

libtnc

Copyright (C) 2008

Mike McCauley

A TNC implementation for Windows and
Unix. For libtnc version 1.19

1.0 Introduction

libtnc is an open-source implementation of the Trusted Network Connect (TNC) specification, developed by the Trusted Computing Group (TCG).

TNC is a specification for protocols and APIs that allows network authentication services (such as RADIUS servers) to assess the security posture of a client device wanting to connect to a network (such as a wireless PC) before it gets access to the network. The motivation for this is to check that the client PC is safe and secure (i.e. that software is up to date, the firewall is running and that the anti-virus software is installed, running and up-to-date. TNC implementations are typically found in 802.1X wireless supplicants and RADIUS servers. Details of TNC can be found at

<https://www.trustedcomputinggroup.org/specs/TNC>

libtnc is an open-source implementation of these TNC specifications which compiles and runs on Windows and many Unix-like systems. It also includes the sources of a generic IMC/IMV pair which can assess basic security posture for Windows and Unix clients. It can be used to build TNC compliant clients, such as 802.1X supplicants and TNC compliant servers, such as RADIUS server. It can also be used to build a simple INC/IMV pair that can be used to assess the security posture of Windows and Unix clients from a Windows or Unix server.

This document describes the source, how to use libtnc to implement TNC compliant supplicants and servers, and how to configure the sample generic IMC/IMV pair.

2.0 Coverage

The following TNC specifications are supported by libtnc APIs:

- IF-IMC 1.2
- IF-IMV 1.2
- IF-TNCCS 1.1

In addition the following components are provided

- A fully functional OSC-IMC for Windows and Unix, which can answer some simple posture queries sent by the matching IMV below.
- A fully functional OSC-IMV for Windows and Unix, which can be configured to do arbitrarily complicated posture assessment via a scriptable policy file, and works with the IMC above.
- Sample skeleton IMC and IMVs that can serve as a starting point for your own IMC/IMV pair.
- Perl bindings for IF-IMC, IF-IMV and IF-TNCCS

3.0 Building and Installing

3.1 Unix

Can be used for any Unix, Linux or Cygwin environment. Requires standard Unix development tools, gcc, make etc. Requires GNU libxml2 and GNU libiconv.

1. Download the libtnc distribution file, typically named something like libtnc-n.nn.tar.gz to a work directory.
2. tar zxvf libtnc-n.nn.tgz
3. cd libtnc-n.nn
4. ./configure
5. make
6. make check
7. make install

The loadable OSC-IMC module will be in src/osc/.libs/libosc_imc.so. The loadable OSC-IMV module will be in src/osc/.libs/libosc_imv.so.

3.2 Windows

Requires Visual C++ 2005 Express or better. Requires GNU libxml2-2.6.30.win32 and iconv-1.9.2.win32 installed in C:\gnu. Get binaries for these required libraries from <http://www.zlatkovic.com/pub/libxml>

1. Download the libtnc distribution file, typically named something like libtnc-n.nn.tar.gz to a work directory.
2. Using WinZip or similar application, unpack the distribution into the work directory

3. Using My Computer or similar, navigate to the work directory then to libtnc-n.nn\vs2005\libtnc
4. Double-click on the libtnc.sln solution file. Visual C++ window will appear with the libtnc solution.
5. Select Build->Build Solution. All the components, including the OSC-IMC and OSC-IMV dlls will be created.

The loadable OSC-IMC will be in vs2005\libtnc\Release\osc_imc.dll. The loadable OSC-IMV will be in vs2005\libtnc\Release\osc_imv.dll.

4.0 IF-IMC

libtnc implements the following C binding calls to and from IMCs as required by IF-IMC:

- TNC_IMC_Initialize
- TNC_IMC_NotifyConnectionChange
- TNC_IMC_BeginHandshake
- TNC_IMC_ReceiveMessage
- TNC_IMC_BatchEnding
- TNC_IMC_ProvideBindFunction
- TNC_IMC_Terminate
- TNC_TNCC_ReportMessageTypes
- TNC_TNCC_RequestHandshakeRetry
- TNC_TNCC_BindFunction
- TNC_TNCC_SendMessage

4.1 C Function Calls

A number of C functions are available to load a set of IMCs and to interoperate with them:

4.1.1 libtnc_imc_load_config

```
int libtnc_imc_load_config(const char* filename)
```

