TCG Compliance_TNC IF-IMC Compliance Test Plan

Version 1.0
Revision 0.09
29 January 2008
Published

Contact:
admin@trustedcomputinggroup.org

TCG PUBLISHED

Table of Contents

1 Introduction

This section summarizes the purpose and intended audience for this document.

1.1 Purpose

The purpose of this document is to provide specific requirements for the compliance tests for IF-IMC v1.1 [2]; in particular, it defines and lists all the compliance test cases that must be passed to prove Compliance with respect to the IF-IMC v1.1 specification. This document does not contain any normative statements.

1.2 Intended Audience

The intended audience for this document includes test designers and implementers, as well as product developers and customers who need to understand the IF-IMC compliance tests. Readers should be familiar with the TNC Architecture [1], with the Compliance_TNC Compliance and Interoperability Principles specification [3] and with IF-IMC v1.1.

2 Specifications and Components

2.1 Specifications

This document is based on the IF-IMC v1.1 specification [2] and on the Compliance_TNC Compliance and Interoperability Principles document [3]. The IF-IMC v1.1 specification defines the IF-IMC interface. The Compliance_TNC Compliance and Interoperability Principles document provides an overview of the Compliance_TNC testing.

2.2 Components

There are two IF-IMC compliance tests that test the two kinds of components that interface with IF-IMC: Integrity Measurement Collectors and TNC Clients.

2.2.1 IF-IMC Compliance Test for Integrity Measurement Collectors (IMCs)

The IF-IMC Compliance test for Integrity Measurement Collectors (IMCs) tests that an IMC properly implements IF-IMC. The Test Target for this test (the component under test) is an IMC.

One difficult aspect of this test is that most IMCs require an active IMV from the same vendor in order to function properly. Therefore, a test program (a single executable binary) will be developed that loads a matching IMC and IMV pair then runs through a series of tests with the IMC, allowing the IMC to communicate with the IMV.

2.2.2 IF-IMC Compliance Test for TNC Clients (TNCCs)

The IF-IMC Compliance test for TNC Clients (TNCCs) tests that a TNCC properly implements IF-IMC. The Test Target for this test (the component under test) is a TNCC.

To test TNCCs, a test IMC will be developed that exercises the IF-IMC API and verifies that the TNCC complies with the IF-IMC specification. A matching IMV will also be developed so that the test IMC has something to talk to.

3 Specifications Requirements

The IF-IMC v1.1 specification includes many requirements and recommendations for Integrity Measurement Collectors and TNC Clients. This section lists only the requirements since the compliance tests for IF-IMC only test normative requirements (not recommendations).

This section has two subsections. The first one lists requirements upon Integrity Measurement Collectors, which are tested by the IF-IMC compliance test for IMCs. The second one lists requirements upon TNC Clients, which are tested by the IF-IMC compliance test for TNCCs.

As required by the TCG Compliance and Interoperability Guidelines, each requirement listed below has a unique name composed of the string “CTNC” (for Compliance_TNC), “IFIMC1.1” (indicating that these are requirements from IF-IMC v1.1), “IMC” or “TNCC” depending on which component the requirement applies to, a requirement number unique within the preceding prefix, and compliance classifer (“M” for MUST, “S” for SHOULD, “O” for OPTIONAL or MAY, “X” for Expressly Forbidden or MUST NOT). Usage classifiers are not included in requirement names at this time.

3.1 Requirements on IMCs

[CTNC-IFIMC1.1-IMC-REQ-1-M] Vendor-specific functions MUST have a name that begins with “`TNC_XXX_`” where `XXX` is replaced by the vendor ID of the organization that defined the extension. (IF-IMC section 2.6) Vendor-specific functions MUST have a name that begins with “TNC_XXX_” where XXX is replaced by the vendor ID of the organization that defined the extension. (IF-IMC section 3.2.4)

[CTNC-IFIMC1.1-IMC-REQ-2-M] If more than one TNC Client may be running at once on a single machine (rare, but possible) and an IMC is loaded by both TNC Clients, the IMC MUST work properly even if the TNC Clients happen to choose the same network connection ID for different connections. (IF-IMC section 2.7.2)

[CTNC-IFIMC1.1-IMC-REQ-3-M] WARNING: The message routing and delivery algorithm just described is not a one-to-one model. A single message may be received by several recipients (for example, two IMVs from a single vendor, two copies of an IMC, or nosy IMVs that monitor all messages). If several of these recipients respond, this may confuse the original sender. IMCs and IMVs MUST work properly in this environment. They MUST NOT assume that only one party will receive and/or respond to a message. (IF-IMC section 2.7.4)

[CTNC-IFIMC1.1-IMC-REQ-4-M] On platforms that don’t define a Dynamic Function Binding mechanism, all optional functions MUST be implemented, vendor-specific functions MUST NOT be implemented or used except by private convention, and provisions must be made to insure that TNCCs and IMCs that support different version numbers interact safely. (IF-IMC section 3.2.2) On platforms that don’t define a Dynamic Function Binding mechanism, all optional [IF-IMC API] functions MUST be implemented. (IF-IMC section 3.6)

[CTNC-IFIMC1.1-IMC-REQ-5-M]	An IMC or TNC Client MUST work properly if a vendor-specific function is not implemented by the other party […]. (IF-IMC section 3.2.4)
[CTNC-IFIMC1.1-IMC-REQ-6-M]	An IMC or TNC Client […] MUST ignore vendor-specific functions that it does not understand. (IF-IMC section 3.2.4)
[CTNC-IFIMC1.1-IMC-REQ-7-M]	The vendor ID is converted to ASCII numbers or the equivalent, using a decimal representation whose initial digit MUST NOT be zero (0). (IF-IMC section 3.2.4)
[CTNC-IFIMC1.1-IMC-REQ-8-M]	However, since more than one TNC Client may be running at once on a single machine (rare, but possible), any IMC DLL MUST be prepared to be loaded in multiple processes at once and to have these processes issue overlapping calls to the DLL. (IF-IMC section 3.3)
[CTNC-IFIMC1.1-IMC-REQ-9-M]	The TNCC can choose any value for the IMC ID and the IMC MUST NOT attach any significance to the value chosen. (IF-IMC section 3.4.2.1)
[CTNC-IFIMC1.1-IMC-REQ-10-M]	As described in section 2.10.3 above, it is sometimes desirable to retry an Integrity Check Handshake (when remediation is complete, for instance). Some TNCCs will not support this but all IMCs MUST do so. (IF-IMC section 3.4.2.2)
[CTNC-IFIMC1.1-IMC-REQ-11-M]	[The TNC Client] can even choose a network connection ID that was used by a previous network connection that has now been deleted and is invalid. The IMC MUST NOT attach any significance to the value chosen. (IF-IMC section 3.4.2.2)
[CTNC-IFIMC1.1-IMC-REQ-12-M]	The IMC MUST pass one of the values listed in section 3.5.5. The IMC MUST NOT use any other handshake retry reason value with this version of the IF-IMC API. (IF-IMC section 3.4.2.4)
[CTNC-IFIMC1.1-IMC-REQ-13-M]	An IMC MUST NOT send messages whose message type includes one of these reserved values. (IF-IMC section 3.4.2.5) The vendor ID TNC_VENDORID_ANY (0xffffff) and the subtype TNC_SUBTYPE_ANY (0xff) are reserved as wild cards as described in section 3.8.1. An IMC MUST NOT send messages whose message type includes one of these reserved values. (IF-IMC section 3.4.2.5)
[CTNC-IFIMC1.1-IMC-REQ-14-M]	The message type TNC_VENDORID_ANY (0xffffff) is reserved as a wild card as described in section 3.8.1. IMCs may request messages with this vendor ID to indicate that they want to receive messages whose message type includes any vendor ID. However, an IMC MUST NOT send messages whose message type includes this reserved value and a TNCC MUST NOT deliver such messages. (IF-IMC section 3.4.2.7)
[CTNC-IFIMC1.1-IMC-REQ-15-M]	The message subtype TNC_SUBTYPE_ANY (0xff) is reserved as a wild card as described in section 3.8.1. IMCs may request messages with this message subtype to indicate that they want to receive messages whose

