Securing RXSOCKET applications with TLS

Presenters:

Perry Ruiter

Dave Jones

Abstract

VM 6.4 included support for securing IUCV based sockets with TLS. Sadly 6.4 did not enhance Rexx Sockets to exploit that support. Now that 7.1 has shipped (still) without TLS support in Rexx Sockets, customers are forced to take matters into their own hands. Attend this session for an overview of z/VM's SSL/TLS support, what was new in 6.4, the changes done to add TLS support to Rexx Sockets and finally, we will review a popular Rexx Sockets application that has been secured with TLS

Agenda

- Introduction
- SSL Configuration in z/VM
- Create in Internal z/VM Certificate Database
- Update z/VM TCP/IP Configuration
- RXSOCKET Updates
- Examples

Trade Marks

Introduction

This document provides practical information for the configuration of secured (encrypted) communications with a z/VM ® 7.1 system, based on the Secure Socket Layer/Transport Layer Security (SSL/TLS) technology. Once z/VM SSL/TLS application servers are configured and started with TCP/IP, z/VM TCP/IP applications servers can participate in SSL/TLS connections.

SSL == old protocol; TLS = =new protocol

In addition, z/VM TCP/IP supports Dynamic Secured Socket Layer/Transport Layer Security (Dynamic SSL/TLS) connections. In such connections, application servers themselves control the level of acceptance of SSL and the digital certificate to be used.

This presentation focuses on the configuration of z/VM RSCLIENT/RSSERVER and IPGATE application server for SSL/TLS connections, and provides client secure configuration examples.

It is assumed that the reader has a good understanding of z/VM TCP/IP server configuration, SSL/TLS concepts and digital certificates.

For a complete information on the SSL implementation in z/VM, refer to z/VM documentation: *TCP/IP Planning and Customization* , SC24-6238-xx

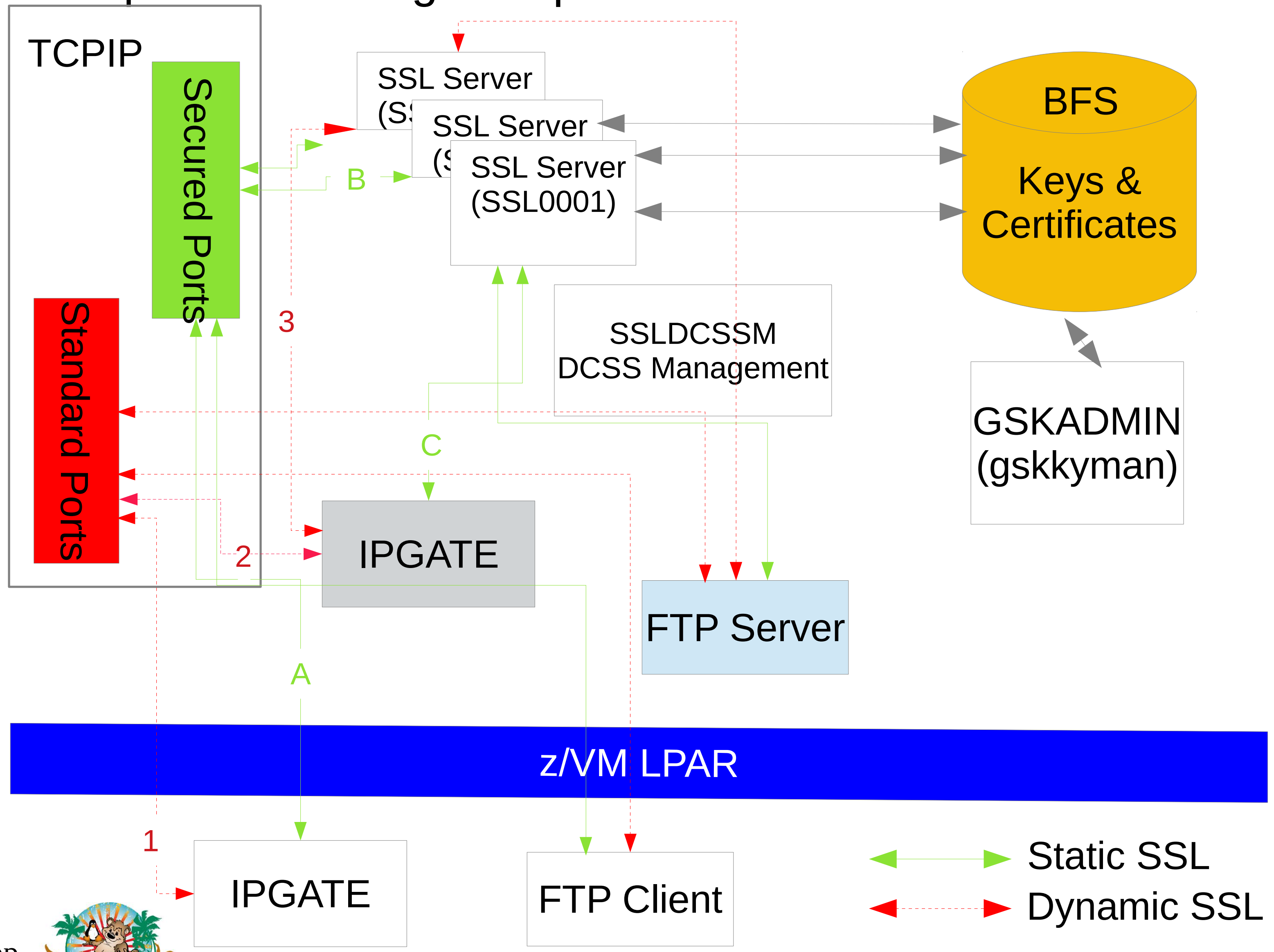
SSL CONFIGURATION IN z/VM

Topics

- z/VM SSL implementation global picture
- SSL connection principles
- SSL session general processing steps
- Static SSL connection
- Dynamic SSL connection
- SSL server environment in z/VM
- Concept of « pool »
- Hardware cryptographic support

SSL CONFIGURATION IN z/VM

z/VM SSL implementation global picture



SSL CONFIGURATION IN z/VM

Secured vs. Standard ports

Example:

```
PORT
    :
    80 TCP HTTPSD      ; Web server
    :
    81 TCP HTTPSD2 SECURE <label>  ; Secure server
    :
```

Standard port

Secured port

SSL CONFIGURATION IN z/VM

SSL connection principles

A SSL session consists in the following steps (phases):

1) CONNECT
2) HANDSHAKE
3) DATA TRANSMISSION
4) CLOSE

These steps are described below.