4.1.2 libtnc_imc_load_std_config

```
int libtnc_imc_load_std_config()
```

4.1.3 libtnc_imc_load_modules

```
TNC_Result libtnc_imc_load_modules  
(const char* filenames[],  
TNC_UInt32 numfiles)
```

4.1.4 libtnc_imc_unload

```
TNC_Result libtnc_imc_unload()
```

4.1.5 libtnc_imc_NotifyConnectionChange

```
TNC_Result libtnc_imc_NotifyConnectionChange  
(TNC_ConnectionID connectionID,  
TNC_ConnectionState newState)
```

4.1.6 libtnc_imc_BeginHandshake

```
TNC_Result libtnc_imc_BeginHandshake  
(TNC_ConnectionID connectionID)
```

4.1.7 libtnc_imc_ReceiveMessage

```
TNC_Result libtnc_imc_ReceiveMessage  
(TNC_ConnectionID connectionID,  
TNC_BufferReference messageBuffer,  
TNC_UInt32 messageLength,  
TNC_MessageType messageType)
```

4.1.8 libtnc_imc_BatchEnding

```
TNC_Result libtnc_imc_BatchEnding  
(TNC_ConnectionID connectionID)
```

4.1.9 libtnc_imc_Terminate

```
TNC_Result libtnc_imc_Terminate()
```

4.2 Vendor Extensions**4.2.1 TNC_9048_LogMessage**

If the IMC requests a binding for this function, it will be given a pointer to a logging function which can be supplied by the calling application:

```
TNC_Result libtnc_logMessage  
(TNC_UInt32 severity,  
const char* format,  
...)
```

This varargs style function may be called when the IMC wishes to log a message through the calling application's logging system. Severity is one of the following constants defined in libtnc.h

- TNC_LOG_SEVERITY_ERR (0)
- TNC_LOG_SEVERITY_WARNING (1)
- TNC_LOG_SEVERITY_NOTICE (2)
- TNC_LOG_SEVERITY_INFO (3)
- TNC_LOG_SEVERITY_DEBUG (4)

The default implementation of `libtnc_logMessage()` provided with `libtnc` simply logs the message and arguments to `stderr` using `vfprintf()`. severity is ignored. If the calling application defines its own `libtnc_logMessage()`, it will override the default implementation.

5.0 IF-IMV

`libtnc` implements the following C binding calls to and from IMVs as required by IF-IMV:

- `TNC_IMV_Initialize`
- `TNC_IMV_NotifyConnectionChange`
- `TNC_IMV_ReceiveMessage`
- `TNC_IMV_BatchEnding`
- `TNC_IMV_Terminate`
- `TNC_IMV_ProvideBindFunction`
- `TNC_TNCS_ReportMessageTypes`
- `TNC_TNCS_BindFunction`
- `TNC_TNCS_RequestHandshakeRetry`
- `TNC_TNCS_ProvideRecommendation`
- `TNC_TNCS_SendMessage`
- `TNC_TNCS_GetAttribute`
- `TNC_TNCS_SetAttribute`

5.1 C Function Calls

A number of C functions are available to load a set of IMVs and to interoperate with them:

5.1.1 `libtnc_imv_load_config`

```
int libtnc_imv_load_config(const char* filename)
```

5.1.2 `libtnc_imv_load_std_config`

```
int libtnc_imv_load_std_config()
```

5.1.3 `libtnc_imv_load_modules`

```
TNC_Result libtnc_imv_load_modules  
    (const char* filenames[],  
     TNC_UInt32 numfiles)
```

5.1.4 `libtnc_imv_unload`

```
TNC_Result libtnc_imv_unload()
```

5.1.5 libtnc_imv_NotifyConnectionChange

```
TNC_Result libtnc_imv_NotifyConnectionChange
    (TNC_ConnectionID connectionID,
     TNC_ConnectionState newState)
```

5.1.6 libtnc_imv_SolicitRecommendation

```
TNC_Result libtnc_imv_SolicitRecommendation
    (TNC_ConnectionID connectionID)
```

5.1.7 libtnc_imv_ReceiveMessage

```
TNC_Result libtnc_imv_ReceiveMessage
    (TNC_ConnectionID connectionID,
     TNC_BufferReference messageBuffer,
     TNC_UInt32 messageLength,
     TNC_MessageType messageType)
```

5.1.8 libtnc_imv_BatchEnding

```
TNC_Result libtnc_imv_BatchEnding
    (TNC_ConnectionID connectionID)
```

5.1.9 libtnc_imv_Terminate

```
TNC_Result libtnc_imv_Terminate()
```

5.2 Vendor Extensions**5.2.1 TNC_9048_LogMessage**

If the IMC requests a binding for this function, it will be given a pointer to a logging function which can be supplied by the calling application, as described in “TNC_9048_LogMessage” on page 4

6.0 IF-TNCCS TNCC

This API implements the functions required by IF-TNCCS on the TNC Client side.