	message subtype has any value. However, an IMC MUST NOT send messages whose message subtype includes this reserved value and a TNCC MUST NOT deliver such messages. (IF-IMC section 3.4.2.8)
[CTNC-IFIMC1.1-IMC-REQ-16-M]	IMCs and TNCCs MUST be prepared for any function to return any result code. (IF-IMC section 3.4.2.10 and 3.5.1)
[CTNC-IFIMC1.1-IMC-REQ-17-M]	The reserved value TNC_CONNECTIONID_ANY MUST NOT be used as a normal network connection ID. Instead, it may be passed to TNC_TNCC_RequestHandshakeRetry to indicate that handshake retry is requested for all current network connections. (IF-IMC section 3.5.3)
[CTNC-IFIMC1.1-IMC-REQ-18-M]	Some of the functions in the IF-IMC API are marked as mandatory below. Mandatory [IF-IMC API] functions MUST be implemented. (IF-IMC section 3.6)
[CTNC-IFIMC1.1-IMC-REQ-19-M]	An IMC or TNC Client MUST work properly if one or more optional [IF-IMC API] functions are not implemented by the other party. (IF-IMC section 3.6) [Note that there are no optional TNCC functions in IF-IMC 1.1 so no test case is needed for this requirement at this time.]
[CTNC-IFIMC1.1-IMC-REQ-20-M]	All IMCs MUST implement TNC_IMC_Initialize (IF-IMC section 3.7.1)
[CTNC-IFIMC1.1-IMC-REQ-21-M]	The IMC MUST check these [minVersion and maxVersion] to determine whether there is an API version number that it supports in this range. (IF-IMC Section 3.7.1)
[CTNC-IFIMC1.1-IMC-REQ-22-M]	If not [if the API version number specified is not supported], the IMC MUST return TNC_RESULT_NO_COMMON_VERSION. (IF-IMC Section 3.7.1)
[CTNC-IFIMC1.1-IMC-REQ-23-M]	If the state is TNC_CONNECTION_STATE_DELETE, the IMC MUST NOT pass this network connection ID to the TNC Client after this function [TNC_IMC_NotifyConnectionChange] returns (unless the TNCC later creates another network connection with the same network connection ID). (IF-IMC Section 3.7.2)
[CTNC-IFIMC1.1-IMC-REQ-24-M]	All IMCs MUST implement this function [TNC_IMC_BeginHandshake] (IF-IMC Section 3.7.3)
[CTNC-IFIMC1.1-IMC-REQ-25-M]	[TNC_IMC_ReceiveMessage is an optional function] The IMC MUST NOT ever modify the buffer contents [passed into TNC_IMC_ReceiveMessage function] (IF-IMC Section 3.7.4)
[CTNC-IFIMC1.1-IMC-REQ-26-M]	[TNC_IMC_ReceiveMessage is an optional function] The IMC MUST NOT access the buffer [passed in the TNC_IMC_ReceiveMessage function] after TNC_IMC_ReceiveMessage has returned. (IF-IMC Section 3.7.4)

[CTNC-IFIMC1.1-IMC-REQ-27-M]	[TNC_IMC_ReceiveMessage is an optional function] In the imcID, the IMC MUST pass the value provided to `TNC_IMC_Initialize`. (IF-IMC Section 3.8.1)
[CTNC-IFIMC1.1-IMC-REQ-28-M]	An IMC calls this function [TNC_TNCC_SendMessage] to give a message to the TNCC for delivery. The imcID parameter [passed to the TNC_TNCC_SendMessage function] MUST contain the value provided to TNC_IMC_Initialize. (IF-IMC Section 3.8.2)
[CTNC-IFIMC1.1-IMC-REQ-29-M]	An IMC calls this function [TNC_TNCC_SendMessage] to give a message to the TNCC for delivery. The connectionID parameter [passed to the TNC_TNCC_SendMessage function] MUST contain a valid network connection ID. (IF-IMC Section 3.8.2)
[CTNC-IFIMC1.1-IMC-REQ-30-M]	The IMC MUST NOT call this function [TNC_TNCC_SendMessage] unless it has received a call to TNC_IMC_BeginHandshake, TNC_IMC_Receive Message, or TNC_IMC_BatchEnding for this connection and the IMC has not yet returned from that function. (IF-IMC Section 3.8.2)
[CTNC-IFIMC1.1-IMC-REQ-31-M]	The IMC MUST NOT specify a message type whose vendor ID is 0xffffff or whose subtype is 0xff [when calling TNC_TNCC_SendMessage]. (IF-IMC Section 3.8.2)
[CTNC-IFIMC1.1-IMC-REQ-32-M]	An IMC calls this function [TNC_TNCC_RequestHandshakeRetry] to ask a TNCC to retry an Integrity Check Handshake. The IMC MUST pass its IMC ID as the imcID parameter, a network connection ID as the connectionID parameter, and one of the handshake retry reasons listed in section 3.5.5, as the reason parameter. (IF-IMC Section 3.8.3) The following is the complete set of permissible values for the TNC_Retry_Reason type in this version of the IF-IMC API. TNC_RETRY_REASON_IMC_REMEDIATION_COMPLETE, TNC_RETRY_REASON_IMC_SERIOUS_EVENT, TNC_RETRY_REASON_IMC_INFORMATIONAL_EVENT, TNC_RETRY_REASON_IMC_PERIODIC.
[CTNC-IFIMC1.1-IMC-REQ-33-M]	[Windows Platform Binding] Since more than one TNC Client may be running at once on a single machine (rare, but possible), any IMC DLL MUST be prepared to be loaded in multiple processes at once and to have these processes issue overlapping calls to the DLL. (IF-IMC Section 4.1.3) [UNIX/Linux Platform Binding] Since more than one TNC Client may be running at once on a single machine (rare, but possible), any IMC MUST be prepared to be loaded in multiple processes at once and to have these processes issue overlapping calls to the IMC. (IF-IMC Section 4.2.4)
[CTNC-IFIMC1.1-IMC-REQ-34-M]	[Windows Platform Binding] The Microsoft Windows DLL platform binding for the IF_IMC API defines one additional function that MUST be implemented by IMCs implementing this platform binding. (IF-IMC Section