SSL CONFIGURATION IN z/VM

SSL session general processing steps

CONNECT step:

In this initial phase, a remote client is requesting a connection with an application server (IPGATE, FTP...). An SSL server is designated to handle the secure connection. Two separate connections are established in the SSL session, depending on whether a static or dynamic SSL connection is requested. The differences are explained in the next section

SSL CONFIGURATION IN z/VM

SSL session general processing steps

HANDSHAKE step:

The client initiates a handshake protocol to produce the cryptographic parameters for the session. The SSL server (on behalf of the application server) presents the server certificate to the client. If a certificate validation is required by the client, the certificate signature is validated using the issuer Certificate Authority (CA) certificate, which must be available to the client. After validation, the server and the client:

- Agree on cryptographic parameters (protocol, algorithms)
- Generate shared secrets
- Generate symmetric key from the shared secrets, used to encrypt/decrypt the data in the connection

SSL CONFIGURATION IN z/VM

SSL session general processing steps

DATA TRANSMISSION step:

- Encrypted data is produced on the client and transmitted to the server over the network
- Inbound encrypted data received from the remote client is first decrypted by the SSL servers, then forwarded in clear to the application server (IPGATE, FTP)
- Outbound unencryted data received from the application server is encrypted by the SSL server, transmitted to the remote client over the network and decrypted locally.

SSL CONFIGURATION IN z/VM

SSL session general processing steps

CLOSE step:

When a close request is received from either the client or the application server, the SSL server sends a close request to the other party and cleans up the connection.

SSL CONFIGURATION IN z/VM

Static SSL connection

- The secure "SSL attribute" is granted as soon as the session is initially established (connect phase)
- z/VM TCP/IP application servers (IPGATE, FTP...) are "SSL unaware" which means that SSL encryption/decryption is completely handled by the TCP/IP and SSL servers.
- the application server configuration remains unchanged, but secure listening ports are defined in the TCP/IP server configuration and specified in the client configuration as well
- In the figure above the green solid line marked with 'A', 'B', and 'C' represents a static SSL connect phase for the IPGATE server

SSL CONFIGURATION IN z/VM

Dynamic SSL connection

- Both the server and the client are able to control the acceptance and the establishment of the secure “SSL attribute” for the session
- The z/VM application server is “SSL aware” and will itself handle the communication with the SSL server by mean of a set of specialized APIs and the use of appropriate digital certificate accessible by the SSL server,
- Secure ports are no longer required with dynamic SSL/TLS, as the application servers will continue to listen on their standard ports.
- In the figure above the red dashed solid line marked with ‘1’, ‘2’, and ‘3’ represents a dynamic SSL connect phase for the IPGATE server

SSL CONFIGURATION IN z/VM

SSL server environment in z/VM

A z/VM SSL/TLS server environment consists of the following components:

- One TCP/P VM server configured to enable SSL/TLS connections
- One (or more) pools of SSL/TLS servers associated with that TCP/IP server that implement the actual SSL TLS encryption/decryption algorithms.
- One DCSS Management Agent virtual machine maintaining SSL/TLS server cache information in a z/VM shared segment, for the SSL/TLS server(s) associated to the TCP/IP server

Multiple SSL server environments can be defined in the same z/VM, running independently from each other

SSL CONFIGURATION IN z/VM

SSL server environment in z/VM

At z/VM 7.1 installation, a default SSL/TLS server environment is created with the following components

- TCP/IP server **TCPIP**
- SSL servers **SSL0000** n (n =1 to 5)
- DCSS agent **SSLDCSSM**

The SSL environments rely on certificates defined in Certificate and key databases. The databases and certificates management tasks (create, deletion, certificates exports and imports) are performed from the **GSKADMIN** virtual machine, by mean of a utility program called *gskkyman* .

A single database can be used by all SSL server environments.

A single certificate in a database can be used by all the SSL server environments sharing that database.

SSL CONFIGURATION IN z/VM

Concept of « pool »

z/VM has had for a long time the concept of a “pool” of virtual machines, all configured to work on the same type of workload, say, performing SSL/TS encryption.

A pool is defined in the USER DIRECT file via either a USER or IDENTITY statement followed by the “POOL” statement. An example:

```
IDENTITY SSL LBYONLY 160M 256M G
POOL LOW 1 HIGH 5 PROFILE TCPSSLU
```

Creates a set of 5 virtual machines (SSL00001...SSL00005), all having common characteristics (class G, 160M memory, surrogate logon only, and based on the TCPSSLU profile).

The default SSL server pool (5 servers shown above) is designed to serve a maximum of 3000 connections, with a maximum of 600 sessions per server.

SSL CONFIGURATION IN z/VM

Hardware cryptographic support

z/VM SSL is supporting both forms of cryptographic hardware:

CPACF CP-Assisted Cryptographic Facility.

- This is a no charge feature built in the IBM Z ® or Linux One™ cores, designed to accelerate the use of symmetric algorithms (AES, DES) or hash functions (SHA-1, SHA-256). No configuration is required as the SSL/TLS server makes use of this feature automatically.

Crypto Express card.

- Used to accelerate asymmetric algorithms such as clear-key RSA. When available to the z/VM LPAR, a crypto express card can be used by the SSL/TLS server, providing that a CYPTO APVIRTUAL statement is coded in the SSL server z/VM profile (e.g. TCPSSLU).

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Topics

- **GSKADMIN** and *gskkyman*
- Create the database
- Grant read access
- Create the Self-signed CA certificate
- Create the CA-signed server certificate
- Display certificate information

CREATE INTERNAL z/VM CERTIFICATE DATABASE

GSKADMIN and *gskkyman*

To create and manage the database, the z/VM user id GSKADMIN is available.

The utility program *gskkyman* is used to perform management tasks against the certificate database.

The GSKADMIN user owns both the BFS file space where the key database resides and the BFS file space used as SSL server temporary work space.

GSKADMIN also serves as the SSL server administrative user ID, as well.