6.1 C Bindings for calls into TNCC**6.1.1 libtnc_tncc_Initialize**

```
TNC_Result libtnc_tncc_Initialize(const char* filename)
```

Alternative to libtnc_tncc_InitializeStd. Reads the list of IMCs from the given filename (even on Windows)

6.1.2 libtnc_tncc_InitializeStd

```
TNC_Result libtnc_tncc_InitializeStd()
```

Initialize the TNCC and IMC layers. Reads the list of IMCs from the standard places according to IF-IMC. On Windows, reads from the Registry at HKLM\SOFTWARE\Trusted Computing Group\TNC\IMCs. On Unix, reads /etc/tnc_config

6.1.3 libtnc_tncc_PreferredLanguage

```
TNC_Result libtnc_tncc_PreferredLanguage(const char* language)
```

Sets the clients preferred language.

6.1.4 libtnc_tncc_Terminate

```
TNC_Result libtnc_tncc_Terminate()
```

Terminates the TNCC layer and unloads all IMCs.

6.1.5 libtnc_tncc_CreateConnection

```
libtnc_tncc_connection*  
libtnc_tncc_CreateConnection(void* appData)
```

Creates a new connection. The returned pointer should be considered as a pointer to an opaque structure. Returns NULL on failure. The resulting pointer should be passed to other libtnc_tncc calls. The appData may be used by the applicaiton to associate its own data with a libtnc_tncc_connection. It can be recovered with libtnc_tncc_ConnectionAppData() and is not used by the TNCC internals.

6.1.6 libtnc_tncc_ConnectionAppData

```
void* libtnc_tncc_ConnectionAppData  
(libtnc_tncc_connection* conn)
```

Recover the appData that was passed to libtnc_tncc_CreateConnection().

6.1.7 libtnc_tncc_DeleteConnection

```
TNC_ConnectionID libtnc_tncc_DeleteConnection  
(libtnc_tncc_connection* conn)
```

Delete a previously established connection.

6.1.8 libtnc_tncc_BeginSession

```
TNC_Result libtnc_tncc_BeginSession  
(libtnc_tncc_connection* conn)
```

Start a new session on a previously created connection. Calls libtnc_imc_NotifyConnectionChange(TNC_CONNECTION_STATE_CREATE) and libtnc_imc_NotifyConnectionChange(TNC_CONNECTION_STATE_HANDSHAKE) of the IMCs, which will usually trigger the transmission of the first IF-TNCC batch by calling TNC_TNCC_SendBatch

6.1.9 libtnc_tncc_ReceiveBatch

```
TNC_Result libtnc_tncc_ReceiveBatch
    (libtnc_tncc_connection* conn,
     const char* messageBuffer,
     size_t messageLength)
```

Passes a message batch to the TNCC. The messages within the batch will be passed to IMCs that have registered interest in those types of message. You must call libtnc_tncc_EndBatch() afterwards in order to trigger any messages to be sent to the TNCS.

6.2 C Bindings for calls made out by TNCC

The following function calls will be made by TNCC in response to activities by the IMCs and IF-IMC layer. The calling application is required to implement them:

6.2.1 TNC_TNCC_RequestHandshakeRetry

```
TNC_Result TNC_TNCC_RequestHandshakeRetry
    (TNC_IMCID imcID,
     TNC_ConnectionID connectionID,
     TNC_RetryReason reason)
```

Called if the IMV layer determines the need to redo the TNC handshake.

6.2.2 TNC_TNCC_SendBatch

```
TNC_Result TNC_TNCC_SendBatch
    (libtnc_tncc_connection* conn,
     const char* messageBuffer,
     size_t messageLength)
```

Called by the TNCC layer from within libtnc_tncc_EndBatch() in order to send a completed batch to the TNCS server. You must implement this to pass the batch to the TNCS, where it would be passed to libtnc_tncc_ReceiveBatch().