	4.1.7) IMCs implementing the Microsoft Windows DLL platform binding MUST define this additional platform-specific function [TNC_IMC_ProvideBindFunction]. (IF-IMC Section 4.1.7.1)
[CTNC-IFIMC1.1-IMC-REQ-35-M]	[Windows Platform Binding] The IMC MUST set the imcID parameter [passed to TNC_TNCC_BindFunction function] to the IMC ID value provided to TNC_IMC_Initialize. (IF-IMC Section 4.1.8.1)
[CTNC-IFIMC1.1-IMC-REQ-36-M]	[Windows Platform Binding] The IMC MUST set the functionName parameter [passed to TNC_TNCC_BindFunction function] to a pointer to a C string identifying the function whose pointer is desired (i.e., "TNC_TNCC_SendMessage") (IF-IMC Section 4.1.8.1)
[CTNC-IFIMC1.1-IMC-REQ-37-M]	[Windows Platform Binding] The IMC MUST set the pOutFunctionPointer parameter [passed to the TNC_TNCC_BindFunction function] to a pointer to storage into which the desired function pointer will be stored. (IF-IMC Section 4.1.8.1)
[CTNC-IFIMC1.1-IMC-REQ-38-M]	[Windows Platform Binding] A well-known registry key is used by the TNCC to load IMCs. […] Additional values or keys may be present within the keys listed above. […] TNC Clients and IMCs MUST ignore unrecognized values and keys. (IF-IMC Section 4.1.9)
[CTNC-IFIMC1.1-IMC-REQ-39-M]	[UNIX/Linux Platform Binding] Both the IMC and the TNC Client MUST use POSIX threads (pthreads) for threading and synchronization to ensure compatibility. [However, since the IMC is not required to use threads or be thread-safe, it is not possible to test this requirement.] (IF-IMC Section 4.2.4)
[CTNC-IFIMC1.1-IMC-REQ-40-M]	[UNIX/Linux Platform Binding] The UNIX/Linux Dynamic Linkage platform binding for the IF-IMC API defines one additional function that MUST be implemented by IMCs implementing this platform binding. (IF-IMC Section 4.2.8) IMCs implementing the UNIX/Linux Dynamic Linkage platform binding MUST define this additional platform-specific function [TNC_IMC_ProvideBindFunction]. (IF-IMC Section 4.2.8.1)
[CTNC-IFIMC1.1-IMC-REQ-41-M]	[UNIX/Linux Platform Binding] The IMC MUST set the imcID parameter [passed to the TNC_TNCC_BindFunction] to the IMC ID value provided to TNC_IMC_Initialize. (IF-IMC Section 4.2.9.1)
[CTNC-IFIMC1.1-IMC-REQ-42-M]	[UNIX/Linux Platform Binding] The IMC MUST set the functionName parameter [passed to the TNC_TNCC_BindFunction] to a pointer to a C string identifying the function whose pointer is desired (i.e., "TNC_TNCC_SendMessage"). (IF-IMC Section 4.2.9.1)
[CTNC-IFIMC1.1-IMC-REQ-43-M]	[UNIX/Linux Platform Binding] The IMC MUST set the pOutFunctionPointer parameter [passed to the TNC_TNCC_BindFunction] to a pointer to storage into

which the desired function pointer will be stored. (IF-IMC Section 4.2.9.1)

3.2 Requirements on TNC Clients

[CTNC-IFIMC1.1-TNCC-REQ-1-M]	Vendor-specific functions MUST have a name that begins with "`TNC_XXX_`" where `XXX` is replaced by the vendor ID of the organization that defined the extension. (IF-IMC section 2.6) Vendor-specific functions MUST have a name that begins with "TNC_XXX_" where XXX is replaced by the vendor ID of the organization that defined the extension. (IF-IMC section 3.2.4)
[CTNC-IFIMC1.1-TNCC-REQ-2-M]	The TNCC MUST use the same connection ID for all IMCs when referring to a particular connection. (IF-IMC section 2.7.2)
[CTNC-IFIMC1.1-TNCC-REQ-3-M]	A zero length message is perfectly valid and MUST be properly delivered by the TNCC and TNCS just as any other IMC-IMV message would be. (IF-IMC section 2.7.3)
[CTNC-IFIMC1.1-TNCC-REQ-4-M]	On platforms that don't define a Dynamic Function Binding mechanism, all optional functions MUST be implemented, vendor-specific functions MUST NOT be implemented or used except by private convention, and provisions must be made to insure that TNCCs and IMCs that support different version numbers interact safely. (IF-IMC section 3.2.2) On platforms that don't define a Dynamic Function Binding mechanism, all optional [IF-IMC API] functions MUST be implemented. (IF-IMC section 3.6).
[CTNC-IFIMC1.1-TNCC-REQ-5-M]	An IMC or TNC Client MUST work properly if a vendor-specific function is not implemented by the other party […]. (IF-IMC section 3.2.4)
[CTNC-IFIMC1.1-TNCC-REQ-6-M]	An IMC or TNC Client […] MUST ignore vendor-specific functions that it does not understand. (IF-IMC section 3.2.4)
[CTNC-IFIMC1.1-TNCC-REQ-7-M]	The vendor ID is converted to ASCII numbers or the equivalent, using a decimal representation whose initial digit MUST NOT be zero (0). (IF-IMC section 3.2.4)
[CTNC-IFIMC1.1-TNCC-REQ-8-M]	The TNCC MUST be reentrant (able to receive and process a function call even when one is already underway). (IF-IMC section 3.3)
[CTNC-IFIMC1.1-TNCC-REQ-9-M]	IMC DLLs are not required to be reentrant. Therefore, the TNC Client MUST NOT call an IMC DLL from a callback function (like TNC_TNCC_SendMessage) [...]. (IF-IMC section 3.3)
[CTNC-IFIMC1.1-TNCC-REQ-10-M]	[For the network connection state value,] the TNCC MUST pass one of the values listed in section 3.5.3. The TNCC MUST NOT use any other network connection

	state value with this version of the IF-IMC API. (IF-IMC section 3.4.2.3) The TNC Client calls this function [TNC_IMC_NotifyConnectionChange] to inform the IMC that the state of the network connection identified by connectionID has changed to newState. Section 3.5.4[1] lists all the possible values of newState for this version. The TNCC MUST NOT use any other values with this version of IF-IMC (IF-IMC Section 3.7.2) The newState parameter (passed in to TNC_IMC_NotifyConnectionChange function) MUST contain one of the values in section 3.5.4[2]. [Section 3.5.4 Network Connection State Values: The complete set of permissible values for the TNC_Connection_State type includes: TNC_CONNECTION_STATE_CREATE, TNC_CONNECTION_STATE_HANDSHAKE, TNC_CONNECTION_STATE_ACCESS_ALLOWED, TNC_CONNECTION_STATE_ACCESS_ISOLATED, TNC_CONNECTION_STATE_ACCESS_NONE, and TNC_CONNECTION_STATE_DELETE]
[CTNC-IFIMC1.1-TNCC-REQ-11-M]	TNC Clients and TNC Servers MUST properly deliver messages with any message type (as described in section 2.7.4). (IF-IMC section 3.4.2.5) [Note that this requirement contradicts [CTNC-IFIMC1.1-TNCC-REQ-12-M] and [CTNC-IFIMC1.1-TNCC-REQ-13-M]. The TNC-WG has decided to resolve this conflict by changing this requirement to say that “TNC Clients and TNC Servers MUST properly deliver messages with any message type that does not include a wild card”. This change will be made in future versions of the IF-IMC specification.]
[CTNC-IFIMC1.1-TNCC-REQ-12-M]	The message type TNC_VENDORID_ANY (0xffffff) is reserved as a wild card as described in section 3.8.1. IMCs may request messages with this vendor ID to indicate that they want to receive messages whose message type includes any vendor ID. However, an IMC MUST NOT send messages whose message type includes this reserved value and a TNCC MUST NOT deliver such messages. (IF-IMC section 3.4.2.7)
[CTNC-IFIMC1.1-TNCC-REQ-13-M]	The message subtype TNC_SUBTYPE_ANY (0xff) is reserved as a wild card as described in section 3.8.1. IMCs may request messages with this message subtype to indicate that they want to receive messages whose message subtype has any value. However, an IMC MUST NOT send messages whose message subtype includes this reserved value and a TNCC MUST NOT deliver such messages. (IF-IMC section 3.4.2.8)