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the database

The following information is required to create the database:

- database name – use “Database.kdb”
- database password – user defined
- password expiration – 365 days (one year)
- database record length – use default value 5000
- Comply to FIPS 6 standard – enter 1

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the database

```
    gskkyman

           Database Menu

       1 - Create new database
       2 - Open database
       3 - Change database password
       4 - Change database record length
       5 - Delete database
       6 - Create key parameter file
       7 - Display certificate file (Binary or Base64 ASN.1 DER)

      0 - Exit program
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the database

```
Enter option number:
1

Enter key database name (press ENTER to return to menu):
Database.kdb
Enter database password (press ENTER to return to menu):

Re-enter database password:

Enter password expiration in days (press ENTER for no expiration):
365

Enter database record length (press ENTER to use 5000):

Enter 1 for FIPS mode database or 0 to continue:
1

Key database /etc/gskadm/Database.kdb created.
```

The database has now been created.

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the database

Once the database has been created, the database password must be stored to allow the SSL server to work with the database with automatic login. On the main menu, select option 10:

```
        Expiration: 2020/06/18 10:30:29
        Type: FIPS

    1 - Manage keys and certificates
    2 - Manage certificates
    3 - Manage certificate requests
    4 - Create new certificate request
    5 - Receive requested certificate or a renewal certificate
    6 - Create a self-signed certificate
    7 - Import a certificate
    8 - Import a certificate and a private key
    9 - Show the default key
   10 - Store database password
   11 - Show database record length

    0 - Exit program

 Enter option number (press ENTER to return to previous menu):
 10

 Database password stored in /etc/gskadm/Database.sth.

 Press ENTER to continue.
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Grant read access

First, Select option **0** to exit from the **GSKKYMAN** program.

The POSIX statement in the TCPSSLU profile used to generate the default SSL pool sets the SSL server group ownership to security.

At this point, only the GSKADMIN user has access to the files in r/w mode. We want users from the same group (security) be able to access the files in read mode. The SSL servers are part of of the security group.

Execute the following **openvm** commands to grant the read authority for the security group to the kdb and sth files:

```
Ready;
openvm permit /etc/gskadm/Database.kdb rw- r-- ---
Ready;
openvm permit /etc/gskadm/Database.sth rw- r-- ---
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the Self-signed CA certificate

(note: This is just an example for the sake of showing how it's done. In most cases, you will be using a certificate created by an external CA.)

Logged on as the GSKADMIN user id, start the gskkyman program:

```
gskkyman

       Database Menu

   1 - Create new database
   2 - Open database
   3 - Change database password
   4 - Change database record length
   5 - Delete database
   6 - Create key parameter file
   7 - Display certificate file (Binary or Base64 ASN.1 DER)

  0 - Exit program

Enter option number:
2

Enter key database name (press ENTER to return to menu):
Database.kdb
Enter database password (press ENTER to return to menu):
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the Self-signed CA certificate (cont)

```
    1 - Manage keys and certificates
    2 - Manage certificates
    3 - Manage certificate requests
    4 - Create new certificate request
    5 - Receive requested certificate or a renewal certificate
    6 - Create a self-signed certificate
    7 - Import a certificate
    8 - Import a certificate and a private key
    9 - Show the default key
   10 - Store database password
   11 - Show database record length

    0 - Exit program

Enter option number (press ENTER to return to previous menu):
6

       Certificate Usage

    1 - CA certificate
    2 - User or server certificate

Select certificate usage (press ENTER to return to menu):
1
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the Self-signed CA certificate (cont)

```
       RSA Key Size

   1 - 1024-bit key
   2 - 2048-bit key
   3 - 4096-bit key

Select RSA key size (press ENTER to return to menu):
2

       Signature Digest Type

   1 - SHA-1
   2 - SHA-224
   3 - SHA-256
   4 - SHA-384
   5 - SHA-512

Select digest type (press ENTER to return to menu):
5
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the Self-signed CA certificate (cont)

```
Enter label (press ENTER to return to menu):
ZVMCA
Enter subject name for certificate
  Common name (required):
zvmca

  Organizational unit (optional):
ITC

  Organization (required):
ITC

  City/Locality (optional):


  State/Province (optional):


  Country/Region (2 characters - required):
US

Enter number of days certificate will be valid (default 365):
365

Enter 1 to specify subject alternate names or 0 to continue:
0

Please wait .....

Certificate created.
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the CA-signed server certificate

From the Key Management Menu, select option 1 - Manage keys and certificates

```
     1 - Manage keys and certificates
     2 - Manage certificates
     3 - Manage certificate requests
     4 - Create new certificate request
     5 - Receive requested certificate or a renewal certificate
     6 - Create a self-signed certificate
     7 - Import a certificate
     8 - Import a certificate and a private key
     9 - Show the default key
    10 - Store database password
    11 - Show database record length

     0 - Exit program
  1
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the CA-signed server certificate (cont)

Then select “1” for ZVMCA

```
Enter option number (press ENTER to return to previous menu):

       Key and Certificate List

       Database: /etc/gskadm/Database.kdb

   1 - ZVMCA

   0 - Return to selection menu

Enter label number (ENTER to return to selection menu, p for previous
list):
1
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the CA-signed server certificate (cont)

```
        Key and Certificate Menu

        Label: ZVMCA

    1 - Show certificate information
    2 - Show key information
    3 - Set key as default
    4 - Set certificate trust status
    5 - Copy certificate and key to another database
    6 - Export certificate to a file
    7 - Export certificate and key to a file
    8 - Delete certificate and key
    9 - Change label
   10 - Create a signed certificate and key
   11 - Create a certificate renewal request

    0 - Exit program

 Enter option number (press ENTER to return to previous menu):

 10
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the CA-signed server certificate (cont)

Then select option 2

```
        Certificate Usage

    1 - CA certificate
    2 - User or server certificate
  Select certificate usage (press ENTER to return to menu):

 2
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Create the CA-signed server certificate (cont)

Then, following the same steps used in creating the CA certificate, enter the following data for the server certificate:

Key algorithm	– **RSA**
Key size	– **2048**
Label	– **SMBSSI**
Common name	– **smbssi**
Organizational unit	*(leave blank)*
Organization	– **ITC**
City Locality	– *(leave blank)*
State/Province	– *(leave blank)*
Country	– **US**
Validity	– **720**
Alternate names	– **0**

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Display certificate information

Information about certificates stored in the database can be displayed using Option 1 from the menu:

```
          Key and Certificate Menu

          Label: ZVMCA

      1 - Show certificate information
      2 - Show key information
      3 - Set key as default
      4 - Set certificate trust status
      5 - Copy certificate and key to another database
      6 - Export certificate to a file
      7 - Export certificate and key to a file
      8 - Delete certificate and key
      9 - Change label
     10 - Create a signed certificate and key
     11 - Create a certificate renewal request

      0 - Exit program

   Enter option number (press ENTER to return to previous menu):

   1
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Display certificate information

```
                  Certificate Information

              Label: ZVMCA
          Record ID: 11
   Issuer Record ID: 11
            Trusted: Yes
            Version: 3
      Serial number: 5d0a8f8000087035
        Issuer name: zvmca
                     ITC
                     ITC
                     US
       Subject name: zvmca
                     ITC
                     ITC
                     US
     Effective date: 2019/06/19
    Expiration date: 2020/06/18
Signature algorithm: sha512WithRsaEncryption
   Issuer unique ID: None
  Subject unique ID: None
Public key algorithm: rsaEncryption
    Public key size: 2048
         Public key: 30 82 01 0A 02 82 01 01 00 A1 26 8F 88 5F EC 6C
                     47 10 E6 2B DF 31 3D 7C C9 CE 31 EE 32 4B 44 13
                     8D 7F 77 F6 FC 97 B5 79 2B C9 BB 90 97 0E FA C2
                     C3 69 43 0B A0 0E 61 BB 50 CA BA 89 65 40 7B A7
                     71 C3 DD E3 02 93 87 24 F3 05 62 16 83 B8 67 B0
                     BC BF FE DF 07 02 80 3F 52 44 7A 70 DE CE 6F C7
                     E1 EA 69 0D 75 23 49 C7 C2 27 EB A7 81 A1 14 9A
                     EE C7 C6 1D CE E1 1A 90 24 7B 46 9F E2 6B 97 EE
                     CB 85 65 96 32 38 0F F1 B2 57 8C 26 BA 55 3E 4C
                     3D 00 83 4F 26 61 58 36 91 D9 15 09 7D DD 3B 28
                     B1 04 3A EB 8D 36 1D C2 6B 0F F7 EF 5A 64 DE C3
                     58 92 37 1A C5 84 97 96 A9 E0 B1 F7 4B FC 68 D0
                     E6 F3 D5 72 E2 4E 54 A6 5F A1 4E BE 87 2E 17 C6
                     FE 83 A0 BC D7 C5 8C 73 A8 A6 BB F5 AA CE 47 F8
                     7C CE 22 17 8E 8F DF AB F4 B4 5F 22 77 8C 3B 97
                     96 A5 31 A3 9F BA 51 77 82 BE 43 50 20 39 65 17
```

CREATE INTERNAL z/VM CERTIFICATE DATABASE

Display certificate information

```
                        10 FD 4B 08 DF D5 CF 36 A1 02 03 01 00 01
  Number of extensions: 4

Enter 1 to display extensions, 0 to return to menu:

1

       Certificate Extensions List

   1 - subjectKeyIdentifier
   2 - authorityKeyIdentifier
   3 - keyUsage (critical)
   4 - basicConstraints (critical)

Enter extension number (press ENTER to return to previous menu):
1

 49 DA C1 22 5E D6 FB 60 E3 74 C4 0D FE F4 25 85
 08 4D 9B 47

Press ENTER to continue.


       Certificate Extensions List

   1 - subjectKeyIdentifier
   2 - authorityKeyIdentifier
   3 - keyUsage (critical)
   4 - basicConstraints (critical)

Enter extension number (press ENTER to return to previous menu):
2

Key identifier:
 49 DA C1 22 5E D6 FB 60 E3 74 C4 0D FE F4 25 85
 08 4D 9B 47

Press ENTER to continue.
```

UPDATE z/VM TCP/IP CONFIGURATION

Topics

- Update the `SYSTEM DTCPARMS` file
- Update the `PROFILE TCPIP` file
- Restart `TCPIP`
- Check log file
- `QUERY NAMES`

UPDATE z/VM TCP/IP CONFIGURATION

Update the SYSTEM DTCPARMS file

Log onto the TCPMAINT user id. Insure that the TCPMAINT 198 mdisk is accessed as file mode D.

Edit the SYSTEM DTCPARMS file and add the following line.

```
XEDIT SYSTEM DTCPARMS D
```

```
.*********************************************************************
.* SYSTEM DTCPARMS created by DTCIPWIZ EXEC on 26 Dec 2016
.* Configuration program run by MAINT640 at 18:09:59
.*********************************************************************
:nick.TCPIP    :type.server
               :class.stack
.*             :attach.0800-0802
:DCSS_Parms.<DEFAULT>
```

UPDATE z/VM TCP/IP CONFIGURATION

Update the PROFILE TCPIP file

Edit the PROFILE TCPIP file and add the following lines.

```
XEDIT PROFILE TCPIP D
```

```
SSLSERVERID * TIMEOUT 60
; SSLLIMITS MAXSESSIONS 3000 MAXPERSSLSERVER 600
```

- The “*” wildcard is used to tell the TCP/IP server that the SSL servers are taken for the SSL server pool associated to the TCP/IP stack. This is the default pool with prefix SSL. Note that the prefix must not be specified in the statement, only the wildcard. The association between the TCP/IP server and the SSL server pool is established in the DTCPARMS file

- The timeout is the number of seconds to wait for the TCP/IP server before starting the other TCP/IP servers specified in the AUTOLOG statement. The default value is 30.