6.2.3 libtnc_tncc_BindFunction

```
TNC_Result libtnc_tncc_BindFunction
    (TNC_IMCID imcID,
     char *functionName,
     void **pOutfunctionPointer)
```

This function is called by TNC_TNCC_BindFunction when an IMC wish to discover an address of a function it wishes to call. The default implementation just returns TNC_RESULT_INVALID_PARAMETER, but it may be overridden by the calling application to provide its own implementations of IMC functions or to add new ones that the IMC may call.

6.2.4 `libtnc_logMessage`

```
TNC_Result libtnc_logMessage
    (TNC_UInt32 severity,
     const char* format, ...)
```

The default implementation logs the message to stderr. The default implementation will be linked if the calling application does not provide an implementation.

6.3 Coding a TNCC Client

A typical TNCC client will usually initialize the TNCC and IMC layers, loading all the configured IMCs at startup:

```
libtnc_tncc_InitializeStd();
libtnc_tncc_PreferredLanguage("en");
```

The client application will usually make at the beginning of a new TNCC-TNCS connection:

```
myImcConn = libtnc_tncc_CreateConnection(NULL);
libtnc_tncc_BeginSession(myImcConn);
```

Note: During the call to `libtnc_tncc_BeginSession`, `TNC_TNCC_SendBatch()` will probably be called to send the first batch of messages to the TNCS. The client application must implement the transmission and delivery of the batch to the TNCS.

Whenever a message batch is received from the TNCS, it is passed to the TNCC layer with the following code. During a TNC handshake there may be a number of batches received. The receipt of a batch will usually trigger the transmission of a reply batch by calling `TNC_TNCC_SendBatch()`.

```
libtnc_tncc_ReceiveBatch(myImcConn,
    messageBuffer, messageLength);
```

After the TNC handshake is complete and network access has been granted, the client application calls:

```
libtnc_tncc_DeleteConnection(myImcConn);
```

7.0 IF-TNCCS TNCS

This API implements the functions required by IF-TNCCS on the TNC Server side.

7.1 C Bindings for calls into TNCS

7.1.1 `libtnc_tncc_Initialize`

```
TNC_Result libtnc_tncc_Initialize(const char* filename)
```

Alternative to `libtnc_tncc_InitializeStd`. Reads the list of IMVs from the given filename (even on Windows)

7.1.2 libtnc_tncs_InitializeStd

```
TNC_Result libtnc_tncs_InitializeStd()
```

Initialize the TNCS and IMV layers. Reads the list of IMVs from the standard places according to IF-IMC. On Windows, reads from the Registry at HKLM\SOFTWARE\Trusted Computing Group\TNC\IMVs. On Unix, reads /etc/tnc_config

7.1.3 libtnc_tncs_Terminate

```
TNC_Result libtnc_tncs_Terminate()
```

Terminates the TNCS layer and unloads all IMVs.

7.1.4 libtnc_tncs_CreateConnection

```
libtnc_tncs_connection* libtnc_tncs_CreateConnection  
(void* appData)
```

Creates a new connection. The returned pointer should be considered as a pointer to an opaque structure. Returns NULL on failure. The resulting pointer should be passed to other libtnc_tncs calls. The appData may be used by the application to associate its own data with a libtnc_tncs_connection. It can be recovered with libtnc_tncs_ConnectionAppData() and is not used by the TNCS internals.

7.1.5 libtnc_tncs_ConnectionAppData

```
void* libtnc_tncs_ConnectionAppData  
(libtnc_tncs_connection* conn)
```

Recover the appData that was passed to libtnc_tncs_CreateConnection().

7.1.6 libtnc_tncs_SetRecommendationPolicy

```
int libtnc_tncs_SetRecommendationPolicy  
(libtnc_tncs_connection* conn, int policy)
```

Sets the recommendation policy to be implemented by the TNCS layer. May be one of:

- **LIBTNC_TNCS_RECOMMENDATION_POLICY_ALL**
The default. All IMVs must make the same recommendation before a final recommendation will be made by TNCS.
- **LIBTNC_TNCS_RECOMMENDATION_POLICY_ANY**
At least one IMV must make a recommendation. The least liberal recommendation will be adopted as the final recommendation.

7.1.7 libtnc_tncs_DeleteConnection

```
TNC_ConnectionID libtnc_tncs_DeleteConnection  
(libtnc_tncs_connection* conn)
```

Delete a previously established connection.