[1] IF-IMC v1.1 specification references section 3.5.3. However, section 3.5.4 lists the complete set of permissible values for TNC_Connection_State type.
[2] IF-IMC v1.1 specification references section 3.5.3. However, section 3.5.4 lists the complete set of permissible values for TNC_Connection_State type.

[CTNC-IFIMC1.1-TNCC-REQ-14-M] IMCs and TNCCs MUST be prepared for any function to return any result code. (IF-IMC section 3.4.2.10 and 3.5.1)

[CTNC-IFIMC1.1-TNCC-REQ-15-M] The reserved value TNC_CONNECTIONID_ANY MUST NOT be used as a normal network connection ID. Instead, it may be passed to TNC_TNCC_RequestHandshakeRetry to indicate that handshake retry is requested for all current network connections. (IF-IMC section 3.5.3)

[CTNC-IFIMC1.1-TNCC-REQ-16-M] Some of the functions in the IF-IMC API are marked as mandatory below. Mandatory [IF-IMC API] functions MUST be implemented. (IF-IMC section 3.6)

[CTNC-IFIMC1.1-TNCC-REQ-17-M] An IMC or TNC Client MUST work properly if one or more optional [IF-IMC API] functions are not implemented by the other party. (IF-IMC section 3.6)

[CTNC-IFIMC1.1-TNCC-REQ-18-M] The TNC Client MUST NOT call any other IF-IMC API functions for an IMC until it has successfully completed a call to TNC_IMC_Initialize() (IF-IMC Section 3.7.1)

a. [Windows Platform Binding] The TNCC MUST always call the TNC_IMC_Initialize function first. (IF-IMC Section 4.1.1)

b. [UNIX/Linux Platform Binding] The TNCC MUST always call the TNC_IMC_Initialize function first. (IF-IMC Section 4.2.1)

[CTNC-IFIMC1.1-TNCC-REQ-19-M] Once a call to this function [TNC_IMC_Initialize] completed successfully, this function MUST NOT be called again for a particular IMC-TNCC pair until a call to TNC_IMC_Terminate has completed successfully. (IF-IMC Section 3.7.1)

[CTNC-IFIMC1.1-TNCC-REQ-20-M] The TNC Client MUST set minVersion to the minimum IF-IMC API version number that it supports (IF-IMC Section 3.7.1.)

[CTNC-IFIMC1.1-TNCC-REQ-21-M] The TNC Client MUST set maxVersion to the maximum API version number that it supports. (IF-IMC Section 3.7.1)

[CTNC-IFIMC1.1-TNCC-REQ-22-M] The TNC Client also MUST set pOutActualVersion so that the IMC can use it as an output parameter to provide the actual API version number to be used. With the C binding, this would involve setting pOutActualVersion to point to a suitable storage location. (IF-IMC Section 3.7.1)

[CTNC-IFIMC1.1-TNCC-REQ-23-M] The imcID parameter (passed in to TNC_IMC_NotifyConnectionChange function) MUST contain the IMC ID value provided to TNC_IMC_Initialize. (IF-IMC Section 3.7.2)

[CTNC-IFIMC1.1-TNCC-REQ-24-M] The connectionID parameter (passed in to TNC_IMC_NotifyConnectionChange function) MUST contain a valid network connection ID. (IF-IMC Section 3.7.2)

[CTNC-IFIMC1.1-TNCC-REQ-25-M]	The imcID parameter [passed in to the TNC_IMC_BeginHandshake function] MUST contain the IMC ID value provided to TNC_IMC_Initialize. (IF-IMC Section 3.7.3)
[CTNC-IFIMC1.1-TNCC-REQ-26-M]	The connectionID parameter [passed in to the TNC_IMC_BeginHandshake function] MUST contain a valid network connection ID. (IF-IMC Section 3.7.3)
[CTNC-IFIMC1.1-TNCC-REQ-27-M]	[TNC_IMC_ReceiveMessage is an optional function] The imcID parameter [passed into TNC_IMC_ReceiveMessage function] MUST contain the IMC ID value provided to TNC_IMC_Initialize. (IF-IMC Section 3.7.4)
[CTNC-IFIMC1.1-TNCC-REQ-28-M]	[TNC_IMC_ReceiveMessage is an optional function] The connectionID parameter [passed into the TNC_IMC_ReceiveMessage function] MUST contain a valid network connection ID. (IF-IMC Section 3.7.4)
[CTNC-IFIMC1.1-TNCC-REQ-29-M]	[TNC_IMC_ReceiveMessage is an optional function] The message parameter [passed into TNC_IMC_ReceiveMessage function] MUST contain a reference to a buffer containing the message being delivered to the IMC. (IF-IMC Section 3.7.4)
[CTNC-IFIMC1.1-TNCC-REQ-30-M]	[TNC_IMC_ReceiveMessage is an optional function] The messageLength parameter [passed into TNC_IMC_ReceiveMessage function] MUST contain the number of octets in the message. (IF-IMC Section 3.7.4)
[CTNC-IFIMC1.1-TNCC-REQ-31-M]	[TNC_IMC_ReceiveMessage is an optional function] The messageType parameter [passed into TNC_IMC_ReceiveMessage function] MUST contain the type of the message. It MUST match one of the TNC_MessageType values previously supplied by the IMC to the TNCC in the IMC's most recent call to TNC_TNCC_ReportMessageTypes. (IF-IMC Section 3.7.4)
[CTNC-IFIMC1.1-TNCC-REQ-32-M]	[The TNC_IMC_ReceiveMessage is an optional function] The TNC Client calls this function [TNC_IMC_ReceiveMessage] to deliver a message to the IMC. The message is contained in the buffer referenced by message and contains the number of octets (bytes) indicated by messageLength. The type of message is indicated by messageType. The message MUST be from an IMV (or a TNCS or other party acting as an IMV). (IF-IMC Section 3.7.4) [Note that compliance with this requirement is hard to test.]
[CTNC-IFIMC1.1-TNCC-REQ-33-M]	[TNC_IMC_BatchEnding is an optional function.] The imcID parameter [passed to the TNC_IMC_BatchEnding function] MUST contain the IMC ID value provided to TNC_IMC_Initialize. (IF-IMC Section 3.7.5)
[CTNC-IFIMC1.1-TNCC-REQ-34-M]	[TNC_IMC_BatchEnding is an optional function.] The connectionID parameter [passed to the TNC_IMC_BatchEnding function] MUST contain a valid network connection ID. (IF-IMC Section 3.7.5)