UPDATE z/VM TCP/IP CONFIGURATION

Restart the TCPIP server

From the TCPMAINT user id

```
FORCE TCPIP
XAUTOLOG TCPIP
```

UPDATE z/VM TCP/IP CONFIGURATION

Check the log file

```
.........
TCPIP   :  DTCRUN1038I Server is configured to support secure connections
TCPIP   :  DTCRUN1034I Associated SSL server pool: SSL*
.........
.........
TCPIP   :  DTCRUN1043I Initiating XAUTOLOG of server SSLDCSSM
.........
.........
SSLDCSSM:  HCPNSD440I Saved segment TCPIP was successfully defined in file
SSLDCSSM:  HCPNSS440I Saved segment TCPIP was successfully saved in file
.........
.........
TCPIP    :  11:02:10 DTCSSL044I SSL Server SSL00001 is available to handled secure
connections
:

TCPIP : 11:02:13 DTCSSL044I SSL Server SSL00003 is available to handle secure connections
TCPIP : 11:02:13 DTCSSL044I SSL Server SSL00004 is available to handle secure connections
TCPIP : 11:02:13 DTCSSL044I SSL Server SSL00002 is available to handle secure connections
TCPIP : 11:02:13 DTCSSL044I SSL Server SSL00005 is available to handle secure connections
```

UPDATE z/VM TCP/IP CONFIGURATION

QUERY NAMES

```
query names
DAVE     - 0200, EJAGGER  -L0005, PAULG    - DSC , PERFSVM  - DSC
MONWRITE - DSC , BATCH    - DSC , RSCSAUTH - DSC , RSCS     - DSC
RSCSDNS  - DSC , IPGATE   - DSC , GCS      - DSC , SSL00005 - DSC
SSL00004 - DSC , SSL00003 - DSC , SSL00002 - DSC , WEB390   - DSC
VMNFS    - DSC , REXECD   - DSC , PORTMAP  - DSC , FTPSERVE - DSC
ZVMSFS   - DSC , SSL00001 - DSC , SSLDCSSM - DSC , TCPIP    - DSC
DATAMOVE - DSC , DIRMAINT - DSC , DTCVSW4  - DSC , DTCVSW3  - DSC
DTCVSW2  - DSC , DTCVSW1  - DSC , VMSERVP  - DSC , VMSERVR  - DSC
VMSERVU  - DSC , VMSERVS  - DSC , OPERSYMP - DSC , DISKACNT - DSC
EREP     - DSC , OPERATOR - 0020, MAINT    -L0004
VSM     - TCPIP
Ready; T=0.01/0.01 07:22:36
```

RXSOCKET Updates

Topics

- New functions
- Syntax

RXSOCKET Updates

New Functions

The RXSOCKET routine now supports 5 new functions (or “cmds”) in the IOCTL call:

1)SIOCSECCLIENT	Start a secure TLS session for a client
2)SIOCSECSERVER	Start a secure TLS session for a server
3)SIOCTLSQRY	Determine if a TLS/SSLserver is available
4)SIOCSECCLOSE	Stop a secure TLS session
5)SIOCSECSTATUS	Request details about a session

(All done by Perry in under 2 hours….)

RXSOCKET Updates

New Functions

First, verify that the correct version of the RXSOCKET module is being used:

```
    'NUCXDROP RXSOCKET'
    rxsversion = socket("Version")
  say 'rxsocket version:' rxsversion
```

```
rxsocket version: REXX/SOCKETS 3.04 12 April 1996 TLS
```

RXSOCKET Updates

Syntax

```
SOCKET (‘IOCTL’,  sock_id, ’SIOCSECCLIENT’, SecDetail_struct)
```

sock_id is the identifier of the socket.
SecDetail_struct is the following data structure

```
TLSLabel          DS  CL8
TLStimeout        DS  F
requestClientCert DS  FL1
validatePeerCert  DS  X
cipher_request    DS  X
reserved1         DS  X
keyring           DS  CL50
buflen            DS  H
buffer            DS  CL255
```

`TLStimeout` -	currently not used and must be 0
`requestClientCert` -	currently not implemented and must be 0
`validatePeerCert` -	client only - 0 = Full Check; 1 =No Check
`cipher_request` -	may use SSLV2? 0 = default cipher suite used; 1 = V2 is not allowed
`keyring` -	currently not used and must be blank

RXSOCKET Updates

Syntax

```
SOCKET (‘IOCTL’,  sock_id, ’SIOCSECSERVER’, SecDetail_struct)
```

sock_id is the identifier of the socket.
SecDetail_struct is the following data structure

```
TLSLabel          DS   CL8
TLStimeout        DS   F
requestClientCert DS   FL1
validatePeerCert  DS   X
cipher_request    DS   X
reserved1         DS   X
keyring           DS   CL50
buflen            DS   H
buffer            DS   CL255
```

`TLStimeout` -	currently not used and must be 0
`requestClientCert` -	currently not implemented and must be 0
`validatePeerCert` -	client only - 0 = Full Check; 1 =No Check
`cipher_request` -	may use SSLV2? 0 = default cipher suite used; 1 = V2 is not allowed
`keyring` -	currently not used and must be blank

RXSOCKET Updates

Syntax

```
SOCKET (‘IOCTL’,  sock_id, ’SIOCTLSQRY’, QueryTLS_struct)
```

sock_id is the identifier of the socket.
QueryTLS_struct is the following data structure

```
TLSLabel        DS    CL8
TLSKeyring      DS    CL50
```

TLSkeyring -	currently not used and must be blank

RXSOCKET Updates

Syntax

```
SOCKET (‘IOCTL’,  sock_id, ’SIOCSECCLOSE’, CloseReq_struct)
```

sock_id is the identifier of the socket.
CloseReq_struct is the following data structure

```
CloseLen        DS H
CloseBuff       DS CL255
```

RXSOCKET Updates

Syntax

```
SOCKET ('IOCTL',  sock_id, 'SIOCSECSTATUS') with rc SecStatus
```

sock_id is the identifier of the socket.
rc is the return code
SecStatus is the following data structure

```
SecLevel            DS   F
CipherClass         DS   X
CipherHash          DS   X
CipherAlgorithm     DS   X
CipherPKAlgorithm   DS   X
CipherKeyLength     DS   F
```

```
SecLevel:
     0 = Not Secure, 1 = Statically Secured, 2 =Dynamically Secured
CipherClass:
     0 = NULLclass, 1 = SSLV2, 2 = SSLV3, 3 = TLS,4=TLS10, 5=TLS11, 6=TLS12
CipherHash:
     0 = SHA1, 1 = MD5, 2 = NULL, 3 = SHA2, 4 =SHA256, 5 = SHA384
CipherAlgorithm:
     0 = NULL, 2 = RC4, 4 = DES3, 7 = AES, 8 = AESGCM, 9 = AES128,
    10 = AES128GCM, 11 = AES256, 12 =AES256GCM
CipherPKAlgorithm:
     0 = NULL, 1 = RSA, 2 = DH_DSS, 3 = DH_RSA, 4 =DHE_DSS, 5 = DHE_RSA,
     6 = ECDH_ECDSA, 7 =ECDHE_ECDSA, 8 = ECDH_RSA, 9 = ECDHE_RSA
```

