMP!"
```

```
    GOBACK.
```

```
END PROGRAM "OFFCLIFF".
```

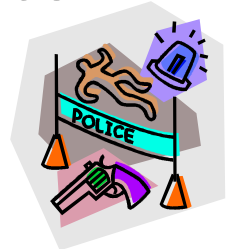
# Really cool lab equipment





# Reconstructing a Vandalized Stack

The Crime: Stack control data is damaged, unable to get a complete Traceback



```

:
Registers and PSW:
GPR0..... 84000000  GPR1..... 84000FF3  GPR2..... 00010078  GPR3..... 20EE0DF0
GPR4..... 20F21018  GPR5..... 20EE2530  GPR6..... 20EE149C  GPR7..... 20F016B0
GPR8..... 20EE22E8  GPR9..... 00000004  GPR10.... 20ED6378  GPR11.... 2181018C
GPR12.... 20EE5A20  GPR13.... 20EE13F0  GPR14.... A180FFA8  GPR15.... 00000004
PSW..... 078D1000 A180FFBA
  
```

Traceback:

DSA	Entry	E	Offset	Statement	Load Mod	Program Unit	Service	Status
-----	-------	---	--------	-----------	----------	--------------	---------	--------

**WARNING** An invalid DSA pointer was found on traceback while processing  
 DSA: 20EE13F0 and CAA: 20EE5A20

:

# Reconstructing a Vandalized Stack...

The Clues:

- **CEECAADDSA, “Dummy DSA”** – First Stack Frame on the stack
- **Backchain Pointer** – Location in stack frame that points to previous stack frame, offset X'04'
- **Next Available Byte, “NAB”** – Location in stack frame where next stack frame is to be built, offset X'4C'

The Tool: IPCS **RUNCHAIN** command

- Processes a chain of control blocks
- Input: Starting address, “link” offset



# Reconstructing a Vandalized Stack...

## LEDATA Output

```
:  
CEECAA: 20EE5A20  
+000000 FLAG0:00 LANGP:08 BOS:20F01018 EOS:20F21018  
+000044 TORC:00000000 TOVF:80071660 ATTN:20EE0E38  
:  
+0002CC DMC:00000000 ABCODE:00000000 RSNCODE:00000000  
+0002D8 ERR:20EE2DF0 GETSX:80070B20 DDSA:20EE63C0  
+0002E4 SECTSIZ:00000000 PARTSUM:00000000  
+0002EC SSEXPNT:00000000 EDB:20EE4698 PCB:20EE41E8  
:
```



# Reconstructing a Vandalized Stack...

```
ip runcchain address(20ee63c0) display length(100) link(76) name(dsa)
```

```

:
DSA005
LIST 20F01510 ASID(X'01CE') LENGTH(X'64') AREA
ASID(X'01CE') ADDRESS(20F01510.) KEY(88)
20F01510. 00104001 20F01370 00000000 A0ED44E0
20F01520. 20ED6250 20EFDF90 00000000 20F21230 20ED4478 20ED41F8 20EFD100 00000000
20F01540. 00FD8640 00058A80 20EFDE60 20ED42E8 20ED43A0 20EE5A20 00000004 20F016B0
20F01560. 20F01410 20ED1F18 20F01510 20EFDE60 20EFD100

```

```

DSA006
LIST 20F016B0 ASID(X'01CE') LENGTH(X'64') AREA
ASID(X'01CE') ADDRESS(20F016B0.) KEY(88)
20F016B0. 00104001 20F01510 20F018D0 A0ED8730
20F016C0. 80072020 A0ED85C0 20EFE048 00000000 20ED4478 20ED6288 20EFD100 00000000
20F016E0. 00FD8640 00058A80 000581BC 20ED42E8 20ED83C0 20EE5A20 00000004 20F01850
20F01700. 20F015B0 20ED3FA8 20F016B0 20EFE048 20EFD100

```