[CTNC-IFIMC1.1-TNCC-REQ-35-M]	[TNC_IMC_Terminate is an optional function.] The TNC Client calls this function to close down the IMC when all work is complete or the IMC reports TNC_RESULT_FATAL. Once a call to TNC_IMC_Terminate is made, the TNC Client MUST NOT call the IMC except to call TNC_IMC_Initialize (which may not succeed if the IMC cannot reinitialize itself). (IF-IMC Section 3.7.6)
[CTNC-IFIMC1.1-TNCC-REQ-36-M]	[TNC_IMC_Terminate is an optional function.] The imcID parameter [passed to the TNC_IMC_Terminate function] MUST contain the IMC ID value provided to TNC_IMC_Initialize. (IF-IMC Section 3.7.6)
[CTNC-IFIMC1.1-TNCC-REQ-37-M]	An IMC calls this function [TNC_TNCC_ReportMessageTypes] to inform a TNCC about the set of message types the IMC is able to receive. All TNC Clients MUST implement this function (IF-IMC Section 3.8.1)
[CTNC-IFIMC1.1-TNCC-REQ-38-M]	The TNC Client MUST NOT ever modify the list of message types [passed to TNC_TNCC_ReportMessageTypes]. (IF-IMC Section 3.8.1)
[CTNC-IFIMC1.1-TNCC-REQ-39-M]	The TNC Client MUST NOT access [the list of message types passed to TNC_TNCC_ReportMessageTypes] after TNC_TNCC_ReportMessageTypes has returned. (IF-IMC Section 3.8.1)
[CTNC-IFIMC1.1-TNCC-REQ-40-M]	TNC Clients MUST support any message type [for the TNC_TNCC_ReportMessageTypes function]. (IF-IMC Section 3.8.1)
[CTNC-IFIMC1.1-TNCC-REQ-41-M]	An IMC calls this function [TNC_TNCC_ReportMessageTypes] to inform a TNCC about the set of message types the IMC is able to receive. Note that although all TNC Clients must implement this function, some IMCs may never call it if they don't support receiving any message types. This is acceptable. In such a case, the TNC Client MUST NOT deliver any messages to the IMC. (IF-IMC Section 3.8.1)
[CTNC-IFIMC1.1-TNCC-REQ-42-M]	All TNC Clients MUST implement this function [TNC_TNCC_SendMessage] (IF-IMC Section 3.8.2)
[CTNC-IFIMC1.1-TNCC-REQ-43-M]	The TNC Client MUST NOT ever modify the buffer contents [passed to TNC_TNCC_SendMessage] (IF-IMC Section 3.8.2)
[CTNC-IFIMC1.1-TNCC-REQ-44-M]	The TNC Client MUST NOT access the buffer [passed to TNC_TNCC_SendMessage] after TNC_TNCC_SendMessage has returned (IF-IMC Section 3.8.2)
[CTNC-IFIMC1.1-TNCC-REQ-45-M]	The TNC Client MUST support any message type [for the function TNC_TNCC_SendMessage]. (IF-IMC Section 3.8.2)
[CTNC-IFIMC1.1-TNCC-REQ-46-M]	[Windows Platform Binding] IMC DLLs are not required to be thread-safe. Therefore, the TNC Client MUST NOT

call an IMC DLL from one thread when another TNC Client thread is in the middle of a call to the same IMC DLL. (IF-IMC Section 4.1.3)

a. [UNIX/Linux Platform Binding] IMC executable files are not required to be thread-safe. Therefore, the TNC Client MUST NOT call an IMC from one thread when another TNC Client thread is in the middle of a call to the same IMC. (IF-IMC Section 4.2.4)

[CTNC-IFIMC1.1-TNCC-REQ-47-M] [Windows Platform Binding] The IMC DLL MAY create threads. The TNC Client MUST be thread-safe. (IF-IMC Section 4.1.3)

a. [UNIX/Linux Platform Binding] The IMC MAY create threads. The TNC Client MUST be thread-safe. (IF-IMC Section 4.2.4)

[CTNC-IFIMC1.1-TNCC-REQ-48-M] [Windows Platform Binding] IMCs implementing the Microsoft Windows DLL platform binding MUST define this additional platform-specific function [TNC_IMC_ProvideBindFunction]. The TNC Client MUST call this function immediately after calling TNC_IMC_Initialize to provide a pointer to the TNCC bind function. (IF-IMC Section 4.1.7.1)

[CTNC-IFIMC1.1-TNCC-REQ-49-M] [Windows Platform Binding] The imcID parameter [passed to the TNC_IMC_ProvideBindFunction function] MUST contain the value provided to TNC_IMC_Initialize. (IF-IMC Section 4.1.7.1)

[CTNC-IFIMC1.1-TNCC-REQ-50-M] [Windows Platform Binding] The bindFunction parameter [passed to the TNC_IMC_ProvideBindFunction function] MUST contain a pointer to the TNCC bind function. (IF-IMC Section 4.1.7.1)

[CTNC-IFIMC1.1-TNCC-REQ-51-M] [Windows Platform Binding] The Microsoft Windows DLL platform binding for the IF-IMC API defines one additional function that MUST be implemented by TNC Clients implementing this platform binding. (IF-IMC Section 4.1.8)

a. TNC Clients implementing the Microsoft Windows DLL platform binding MUST define this additional platform-specific function [TNC_TNCC_BindFunction]. (IF-IMC Section 4.1.8.1)

[CTNC-IFIMC1.1-TNCC-REQ-52-M] [Windows Platform Binding] [The IMC MUST set the pOutFunctionPointer parameter [passed to the TNC_TNCC_BindFunction function] to a pointer to storage into which the desired function pointer will be stored.] If the TNCC does not define the requested function, NULL MUST be stored at pOutFunctionPointer. Otherwise, a pointer to the requested function MUST be stored at pOutFunctionPointer. (IF-IMC Section 4.1.8.1)

[CTNC-IFIMC1.1-TNCC-REQ-53-M] [Windows Platform Binding] Once an IMC obtains a pointer to a particular function [through TNC_TNCC_BindFunction], the TNCC MUST always return the same function pointer value to that IMC for that function name. (IF-IMC Section 4.1.8.1)

[CTNC-IFIMC1.1-TNCC-REQ-54-M] [Windows Platform Binding] A well-known registry key is used by the TNCC to load IMCs. […] Additional values or keys may be present within the keys listed above. […] TNC Clients and IMCs MUST ignore unrecognized values and keys. (IF-IMC Section 4.1.9)

[CTNC-IFIMC1.1-TNCC-REQ-55-M] [UNIX/Linux Platform Binding] Both the IMC and the TNC Client MUST use POSIX threads (pthreads) for threading and synchronization to ensure compatibility. (IF-IMC Section 4.2.4)

[CTNC-IFIMC1.1-TNCC-REQ-56-M] [UNIX/Linux Platform Binding] The TNC Client MUST call the [TNC_IMC_ProvideBindFunction] function immediately after calling TNC_IMC_Initialize to provide a pointer to the TNCC bind function. (IF-IMC Section 4.2.8.1)