Examples

Topics

- Useful Utility functions
- `RSCLIENT/RSSERVER`
- `IPGATE`

Examples

Useful Utility functions

```
DisplaySecStatus: procedure

  parse arg SecLevel +4 CipherClass +1 CipherHash +1 CipherAlgorithm +1,
            CipherPKAlgorithm +1 CipherKeyLength +4 .
  say "SecLevel:" c2d(SecLevel,4)
  say "CipherClass:" c2d(CipherClass,1)
  say "CipherHash:" c2d(CipherHash,1)
  say "CipherAlgorithm:" c2d(CipherAlgorithm,1)
  say "CipherPKAlgorithm:" c2d(CipherPKAlgorithm,1)
  say "CipherKeyLength:" c2d(CipherKeyLength,4)

return
```

Examples

Useful Utility functions

```
BuildSecureDetail: procedure expose tlslabel
return left(tlslabel,8)||,
       '00000000'x||,
       '00'x||,
       '00'x||,
       '00'x||,
       '00'x||,
       left(' ',50)||,
       '0000'x||,
       left(' ',255)
```

Examples

Useful Utility functions

```
BuildQueryTLS: procedure expose tlslabel
return left(tlslabel,8)||,
       left(' ',50)

BuildCloseReq: procedure
return '0000'x||,
       Left(' ',255)
```

Examples

RSCLIENT/RSSERVER

```
/*- RSCLIENT -- Example demonstrating the usage of REXX Sockets ------*/
/**********************************************************************/
/* (c) Copyright IBM Corporation 1996                                 */
/* This programming example is to be used as a sample program only.   */
/* Although this program may have been reviewed by IBM for accuracy,  */
/* there is no guarantee that it is totally free from defects.  The   */
/* code is being provided on an 'as is' basis without any warranty    */
/* expressed or implied.                                              */
/*                                                                    */
/**********************************************************************/
/*                                                                    */
  'NUCXDROP RXSOCKET'
trace o
signal on syntax

/* Set error code values                                              */
ecpref = 'RXS'
ecname = 'CLI'
initialized = 0

parse arg argstring
argstring = strip(argstring)
if substr(argstring,1,1) = '?' then do
  say 'RSSERVER and RSCLIENT  are a pair of programs which provide an'
  say 'example of how to use REXX/SOCKETS to implement a service. The'
  say 'server must be started before the clients get started.        '
  say '                                                              '
  say 'The RSSERVER program runs in its own dedicated virtual machine'
  say 'and returns a number of data lines as requested to the client.'
  say 'It is started with the command:                               '
  say '    RSSERVER                                                  '
  say 'and terminated with the command:                              '
  say '    HX                                                        '
  say '                                                              '
  say 'The RSCLIENT program is used  to request a number of arbitrary'
  say 'data lines  from the  server  and can be run  concurrently any'
  say 'number  of times  by different  clients  until  the server  is'
  say 'terminated.  It is started with the command:                  '
  say '    RSCLIENT number <server>                                  '
  say 'where "number" is the number of data lines to be requested and'
  say '"server" is the ipaddress of the service virtual machine. (The'
  say 'default ipaddress is the one of the host  on which RSCLIENT is'
  say 'running, assuming that RSSERVER runs on the same host.)       '
  exit 100
end

/* Split arguments into parameters and options                        */
parse upper var argstring parameters '(' options ')' rest

/* Parse the parameters                                               */
parse var parameters lines server rest
if ¬datatype(lines,'W') then call error 'E', 24, 'Invalid number'
lines = lines + 0
if rest¬='' then call error 'E', 24, 'Invalid parameters'
```

```
/* Parse the options                                                  */
do forever
  parse var options token options
  select
    when token='' then leave
    otherwise call error 'E', 20, 'Invalid option "'token'"'
  end
end

/* Initialize control information                                     */
port = '1952'               /* The port used by the server            */
address command 'IDENTIFY ( LIFO'
parse upper pull userid . locnode .
  tlslabel = "SMBSSI"

  rxsversion = socket("Version")
  if subword(rxsversion,words(rxsversion)) <> "TLS" then do
    say "RXSOCKETs does not contain TLS support"
    exit 8
  end


/* Initialize                                                         */
  call Socket 'Initialize',tlslabel
if src=0 then initialized = 1
else call error 'E', 200, 'Unable to initialize RXSOCKET MODULE'
if server='' then do
  server = Socket('GetHostId')
  if src¬=0 then call error 'E', 200, 'Cannot get the local ipaddress'
end
ipaddress = server

/* Initialize for receiving lines sent by the server                  */
s = Socket('Socket')
if src¬=0 then call error 'E', 32, 'SOCKET(SOCKET) rc='src
call Socket 'Connect', s, 'AF_INET' port ipaddress
if src¬=0 then call error 'E', 32, 'SOCKET(CONNECT) rc='src
  Call Socket "Ioctl",s,"SIOCSECCLIENT",BuildSecureDetail()
'
if src¬=0 then call error 'E', 32, 'SIOCSECCLIENT rc='src
  Call Socket "Ioctl",s,"SIOCSECSTATUS"
  say '1 RC:' src
  say '1 res' c2x(res)
  if src = 0 then
    call DisplaySecStatus res


  Call Socket "Ioctl",s,"SIOCTLSQUERY",BuildQueryTLS()
  say '2 query RC:' src
if src¬=0 then call error 'E', 32, 'SIOCTLSQUERY rc='src

call Socket 'Write', s, locnode userid lines
if src¬=0 then call error 'E', 32, 'SOCKET(WRITE) rc='src

/* Wait for lines sent by the server                                  */
```