```

DSA007
LIST 20F01850 ASID(X'01CE') LENGTH(X'64') AREA
ASID(X'01CE') ADDRESS(20F01850.) KEY(88)
20F01850. 00002001 20F016D0 20F01A70 A18F2934
20F01860. A18E2588 00000000 00000000 00000095 00102001 20F01610 20F019F0 A198CE12
20F01880. 20ED61C8 00000001 20F21230 20EFDFD0 20ED6064 00000010 00000004 20ED41F8
20F018A0. 20F01750 20ED6038 20EFDE60 20ED62D4 20EFD100

```



# Reconstructing a Vandalized Stack...

## LEDATA 'DSA(20F016B0) CEEDUMP' Output

### Traceback:

DSA	Entry	E	Offset	Statement	Load Mod	Program Unit	Service	Status
1	LOOK		+000024DE		CSICBLDP	LOOK	.....%	Call
2	FORTHE		+0000031E		CSICBLDP	FORTHE	.....%	Call
3	WEAPON		+0000031E		CSICBLDP	WEAPON	.....%	Call
4			+20ED03BE					Call

**WARNING** An invalid DSA pointer was found on traceback while processing  
**DSA: BADBADBD and CAA: 20EE5A20**

DSA	DSA Addr	E	Addr	PU Addr	PU Offset	Comp Date	Compile
1	20F016B0		20ED6250	20ED6250	+000024DE	20080128	COBOL
2	20F01510		20ED41C0	20ED41C0	+0000031E	20080128	COBOL
3	20F01370		20ED2130	20ED2130	+0000031E	20080128	COBOL
4	20F011D0		00000000	00000000	+20ED03BE	*****	

# Reconstructing a Vandalized Stack...

```
ip runcchain address(20ee63c0) display length(100) link(76) name(dsa)
```

DSA001

```
LIST 20EE63C0. ASID(X'01CE') LENGTH(X'64') AREA
```

```
ASID(X'01CE') ADDRESS(20EE63C0.) KEY(88)
```

```
20EE63C0. 00000000 0000D008 00000000 0006AB30 20ECE000 00000000 20EE47B8 00000000
```

```
20EE63E0 LENGTH(X'20')==>All bytes contain X'00'
```

```
20EE6400. 00000000 20EE5A20 00000000 20F01030 20F01030 00000000 00000000 00000000
```

```
20EE6420. 00000000
```

DSA002

```
LIST 20F01030. ASID(X'01CE') LENGTH(X'64') AREA
```

```
ASID(X'01CE') ADDRESS(20F01030.) KEY(88)
```

```
20F01030. 00000000 00000000 00000000 00000000 00104001 20EE63C0 20F014A8 A0ECE324
```

```
20F01040. 20ED00A0 20EFD578 00000000 20F21050 20ECE2BC 20ECE038 20EE47B8 00000000
```

```
20F01060. 00000000 00058A80 20EFD448 20ECE128 20ECE1E4 20EE5A20 00000000 20F011D0
```

```
20F01080. 00000000 00000000 20F01030 20EFD448 00000000
```

DSA003

```
LIST 20F011D0. ASID(X'01CE') LENGTH(X'64') AREA
```

```
ASID(X'01CE') ADDRESS(20F011D0.) KEY(88)
```

```
20F011D0. 00000000 00000000 00000000 00000000 00104001 BADBADBD 00000000 A0ED03C0
```

```
20F011E0. 20ED2130 20EFDBC0 00000000 20F210F0 20ED0358 20ED00D8 20EFD100 00000000
```

```
20F01200. 00FD8640 00058A80 20EFDA90 20ED01C8 20ED0280 20EE5A20 00000000 20F01370
```

```
20F01220. 00000000 00000000 20F011D0 20EFDA90 20EE47B8
```

```
:
```





# Reconstructing a Vandalized Stack...

## LEDATA 'DSA(20F01030) CEEDUMP' Output

Traceback:

DSA	Entry	E	Offset	Statement	Load Mod	Program Unit	Service	Status
1	DUMPSTER		+00000322		CSICBLDP	DUMPSTER	.....%.	Call

DSA	DSA Addr	E	Addr	PU Addr	PU Offset	Comp Date	Compile
1	20F01030		20ECE000	20ECE000	+00000322	20080128	COBOL

# Reconstructing a Vandalized Stack...

What about the corrupted stack frame?