7.1.8 libtnc_tncs_BeginSession

```
TNC_Result libtnc_tncs_BeginSession
    (libtnc_tncs_connection* conn)
```

Start a new session on a previously created connection.

7.1.9 libtnc_tncs_ReceiveBatch

```
TNC_Result libtnc_tncs_ReceiveBatch
    (libtnc_tncs_connection* conn,
     const char* messageBuffer,
     size_t messageLength)
```

Passes a message batch to the TNCS. The messages within the batch will be passed to IMVs that have registered interest in those types of message. You must call libtnc_tncs_EndBatch() afterwards in order to trigger any messages to be sent to the TNCC.

7.1.10 libtnc_tncs_HaveRecommendation

```
TNC_Result libtnc_tncs_HaveRecommendation
    (libtnc_tncs_connection* conn,
     TNC_IMV_Action_Recommendation* recommendation,
     TNC_IMV_Evaluation_Result* evaluation)
```

If the IMV layer has made a recommendation, returns TNC_RESULT_SUCCESS, and (if the pointers are not NULL) sets the contents of the *recommendation* and *evaluation*.

7.2 C Bindings for calls made out by TNCS

The following function calls will be made by TNCS in response to activities by the IMVs and IF-IMV layer. The calling application is required to implement them:

7.2.1 TNC_TNCS_RequestHandshakeRetry

```
TNC_Result TNC_TNCS_RequestHandshakeRetry
    (TNC_IMVID imvID,
     TNC_ConnectionID connectionID,
     TNC_RetryReason reason)
```

Called if the IMV layer determines the need to redo the TNC handshake.

7.2.2 TNC_TNCS_SetAttribute

```
TNC_Result TNC_TNCS_SetAttribute
    (TNC_IMVID imvID,
     TNC_ConnectionID connectionID,
     TNC_AttributeID attributeID,
     TNC_UInt32 bufferLength,
     TNC_BufferReference buffer)
```

Set the value of an attribute. There is a default implementation which will be linked if the application does not define this function.

7.2.3 TNC_TNCS_GetAttribute

```
TNC_Result TNC_TNCS_GetAttribute
    (TNC_IMVID imvID,
     TNC_ConnectionID connectionID,
     TNC_AttributeID attributeID,
     TNC_UInt32 bufferSize,
     TNC_BufferReference buffer,
     TNC_UInt32 *pOutValueLength)
```

Get the value of an attribute. There is a default implementation which will be linked if the application does not define this function.

7.2.4 TNC_TNCS_SendBatch

```
TNC_Result TNC_TNCS_SendBatch
    (libtnc_tncc_connection* conn,
     const char* messageBuffer,
     size_t messageLength)
```

Called by the TNCS layer from within libtnc_tncc_EndBatch() in order to send a completed batch to the TNCC client. You must implement this to pass the batch to the TNCC, where it would be passed to libtnc_tncc_ReceiveBatch().

7.2.5 libtnc_tncc_BindFunction

```
TNC_Result libtnc_tncc_BindFunction
    (TNC_IMVID imvID,
     char *functionName,
     void **pOutfunctionPointer)
```

This function is called by TNC_TNCS_BindFunction when an IMV wishes to discover an address of a function it wishes to call. The default implementation just returns TNC_RESULT_INVALID_PARAMETER, but it may be overridden by the calling application to provide its own implementations of IMV functions or to add new ones that the IMV may call.

7.2.6 libtnc_logMessage

```
TNC_Result libtnc_logMessage
    (TNC_UInt32 severity,
     const char* format, ...)
```

The default implementation logs the message to stderr. The default implementation will be linked if the calling application does not provide an implementation.

7.3 Coding a TNCS Server

A typical TNCC client will usually initialize the TNCC and IMC layers, loading all the configured IMCs at startup:

```
libtnc_tncs_InitializeStd();
```

The server application will usually make at the beginning of a new TNCC-TNCS connection:

```
myImvConn = libtnc_tncv_CreateConnection(NULL);  
libtnc_tncv_BeginSession(myImvConn);
```

Whenever a message batch is received from the TNCS, it is passed to the TNCC layer with the following code. During a TNC handshake there may be a number of batches received. The receipt of a batch will usually trigger the transmission of a reply batch by calling `TNC_TNCC_SendBatch()`.

```
libtnc_tncc_ReceiveBatch(myImcConn,  
                        messageBuffer, messageLength);
```