Examples

RSCLIENT/RSSERVER

```
dataline = ''
num = 0
do forever

  /* Receive a line and display it                                    */
  parse value Socket('Read', s) with len newline
  if src¬=0 | len<=0'' then leave
  dataline = dataline || newline
  do forever
    if pos('15'x,dataline)=0 then leave
    parse var dataline nextline '15'x dataline
    num = num + 1
    say right(num,5)':' nextline
  end
end

/* Terminate and exit                                                 */
  Call Socket "Ioctl",s,"SIOCSECCLOSE",BuildCloseReq()

call Socket 'Terminate'
exit 0

DisplaySecStatus: procedure

  parse arg SecLevel +4 CipherClass +1 CipherHash +1 CipherAlgorithm +1,
            CipherPKAlgorithm +1 CipherKeyLength +4 .
  say "SecLevel:" c2d(SecLevel,4)
  say "CipherClass:" c2d(CipherClass,1)
  say "CipherHash:" c2d(CipherHash,1)
  say "CipherAlgorithm:" c2d(CipherAlgorithm,1)
  say "CipherPKAlgorithm:" c2d(CipherPKAlgorithm,1)
  say "CipherKeyLength:" c2d(CipherKeyLength,4)

return

BuildQueryTLS: procedure expose tlslabel
return left(tlslabel,8)||,
       left(' ',50)

BuildCloseReq: procedure
return '0000'x||,
       left(' ',255)

BuildSecureDetail: procedure expose tlslabel
return left(tlslabel,8)||,
       '00000000'x||,
       '00'x||,
       '00'x||,
       '00'x||,
       '00'x||,
       left(' ',50)||,
       '0000'x||,
       left(' ',255)
/* Calling the real SOCKET function                                   */
socket: procedure expose initialized src
```

```
  a0 = arg(1)
  a1 = arg(2)
  a2 = arg(3)
  a3 = arg(4)
  a4 = arg(5)
  a5 = arg(6)
  a6 = arg(7)
  a7 = arg(8)
  a8 = arg(9)
  a9 = arg(10)
  parse value 'SOCKET'(a0,a1,a2,a3,a4,a5,a6,a7,a8,a9) with src res
return res

/* Syntax error routine                                               */
syntax:
  call error 'E', rc, '==> REXX Error No.' 20000+rc
return

/* Error message and exit routine                                     */
error: procedure expose ecpref ecname initialized
  type = arg(1)
  retc = arg(2)
  text = arg(3)
  ecretc = right(retc,3,'0')
  ectype = translate(type)
  ecfull = ecpref || ecname || ecretc || ectype
  address command 'EXECIO 1 EMSG (CASE M STRING' ecfull text
  if type¬='E' then return
  if initialized then do
    parse value Socket('SocketSetStatus') with . status severreason
    if status¬='Connected' then do
      say 'The status of the socket set is' status severreason
    end
  Call Socket "Ioctl",s,"SIOCSECCLOSE",BuildCloseReq()
    call Socket 'Terminate'
  end
exit retc
```

Examples

RSCLIENT/RSSERVER

```
/*- RSSERVER -- Example demonstrating the usage of REXX Sockets ------*/
/*********************************************************************/
/* (c) Copyright IBM Corporation 1996                                */
/* This programming example is to be used as a sample program only.  */
/* Although this program may have been reviewed by IBM for accuracy, */
/* there is no guarantee that it is totally free from defects.  The  */
/* code is being provided on an 'as is' basis without any warranty   */
/* expressed or implied.                                             */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
trace o
signal on syntax
signal on halt
  'NUCXDROP RXSOCKET'

  tlslabel = "SMBSSI"

  rxsversion = socket("Version")
  if subword(rxsversion,words(rxsversion)) <> "TLS" then do
    say "RXSOCKETs does not contain TLS support"
    exit 8
  end

/* Set error code values                                             */
initialized = 0

parse arg argstring
argstring = strip(argstring)
if substr(argstring,1,1) = '?' then do
  say 'RSSERVER and RSCLIENT  are a pair of programs which provide an'
  say 'example of how to use REXX/SOCKETS to implement a service. The'
  say 'server must be started before the clients get started.       '
  say '                                                             '
  say 'The RSSERVER program runs on a VM Userid.                    '
  say 'It returns a number of data lines as requested to the client. '
  say 'It is started with the command:  RSSERVER                    '
  say 'and terminated by issuing HX.                                '
  say '                                                             '
  say 'The RSCLIENT program is used  to request a number of arbitrary'
  say 'data lines  from the server.  One or more clients can access  '
  say 'the server until it is terminated.                           '
  say 'It is started with the command:  RSCLIENT number <server>    '
  say 'where "number" is the number of data lines to be requested and'
  say '"server" is the ipaddress of the service virtual machine. (The'
  say 'default ipaddress is the one of the host  on which RSCLIENT is'
  say 'running, assuming that RSSERVER runs on the same host.)       '
  say '                                                             '
  exit 100
end

/* Split arguments into parameters and options                       */
parse upper var argstring parameters '(' options ')' rest

/* Parse the parameters                                              */
```

```
parse var parameters rest
if rest¬='' then call error 'E', 24, 'Invalid parameters specified'

/* Parse the options                                                  */
do forever
  parse var options token options
  select
    when token='' then leave
    otherwise call error 'E', 20, 'Invalid option "'token'"'
  end
end

/* Initialize control information                                     */
port = '1952'                 /* The port used for the service        */

/* Initialize                                                         */
say 'RSSERVER: Initializing'
call Socket 'Initialize', 'RSSERVER'
if src=0 then initialized = 1
else call error 'E', 200, 'Unable to initialize SOCKET'
ipaddress = Socket('GetHostId')
if src¬=0 then call error 'E', 200, 'Unable to get the local ipaddress'
say 'RSSERVER: Initialized: ipaddress='ipaddress 'port='port
/* Initialize for accepting connection requests                       */
s = Socket('Socket')
if src¬=0 then call error 'E', 32, 'SOCKET(SOCKET) rc='src

call Socket 'Bind', s, 'AF_INET' port ipaddress
if src¬=0 then call error 'E', 32, 'SOCKET(BIND) rc='src
call Socket 'Ioctl', s, 'FIONBIO', 'ON'
if src¬=0 then call error 'E', 36, 'Cannot set mode of socket' s

call Socket 'Listen', s, 10
if src¬=0 then call error 'E', 32, 'SOCKET(LISTEN) rc='src

/* Wait for new connections and send lines                            */
timeout    = 60
linecount. =  0
wlist = ''
do forever

  /* Wait for an event                                                */
  if wlist¬='' then sockevtlist = 'Write'wlist 'Read * Exception'
  else sockevtlist = 'Write Read * Exception'
  sellist = Socket('Select',sockevtlist,timeout)
  if src¬=0 then call error 'E', 36, 'SOCKET(SELECT) rc='src
  parse upper var sellist . 'READ' orlist 'WRITE' owlist 'EXCEPTION' .
  if orlist¬='' | owlist¬='' then do
     event = 'SOCKET'
     if orlist¬='' then do
        parse var orlist orsocket .
        rest = 'READ' orsocket
     end
     else do
       parse var owlist owsocket .
```