```
ip runchain address(20ee63c0) display length(100) link(76) name(dsa)
:
```

DSA002

```
LIST 20F01030. ASID(X'01CE') LENGTH(X'64') AREA
ASID(X'01CE') ADDRESS(20F01030.) KEY(88)
```

20F01030.					00104001	20EE63C0	20F014A8	A0ECE324
20F01040.	20ED00A0	20EFD578	00000000	20F21050	20ECE2BC	20ECE038	20EE47B8	00000000
20F01060.	00000000	00058A80	20EFD448	20ECE128	20ECE1E4	20EE5A20	00000000	20F011D0
20F01080.	00000000	00000000	20F01030	20EFD448	00000000			

DSA003

```
LIST 20F011D0. ASID(X'01CE') LENGTH(X'64') AREA
ASID(X'01CE') ADDRESS(20F011D0.) KEY(88)
```

20F011D0.					00104001	BADBADBD	00000000	A0ED03C0
20F011E0.	20ED2130	20EFD578	00000000	20F210F0	20ED0358	20ED00D8	20EFD100	00000000
20F01200.	00FD8640	00058A80	20EFDA90	20ED01C8	20ED0280	20EE5A20	00000000	20F01370
20F01220.	00000000	00000000	20F011D0	20EFDA90	20EE47B8			

R15 slot of the previous stack frame provides yet another clue!

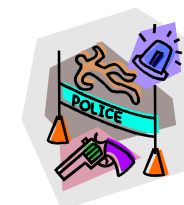




# Reconstructing a Vandalized Stack...



## IPCS Browse Panel



```

20ED00A0  47F0F028  00C3C5C5  000001A0  00000014  | .00..CEE..... |
20ED00B0  47F0F001  98CEAC00  20ED0156  00000000  | .00.q..... |
20ED00C0  00000000  00000000  90ECD00C  4110F038  | .....}...0. |
20ED00D0  98EFF04C  07FF0000  20ED00A0  00000000  | q.0<..... |
20ED00E0  20ED1F18  20ED014E  20ED00A0  20ED02CE  | .....+..... |
20ED00F0  20ED83C0  20ED016A  00104001  00000008  | ..c{..... |
20ED0100  C9D5E3C8  C5404040  F2F0F0F8  F0F1F2F8  | INTHE    20080128 |
20ED0110  F1F7F4F0  F3F5F0F3  F0F3F0F0  04740000  | 174035030300... |
20ED0120  0000076C  A0C87CCC  20000000  10000000  | ...%.H@..... |
20ED0130  00000000  08000000  00800000  00000000  | ..... |
20ED0140  00000002  0000F000  40404040  0005C9D5  | .....0.    ..IN |
20ED0150  E3C8C540  40400500  000120ED  83100000  | THE     .....c... |
20ED0160  0000FFFF  FFB220ED  00A020ED  00D80000  | .....Q.. |
    