[CTNC-IFIMC1.1-TNCC-REQ-57-M] [UNIX/Linux Platform Binding] The imcID parameter [passed to the TNC_IMC_ProvideBindFunction function] MUST contain the value provided to TNC_IMC_Initialize. (IF-IMC Section 4.2.8.1)

[CTNC-IFIMC1.1-TNCC-REQ-58-M] [UNIX/Linux Platform Binding] The bindFunction parameter [passed to the TNC_IMC_ProvideBindFunction function] MUST contain a pointer to the TNCC bind function. (IF-IMC Section 4.2.8.1)

[CTNC-IFIMC1.1-TNCC-REQ-59-M] [UNIX/Linux Platform Binding] The UNIX/Linux Dynamic Linkage platform binding for the IF-IMC API defines one additional function that MUST be implemented by TNC Clients implementing this platform binding. (IF-IMC Section 4.2.9)

a. [UNIX/Linux Platform Binding] TNC Clients implementing the UNIX/Linux Dynamic Linkage platform binding MUST define this additional platform-specific function [TNC_TNCC_BindFunction]. (IF-IMC Section 4.2.9.1)

[CTNC-IFIMC1.1-TNCC-REQ-60-M] [UNIX/Linux Platform Binding] An IMC can use this function [TNC_TNCC_BindFunction] to obtain pointers to other TNCC functions. The IMC MUST set the pOutFunctionPointer parameter to a pointer to storage into which the desired function pointer will be stored. If the TNCC does not define the requested function, NULL MUST be stored at pOutFunctionPointer. Otherwise, a pointer to the requested function MUST be stored at pOutFunctionPointer.

3.3 Other Requirements

Requirements listed in this section are requirements for neither IMC nor TNCC. They are listed here for completeness. However, they are out of scope and we will not provide test cases.

[CTNC-IFIMC1.1-OTHER-1-M] [UNIX/Linux Platform Binding] A line that begins with “IMC” (U+0049, U+004D, U+0043, U+0020) specifies an IMC that may be loaded. The next character MUST be U+0022 (QUOTATION MARK). This MUST be followed by a human-readable IMC name

(potentially zero length) and another U+0022 character (QUOTATION MARK). Of course, the IMC name cannot contain a U+0022 (QUOTATION MARK). But it can contain spaces or other characters.

After the U+0022 that terminates the human-readable name MUST come a space (U+0020) and then the full path of the IMC executable file (up to but not including the U+000A that terminates the line).

The path to the IMC executable file MUST NOT be a partial path. (IF-IMC Section 4.2.3)

This requirement is Installer's requirement, which is out of scope for this document. No test case will be provided for this requirement.

4 Test Cases

This section lists a test case for each requirement in the preceding section.

4.1 Test Cases for IF-IMC Compliance Test for IMCs

There are several asynchronous actions that an IMC may initiate and which the test program must be capable of handling during all normative test cases. For example, an IMC may attempt a handshake retry at anytime based on conditions outside the test program's control. The test program must handle IMC asynchronous actions appropriately and ensure the IMC is generating correct data/messages/etc. for the given action.

The following is the list of asynchronous actions and validation requirements for the test program:

[CTNC-IFIMC1.1-IMC-AA-1] Once loaded and initialized, an IMC may initiate a handshake retry at anytime and the test program must have code that verifies that the IMC under test only uses valid imcID, connection ID, and handshake retry reason values and no others. This entry covers the following IMC requirements: [CTNC-IFIMC1.1-IMC-REQ-12-M] and [CTNC-IFIMC1.1-IMC-REQ-32-M].

[CTNC-IFIMC1.1-IMC-AA-2] Once loaded and initialized, an IMC can send messages at anytime and the test program must have code that detects if the IMC under test ever sends messages with reserved message type values where NOT allowed. This entry covers the following IMC requirements: [CTNC-IFIMC1.1-IMC-REQ-13-M], [CTNC-IFIMC1.1-IMC-REQ-14-M], [CTNC-IFIMC1.1-IMC-REQ-15-M] and [CTNC-IFIMC1.1-IMC-REQ-31-M].

[CTNC-IFIMC1.1-IMC-AA-3] Once loaded and initialized, an IMC uses a network connection ID when communicating with the TNCC. The test program that loads IMCs will contain code to detect if the IMC under test ever uses the reserved value for the network connection ID, other than in the instance a handshake retry is being requested for all current network connections. This entry covers the following IMC requirements: [CTNC-IFIMC1.1-IMC-REQ-17-M] and [CTNC-IFIMC1.1-IMC-REQ-29-M].

[CTNC-IFIMC1.1-IMC-AA-4] The receive message function passes content to an IMC via receive message buffer. The test program that loads IMCs will contain code to verify that the contents of the receive message buffer are unmodified on return of the receive message function. The test program will also include code to detect if the IMC under test ever attempts to access the contents of the receive message buffer after the IMC returns from the receive message function. This test case covers the following IMC requirements: [CTNC-IFIMC1.1-IMC-REQ-25-M] and [CTNC-IFIMC1.1-IMC-REQ-26-M].

The following is the set of normative test cases the test program must support, unless otherwise noted.

[CTNC-IFIMC1.1-IMC-TC-1] The test program that loads the IMC under test will iterate through all the functions defined by the IMC and ensure that each of these is either an IMC function defined in the IF-IMC 1.1 specification or has a name that begins with "`TNC_XXX_`" where

	XXX is a valid vendor ID (composed of ASCII numbers using a decimal representation whose initial digit is not zero). This test case is for the following IMC requirements: [CTNC-IFIMC1.1-IMC-REQ-1-M] and [CTNC-IFIMC1.1-IMC-REQ-7-M].
[CTNC-IFIMC1.1-IMC-TC-2]	Several instances of the test program that loads IMCs will be created on the same machine. These several instances will load the IMC under test and verify that it works properly when the test program instances use the same connection ID for different connections. This test case is for the following IMC requirement:[CTNC-IFIMC1.1-IMC-REQ-2-M].
[CTNC-IFIMC1.1-IMC-TC-3]	The test program that loads the IMV will load two copies of the IMV so that two copies of each IMV message will be sent to the IMC under test. The IMC may create a log entry noting these duplicate messages, ignore them, display an error, or take any other action allowed under the specification. But the IMC must follow the IF-IMC specification and must not crash. This test case is for the following IMC requirement:[CTNC-IFIMC1.1-IMC-REQ-3-M].
[CTNC-IFIMC1.1-IMC-TC-4]	The test program that loads the IMC will implement no vendor-specific functions. The IMC must follow the IF-IMC specification and must not crash. This test case is for the following IMC requirement:[CTNC-IFIMC1.1-IMC-REQ-5-M].
[CTNC-IFIMC1.1-IMC-TC-5]	The test program that loads the IMC will implement some extra vendor-specific functions that the IMC does not understand. The IMC must follow the IF-IMC specification and must not crash. This test case is for the following IMC requirement:[CTNC-IFIMC1.1-IMC-REQ-6-M].
[CTNC-IFIMC1.1-IMC-TC-6]	Several instances of the test program that loads IMCs will be created on the same machine. These several instances will load the IMC under test and verify that it works properly when the test programs issue overlapping calls to the IMC. This test case is for the following IMC requirement:[CTNC-IFIMC1.1-IMC-REQ-8-M].
[CTNC-IFIMC1.1-IMC-TC-7]	The test program that loads IMCs will try loading the IMC several times with different IMC ID values (including edge cases like 0, 1, and the maximum TNC_UInt32 value) and verify that the IMC functions properly with all of these values. This test case is for the following IMC requirement: [CTNC-IFIMC1.1-IMC-9-M].
[CTNC-IFIMC1.1-IMC-TC-8]	The test program that loads IMCs will run a handshake retry and verify that the IMC under test functions properly during this handshake retry (in particular, that it sends messages during the initial round of the handshake retry if it sent them in the initial round of the initial handshake and that it does not crash or hang). This test case is for the following IMC requirement:[CTNC-IFIMC1.1-IMC-REQ-10-M].
[CTNC-IFIMC1.1-IMC-TC-9]	The test program that loads IMCs will run a TNC handshake with a particular connection ID. When this handshake is finished, it will delete the connection ID and then reuse the same ID for a subsequent handshake. It will verify that the IMC under test functions properly during the second handshake (in particular, that it sends messages during the initial round of the second handshake if it sent them in the initial round of the initial