Examples

RSCLIENT/RSSERVER

```
    rest = 'WRITE' owsocket
  end
end
else event = 'TIME'

select

 /* Accept connections from clients, receive and send messages     */
 when event='SOCKET' then do
   parse var rest keyword ts .

   /* Accept new connections from clients                          */
   if keyword='READ' & ts=s then do
     nsn = Socket('Accept',s)
     if src=0 then do
       parse var nsn ns . np nia .
       say 'RSSERVER: Connected by' nia 'on port' np 'and socket' ns
     call socket "Ioctl",ns,"SIOCSECSERVER",BuildSecureDetail()
      say src
     end
   end

   /* Get nodeid, userid and number of lines to be sent            */
   if keyword='READ' & ts¬=s then do
     parse value Socket('Recv',ts) with len nid uid count .
     if src=0 & len>0 & datatype(count,'W') then do
       if count<0 then count = 0
       if count>5000 then count = 5000
       ra = 'by' uid 'at' nid
       say 'RSSERVER: Request for' count 'lines on socket' ts ra
       linecount.ts = linecount.ts + count
       call addsock(ts)
     end
     else do
       call Socket 'Close',ts
       linecount.ts = 0
       call delsock(ts)
       say 'RSSERVER: Disconnected socket' ts
     end
   end

   /* Get nodeid, userid and number of lines to be sent            */
   if keyword='WRITE' then do
     if linecount.ts>0 then do
       num = random(1,sourceline())      /* Return random-selected */
       msg = sourceline(num) || '15'x    /*   line of this program */
       call Socket 'Send',ts,msg
       if src=0 then linecount.ts = linecount.ts - 1
       else linecount.ts = 0
     end
     else do
       call Socket 'Close',ts
       linecount.ts = 0
       call delsock(ts)
       say 'RSSERVER: Disconnected socket' ts
```

```
        end
      end

    end

    /* Unknown event (should not occur)                               */
    otherwise nop
  end
end

/* Terminate and exit                                                 */
  parse value socket("Ioctl",socid,"SIOCSECCLOSE",BuildCloseReq()) with rc rest
  say 'RC:' rc 'rest' rest
call Socket 'Terminate'
say 'RSSERVER: Terminated'
exit 0

/* Procedure to add a socket to the write socket list                 */
addsock: procedure expose wlist
  s = arg(1)
  p = wordpos(s,wlist)
  if p=0 then wlist = wlist s
return

/* Procedure to del a socket from the write socket list               */
delsock: procedure expose wlist
  s = arg(1)
  p = wordpos(s,wlist)
  if p>0 then do
    templist = ''
    do i=1 to words(wlist)
      if i¬=p then templist = templist word(wlist,i)
    end
    wlist = templist
  end
return

/* Calling the real SOCKET function                                   */
socket: procedure expose initialized src
  a0 = arg(1)
  a1 = arg(2)
  a2 = arg(3)
  a3 = arg(4)
  a4 = arg(5)
  a5 = arg(6)
  a6 = arg(7)
  a7 = arg(8)
  a8 = arg(9)
  a9 = arg(10)
  parse value 'SOCKET'(a0,a1,a2,a3,a4,a5,a6,a7,a8,a9) with src res
return res


/* Syntax error routine                                               */
syntax:
```

Examples

RSCLIENT/RSSERVER

```
  call error 'E', rc, '==> REXX Error No.' 20000+rc
return

/* Halt exit routine                                                  */
halt:
  call error 'E', 4, '==> REXX Interrupted'
return

/* Error message and exit routine                                     */
error:
  type = arg(1)
  retc = arg(2)
  text = arg(3)
  ecretc = right(retc,3,'0')
  ectype = translate(type)
  ecfull = 'RXSSRV' || ecretc || ectype
  say '===> Error:' ecfull text
  if type¬='E' then return
  if initialized
     then do
       parse value Socket('SocketSetStatus') with . status severreason
       if status¬='Connected'
          then say 'The status of the socket set is' status severreason
     End
  Call Socket "Ioctl",s,"SIOCSECCLOSE",BuildCloseReq()
  call Socket 'Terminate'
exit retc
DisplaySecStatus: procedure

  parse arg SecLevel +4 CipherClass +1 CipherHash +1 CipherAlgorithm +1,
            CipherPKAlgorithm +1 CipherKeyLength +4 .
  say "SecLevel:" c2d(SecLevel,4)
  say "CipherClass:" c2d(CipherClass,1)
  say "CipherHash:" c2d(CipherHash,1)
  say "CipherAlgorithm:" c2d(CipherAlgorithm,1)
  say "CipherPKAlgorithm:" c2d(CipherPKAlgorithm,1)
  say "CipherKeyLength:" c2d(CipherKeyLength,4)

return

BuildQueryTLS: procedure expose tlslabel
return left(tlslabel,8)||,
       left('',50)

BuildCloseReq: procedure
return '0000'x||,
       left('',255)

BuildSecureDetail: procedure expose tlslabel
return left(tlslabel,8)||,
       '00000000'x||,
       '00'x||,
       '00'x||,
       '00'x||,
       '00'x||,
       left('',50)||,
       '0000'x||,
       left('',255)
```

Examples

IPGATE

The SSL/TLS enabled version of IPGATE will be included on the VM Workshop tools tape. See the comments in the code for additional information.

This version will also have an update by Perry to fix a memory leak error discovered by Berry van Sleeuwen.

Thanks for your time!

Questions?

Perry Ruiter
360toz@hushmail.com

Dave Jones
vmdave9@gmail.com