```

# Reconstructing a Vandalized Stack...

## Complications

- More substantial damage to the stack can make this task much more difficult
- NAB is only useful for current stack segment
  - Use “LEDATA ‘STACK’” to identify additional stack segments
  - Use RUNCHAIN command on first stack frame in the segment (Starts at offset X'18' into stack segment)  
OR
  - Increase initial stack segment size with STACK run-time option so that NAB is always in current stack segment

# Interpreting Raw Data

Manually mapping storage is tedious:



20F016B0	00104001	20F01510	20F018D0	A0ED8730	.. ..0...0.}..g.
20F016C0	80072020	A0ED85C0	20EFE048	00000000	.....e{..\.....
20F016D0	20ED4478	20ED6288	20EFD100	00000000	.....h..J.....
20F016E0	00FD8640	00058A80	000581BC	20ED42E8	..f .....a....Y
20F016F0	20ED83C0	20EE5A20	00000004	20F01850	..c{..!.....0.&
20F01700	20F015B0	20ED3FA8	20F016B0	20EFE048	.0.....y.0....\..

+04 = backchain

+10 = Called routine

+4C = NAB

. . .

There has to be a better way!

# Interpreting Raw Data...

## IPCS CBFORMAT Command

- Formats a control block
- Syntax: CBF(ORMAT) <cbaddr> STR(<cbname>)
  - <cbaddr> can be address or symbol
  - STR(UCTURE) support provided by various components
    - IPCS Commands, Appendix D for MVS control blocks
    - Language Environment Debugging Guide for LE control blocks
      - CEExxx for AMODE 24/31 (CEECAA, CEECIBH, CEEDSA, CEEEDB, CEEHANC, CEESTKH, etc.)
      - CELxxxx for AMODE 64 (CELCIBH, CELDSA, CELEDB, CELLAA, CELLCA, CELSANC, etc.)

# Interpreting Raw Data...

CBF 20F016B0 str(ceedsa)

or

CBF DSA006 str(ceedsa)



```

DSA: 20F016B0
+000000  FLAGS:0010  MEMD:4001  BKC:20F01510  FWC:20F018D0
+00000C  R14:A0ED8730  R15:80072020  R0:A0ED85C0
+000018  R1:20EFE048  R2:00000000  R3:20ED4478
+000024  R4:20ED6288  R5:20EFD100  R6:00000000
+000030  R7:00FD8640  R8:00058A80  R9:000581BC
+00003C  R10:20ED42E8  R11:20ED83C0  R12:20EE5A20
+000048  LWS:00000004  NAB:20F01850  PNAB:20F015B0
+000064  RENT:20ED41C0  CILC:20EFDE08  MODE:00058108
+000078  RMR:20ED8EB8
  
```

# Interpreting Raw Data...

CBF can be combined with RUNCHAIN command:

ip runchain address(20f01030) link(76) name(dsa) exec((cbf x str(ceedsa)))

DSA001

LIST 20F01030. ASID(X'01CE') LENGTH(X'04') AREA  
ASID(X'01CE') ADDRESS(20F01030.) KEY(88)

```

DSA: 20F01030
+000000  FLAGS:0010  MEMD:4001  BKC:20EE63C0  FWC:20F014A8
+00000C  R14:A0ECE324  R15:20ED00A0  R0:20EFD578
+000018  R1:00000000  R2:20F21050  R3:20ECE2BC
+000024  R4:20ECE038  R5:20EE47B8  R6:00000000
+000030  R7:00000000  R8:00058A80  R9:20EFD448
+00003C  R10:20ECE128  R11:20ECE1E4  R12:20EE5A20
+000048  LWS:00000000  NAB:20F011D0  PNAB:00000000
+000064  RENT:00000000  CILC:00000000  MODE:00000000
+000078  RMR:00000000
  
```



DSA002

LIST 20F011D0. ASID(X'01CE') LENGTH(X'04') AREA  
ASID(X'01CE') ADDRESS(20F011D0.) KEY(88)

```

DSA: 20F011D0
+000000  FLAGS:0010  MEMD:4001  BKC:BADBADBD  FWC:00000000
+00000C  R14:A0ED03C0  R15:20ED2130  R0:20EFDDBC0
+000018  R1:00000000  R2:20F210F0  R3:20ED0358
+000024  R4:20ED00D8  R5:20EFD100  R6:00000000
+000030  R7:00FD8640  R8:00058A80  R9:20EFDA90
+00003C  R10:20ED01C8  R11:20ED0280  R12:20EE5A20
+000048  LWS:00000000  NAB:20F01370  PNAB:00000000
+000064  RENT:20ECE000  CILC:20EFD3F0  MODE:00058108
+000078  RMR:20ED8EB8
  
```