handshake and that it does not crash or hang). This test case is for the following IMC requirement:[CTNC-IFIMC1.1-IMC-REQ-11-M].

[CTNC-IFIMC1.1-IMC-TC-10] The test program that loads IMCs will return the values of 0x0, 0xFFFFFFFF, and 0xA as result codes during individual calls for each of the three TNCC functions used by the IMC, as noted below.

TNC_TNCC_ReportMessageTypes()
TNC_TNCC_SendMessage()
TNC_TNCC_RequestHandshakeRetry()

With the Windows DLL binding and the UNIX/Linux binding, TNC_TNCC_BindFunction() will also behave in this manner.

This test case is for the following IMC requirement: [CTNC-IFIMC1.1-IMC-REQ-16-M].

[CTNC-IFIMC1.1-IMC-TC-11] The test program that loads IMCs will iterate through all IF-IMC API functions that are mandatory for the IMC to implement (TNC_IMC_Initialize and TNC_IMC_BeginHandshake) and verify that the IMC under test implements these functions. This test case is for the following IMC requirements: [CTNC-IFIMC1.1-IMC-REQ-18-M], [CTNC-IFIMC1.1-IMC-REQ-20-M], [CTNC-IFIMC1.1-IMC-REQ-24-M].

[CTNC-IFIMC1.1-IMC-TC-12] The test program that loads IMCs will iteratively call the IMC's initialization function with the following (minimum, maximum) version value pairings: (1,1), (1,2), (2,2), and (1, MAX_UINT32). The test program will verify that the IMC under test always returns a valid value, consisting either of the API version it does support or the no common version result code. This test case is for the following IMC requirement: [CTNC-IFIMC1.1-IMC-REQ-21-M] and [CTNC-IFIMC1.1-IMC-REQ-22-M].

[CTNC-IFIMC1.1-IMC-TC-13] The test program that loads IMCs will contain code to detect if the IMC under test ever uses network connection ID values that are in a delete state. This test case is for the following IMC requirement: [CTNC-IFIMC1.1-IMC-REQ-23-M].

[CTNC-IFIMC1.1-IMC-TC-14] The test case for [CTNC-IFIMC1.1-IMC-20-M] will be expanded by having the code in the test program detect if the IMC under test ever attempts to use a different imcID value, with the send message function, different from what was given in the initialization function. This test case is for the following IMC requirement:[CTNC-IFIMC1.1-IMC-REQ-27-M].

[CTNC-IFIMC1.1-IMC-TC-15] The test program that loads IMCs will contain code to detect if the IMC under test ever uses an invalid connection ID value with the send message function. This test case is for the following IMC requirement: [CTNC-IFIMC1.1-IMC-REQ-29-M].

[CTNC-IFIMC1.1-IMC-TC-16] The test program that loads IMCs will contain code to detect if the IMC under test ever attempts to call the send message function for a connection for which the IMC is not servicing a begin handshake, receive message, or batch ending function call and has yet to return. This test case is for the following IMC requirement:[CTNC-IFIMC1.1-IMC-REQ-30-M].

[CTNC-IFIMC1.1-IMC-TC-17] Several instances of the test program for Windows that loads IMCs will be created on the same machine. These several instances will load the IMC under test, issue overlapping calls to the IMC and verify proper operation. This test case is for the following IMC requirement:[CTNC-IFIMC1.1-IMC-REQ-33-M].

[CTNC-IFIMC1.1-IMC-TC-18] The test program(s) for UNIX/Linux or Windows that loads IMCs will call the IMC's TNC_IMC_ProvideBindFunction() to validate that it has been implemented. This test case is for the following IMC requirements: [CTNC-IFIMC1.1-IMC-REQ-34-M].

[CTNC-IFIMC1.1-IMC-TC-19] The test program for Windows that loads IMCs will have code to verify that the IMC under test uses the same imcID value in the TNCC platform bind function call as what was originally given to the IMC in the initialization function. This test case is for the following IMC requirement:[CTNC-IFIMC1.1-IMC-REQ-35-M].

[CTNC-IFIMC1.1-IMC-TC-20] The test program(s) for Windows or UNIX/Linux that loads IMCs will have code to verify that the IMC under test always sends a valid function name string that conforms to the "TNC_TNCC_" or "TNC_XXX_" format (where XXX is a vendor ID) when the IMC calls the TNCC platform bind function. This test case is for the following IMC requirements:[CTNC-IFIMC1.1-IMC-REQ-36-M].

[CTNC-IFIMC1.1-IMC-TC-21] The test program(s) for Windows or UNIX/Linux that loads IMCs will have code to verify that an IMC always passes function pointer storage when the IMC calls the TNCC platform bind function. Verification of function pointer storage can be done by verifying that function pointer is not NULL and then attempting to store a function pointer at the location pointed to. This test case is for the following IMC requirement: [CTNC-IFIMC1.1-IMC-REQ-37-M].

[CTNC-IFIMC1.1-IMC-TC-22] The test program for Windows that loads IMCs will populate the well known registry location with optional values and verify that an IMC ignores unknown optional values correctly and loads without hanging or crashing. This test case is for the following IMC requirement :[CTNC-IFIMC1.1-IMC-REQ-38-M].

[CTNC-IFIMC1.1-IMC-TC-23] If the UNIX/Linux IMC under test is multi-threaded, the operator will run 'ldd' against the IMC's .so file. The output from 'ldd' will be searched for the inclusion of the string "pthread", which is indicative of compliancy. This test case is for the following IMC requirement: [CTNC-IFIMC1.1-IMC-REQ-39-M].

The following test is case is optional for the test program to support, but highly recommended.

[CTNC-IFIMC1.1-IMC-TC-24] *Optional*: Operator will initiate a handshake retry on the IMC under test and the test program will verify that the IMC only uses valid handshake retry reason values and no others. This test case is for the following IMC requirements: [CTNC-IFIMC1.1-IMC-REQ-12-M] and [CTNC-IFIMC1.1-IMC-REQ-31-M].