During the call to `libtnc_tncc_ReceiveBatch()`, the TNCS layer may make a final recommendation about whether or not to grant access to the client. If so, a call to `libtnc_tncs_HaveRecommendation` will return true:

```
if (libtnc_tncs_HaveRecommendation  
    (myImvConn, recP, evalP) == TNC_RESULT_SUCCESS)  
{  
    // grant the type of access recommended  
    .....  
    // delete the connection  
    libtnc_tncs_DeleteConnection(myImvConn);  
}
```

8.0 OSC-IMC

This is a fully functional IMC for Windows and Unix. It works with OSC-IMV to implement a simple but complete TNC posture assessment system. OSC-IMC can answer a number of simple queries sent by OSC-IMV dueing a posture assesment, including:

- Host Operating System name and details
- Size, mode and status of a file in the file syetem
- Value and type of registry entries (Windows only)
- Status of installed RPM packages (Unix only)
- Result of an external program (Windows only)
- Display a message to the user

9.0 OSC-IMV

This is a fully functional IMV for Windows and Unix. It works with OSC-IMC to implement a simple but complete TNC posture assessment system. OSC-IMV can be

configured with a policy file, allowing the system administrator to impose arbitrarily complicated posture assessment policies for Windows and Unix clients.

9.1 Policy File

When OSC-IMC is loaded by libtnc or any other TNC compliant IF-IMV layer, it reads a file that tells it the policy to enforce on OSC-IMC equipped clients. OSC-IMC must be installed on the client beforehand, and the client must be configured with a TNC compliant supplicant.

OSC-IMV on the server then initiates a conversation with OSC-IMC on the client. IT sends queries to OSC-IMC which replies with the requested data. At the end of this process, OSC-IMV make a *recommendation* about whether and how to permit the client to connect to the network.

The Policy File describes what conditions must exist on the client before the server can issue ACCEPT, ISOLATE or NO_ACCESS recommendations.

The default Policy File is:

- /etc/osc_imv_policy.cfg (Unix)
- C:\osc_imv_policy.cfg (Windows)

The name of the policy file can be altered with the OSC_IMV_POLICY_FILE environment variable.

9.2 Policy File Syntax

The syntax of the policy file is described below.

A fundamental idea in the policy file is the idea of 'Functions'. A Function gets data from the client and allows it to be tested in Policy File in the server. A function will only get its value from the client once, when it is first required. Thereafter, if the value of that function is required again, it cached value will be used. The Policy File is reevaluated after each set of data is got from the client until either a recommendation is made by the policy file or until the client or server has no more TNC messages to send to the other side, in which case a 'No Recommendation' will be determined, which generally results in access being denied.

The file format is generally free form. White space is ignored and lines beginning with a hash '#' are also ignored as comments. The file consists of sequences of Statements:

9.3 Statements

9.3.1 recommend

```
recommend recommendation
```

Where *recommendation* can be one of

- ALLOW

- ISOLATE
- NO_ACCESS
- NO_RECOMMENDATION

9.3.2 **usermessage**

```
usermessage 'message'
```

Causes *message* to be displayed on the client computer if possible.

9.3.3 **log**

```
log severity 'message'
```

Causes message to be logged in the server. If the IF-IMV interface supports TNC_9048_LogMessage, that will be called to log the message (this may result in a logging function being called in the calling application, if so configured). Otherwise it will be printed to stderr.

Severity may be one of:

- ERR
- WARNING
- NOTICE
- INFO
- DEBUG

9.3.4 **if**

```
if (predicate) {statements}
```

If the predicate evaluates to true, then the statements enclosed in the curly brackets are executed. Statements may consist of any number of recommend, usermessage, log or if statements.

Predicate may be one or more function tests, joined by 'and' 'or' or enclosed in parentheses.

9.4 **Functions**

A function test takes the form:

```
system.substem('arg') op 'value'
```

System and subsystem are predefined names of the types of data that can be retrieved, and arg may indicate exactly which value to get. value is a literal string, and op is a comparison operator from the set:

- eq
- ==

- <
- >
- contains

The following systems are supported:

9.4.1 System

This gets information about the client's host operating system.

On Unix, the following subsystems are available. They come from the same source as `uname()` in the client machine.

- `System.name()`
Typically 'Linux' or 'SunOS' etc.
- `System.nodename()`
- `System.release()`
- `System.version()`
- `System.machine()`
- `System.lang()`
- `System.user()`

On Windows, the following subsystems are available. The data is retrieved using `GetVersionEx()`.

- `System.name()`
Evaluates to 'Windows'
- `System.majorversion()`
- `System.minorversion()`
- `System.buildnumber()`
- `System.platformid()`
- `System.csdversion()`
- `System.servicepackmajor()`
- `System.servicepackminor()`
- `System.suitemask()`
- `System.producttype()`

9.4.2 File

This gets information about a file on the client's file system. It is available for both Windows and Unix clients.

The following subsystems are available:

- `File.status('filename')`
If the file exists '0' (and `.size` and `.mode` will also be available. Otherwise the `errno` from attempting to `stat` the file.

- `File.size('filename')`
The size of the file in bytes, if the file exists.
- `File.mode('filename')`
The protection mode of the file, if the file exists.

9.4.3 Registry

This gets information from the client's registry. It is available only for Windows clients.

The following subsystems are available:

- `Registry.type('registrykey')`
An integer representing the type of the data in the key, if it exists. If the key does not exist, set to -1.
- `Registry.value('registrykey')`
The stringified value of the key. Integer keys are converted to integer strings. Strings are returned verbatim. Other types are not converted and result in the value 'UNABLETOCONVERT'.

9.4.4 Package

This gets information about RPM packages installed on the client. It is available only for Unix clients.

The following subsystems are available:

- `Package.status('packagename')`
The return value from the RPM query command. 0 means the package is installed. Other values indicate the package is not installed.
- `Package.version('packagename')`
The version number of the installed version of *packagename*.

9.4.5 Extcommand

Runs an external command on the client with configurable arguments. Available only for Windows clients.

It is not possible to control exactly which program is run on the client by this command, since that is hardwired into OSC-IMC, but you can control the arguments passed to it.

It is common to use this command with the `wzccmd.exe` program from Cloudpath networks to evaluate the status of the firewall.

The following subsystems are available:

- `Extcommand.status('extcommand args')`
Runs the external program on the client with the specified command line arguments. Evaluates to the return status if the external program.
- `Extcommand.result('extcommand args')`
Runs the external program on the client with the specified command line arguments. Evaluates to the output on stdout printed by the external program.

9.5 Examples

Some simple examples of Policy Files. They illustrate the syntax only and are not appropriate for production.

9.5.1 Example 1

```
# Always let Linux in
if (System.name() eq 'Linux')
{
  recommend ALLOW
}
# Only let windows in if the firewall is running
if (System.name() eq 'Windows')
{
  if (Extcommand.result('FIREWALL XP CHECK_ANY') == '1')
  {
    recommend ALLOW
  }
}
```

9.5.2 Example 2

```
if (System.name() eq 'Windows'
and Registry.value('SOFTWARE\Xyz\Version') < 11)
{
  recommend ISOLATE
  # A comment
  log WARNING 'they are running an old version of XYZ'
}
```

9.5.3 Example 3

```
if (File.status('/some/file') == '0'
or (File.status('/some/other/file') == '0'
and File.status('/yet/another/file') == '0'))
{
  recommend NO_ACCESS
  usermessage 'Looks like you have been hacked by a rootkit'
}
```

10.0 Sample IMC/IMV

The directory src/sample contains a skeleton IMC and IMV pair in sample_imc.c and sample_imv.c. You may want to use these as a starting point for your own IMC/IMV pair. AS they stand they do nothing useful except compile and provide placeholders for functions that you will be required to implement.

11.0 Perl bindings Interface-TNC

This subdirectory contains a Perl wrapper for the libtnc API.

To build it:

```
cd Interface-TNC
tar zxvf Interface-TNC-1.0.tar.gz
cd Interface-TNC-1.0
perl Makefile.PL
make
make install
```

12.0 Copyright and License

This software is Copyright (C) 2008 Mike McCauley. Use is subject to license conditions. The main licensing options available are GPL V2 or Commercial:

12.1 Open Source Licensing GPL V2

This is the appropriate option if you want to share the source code of your application with everyone you distribute it to, and you also want to give them the right to share who uses it. If you wish to use this software under Open Source Licensing, you must contribute all your source code to the open source community in accordance with the GPL Version 2 when your application is distributed. See <http://www.gnu.org/copyleft/gpl.html>

12.2 Commercial Licensing

This is the appropriate option if you are creating proprietary applications and you are not prepared to distribute and share the source code of your application. Contact info@open.com.au for details.