Notes:

- There is no test case for requirement [CTNC-IFIMC1.1-IMC-REQ-19-M] as there are no optional to implement IF-IMC API functions currently defined in TNC standards.

- No test cases are needed for requirement [CTNC-IFIMC1.1-IMC-REQ-4-M] because all platform bindings defined in the IF-IMC 1.1 specification include Dynamic Function Binding..

4.2 Test Cases for IF-IMC Compliance Test for TNCCs

There are several asynchronous actions that an TNCC may initiate and which the test program must be capable of handling during all normative test cases. The test program must handle TNCC asynchronous actions appropriately and ensure the TNCC is generating correct data/messages/etc. for the given action.

The following is the list of asynchronous actions and validation requirements for the test program:

[CTNC_IFIMC1.1-TNCC-AA-1] The test IMC will contain code to detect if the TNC Client ever calls the test IMC from a callback function. This entry is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-9-M].

[CTNC_IFIMC1.1-TNCC-AA-2] Once loaded and initialized, an IMC uses a network connection ID when communicating with the TNCC. The test IMC will contain code to detect if the TNCC under test ever uses an invalid network connection ID (such as TNC_CONNECTION_ANY or connection ID whose state is deleted). The test IMC verifies that the imcID parameter passed by the TNCC is the IMC ID value provided to TNC_IMC_Initialize. This entry covers the following TNCC requirements:[CTNC-IFIMC1.1-TNCC-REQ-15-M], [CTNC-IFIMC1.1-TNCC-REQ-24-M], [CTNC-IFIMC1.1-TNCC-REQ-25-M], [CTNC-IFIMC1.1-TNCC-REQ-26-M], [CTNC-IFIMC1.1-TNCC-REQ-27-M], [CTNC-IFIMC1.1-TNCC-REQ-28-M], [CTNC-IFIMC1.1-TNCC-REQ-33-M], [CTNC-IFIMC1.1-TNCC-REQ-34-M], [CTNC-IFIMC1.1-TNCC-REQ-36-M], [CTNC-IFIMC1.1-TNCC-REQ-49-M] and [CTNC-IFIMC1.1-TNCC-REQ-57-M].

[CTNC_IFIMC1.1-TNCC-AA-3] The test IMC will have code to detect if the TNCC under test ever calls any IF-IMC API function before successfully completing a call to TNC_IMC_Initialize. This entry is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-18-M].

[CTNC_IFIMC1.1-TNCC-AA-4] The test IMC will have code to detect if the TNCC under test ever calls TNC_IMC_Initialize again after having called it successfully before, unless TNC_IMC_Terminate() has completed successfully. This entry is for the following TNCC requirement:[CTNC-IFIMC1.1-TNCC-REQ-19-M].

[CTNC_IFIMC1.1-TNCC-AA-5] The test IMC will have to detect if the TNCC under test ever advertises an invalid or incorrect minVersion and maxVersion argument with TNC_IMC_Initialize. This entry is for the following TNCC requirements: [CTNC-IFIMC1.1-TNCC-REQ-20-M]and [CTNC-IFIMC1.1-TNCC-REQ-21-M].

[CTNC_IFIMC1.1-TNCC-AA-6] The test IMC will have code to verify that a TNCC always passes valid function pointer storage in pOutActualVersion when the TNCC calls the IMC initialization function TNC_IMC_Initialize. Verification of function pointer storage can be done by verifying that function pointer is not NULL. This entry is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-22-M]

[CTNC_IFIMC1.1-TNCC-AA-7] The test IMC will have code to verify that a TNCC always when calling the TNC_IMC_NotifyConnection function:

(a) Uses a valid newState parameter value , namely one of the values listed in section 3.5.4.

This entry is for the following TNCC requirements: [CTNC-IFIMC1.1-TNCC-REQ-10-M] [CTNC-IFIMC1.1-TNCC-REQ-23-M].

[CTNC_IFIMC1.1-TNCC-AA-8] The test IMC will have code to verify that always when a TNCC is calling the [optional] TNC_IMC_ReceiveMessage() function:

(a) message parameter contains a valid reference (e.g. memory pointer) to the buffer containing the message being delivered to the test IMC. If messageLength parameter is not zero, verification of message buffer can be done by verifying that message pointer is not NULL and then attempting to read messageLength octets from the location pointed to.

(b) messageLength parameters contains the number of octets in the message. If messageLength parameter is not zero, verification of this parameter can be done by reading messageLength octets from location pointed by message buffere reference.

(c) messageType parameter contains the type of message and matches one of the TNC_MessageType values previously supplied by the IMC to the TNCC in the IMC's most recent call to TNC_TNCC_ReportMessageTypes.

This entry is for the following TNCC requirements: [CTNC-IFIMC1.1-TNCC-REQ-29-M], [CTNC-IFIMC1.1-TNCC-REQ-30-M] and [CTNC-IFIMC1.1-TNCC-REQ-31-M]

[CTNC_IFIMC1.1-TNCC-AA-9] The test IMC will have code to verify that a TNCC never calls any IMC function other than TNC_IMC_Initialize after calling the [optional] TNC_IMC_Terminate function. This entry is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-35-M].

[CTNC_IFIMC1.1-TNCC-AA-10] The test IMC will have the capability to detect if the TNCC under test ever attempts to access or modify the message type list after TNC_TNCC_ReportMessageTypes has returned. One method for performing the validation is to use protected memory. This entry is for the following TNCC requirement:[CTNC-IFIMC1.1-TNCC-REQ-38-M] and [CTNC-IFIMC1.1-TNCC-REQ-39-M].

[CTNC_IFIMC1.1-TNCC-AA-11] The test IMC will have the capability to detect if the TNCC under test ever attempts to access or modify the buffer after TNC_TNCC_SendMessage has returned. One method for performing this validation is to use protected memory. This entry is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-43-M] and [CTNC-IFIMC1.1-TNCC-REQ-44-M].

[CTNC_IFIMC1.1-TNCC-AA-12] The test IMC program (either the Windows or Unix/Linux Platform Binding) will have code to detect if the TNCC under test attempts to call the test IMC from one thread when another TNCC client thread is in the middle of a call to the test IMC. This entry is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-46-M].

[CTNC_IFIMC1.1-TNCC-AA-13] [Windows or Unix/Linux Platform Binding] The test IMC program will have code to validate that the TNCC under test immediately calls TNC_IMC_ProvideBindFunction after TNC_IMC_Initialize. This entry is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-48-M] and [CTNC-IFIMC1.1-TNCC-REQ-56-M].

[CTNC_IFIMC1.1-TNCC-AA-14] [Windows or Unix/Linux Platform Binding] The test IMC will have code to verify that a TNCC always when calling the TNC_IMC_ProvideBindFunction function the bindFunction parameter contains a pointer to the TNCC bind function. Verification of this pointer can be performed by checking for a non-NULL value.

This entry is for the following TNCC requirements: [CTNC-IFIMC1.1-TNCC-REQ-50-M] and [CTNC-IFIMC1.1-TNCC-REQ-58-M].

The following is the set of normative test cases the test program must support.

[CTNC-IFIMC1.1-TNCC-TC-1] The test IMC will iterate through all the functions defined by the TNCC and ensure that each of these is either a TNCC function defined in the IF-IMC 1.1 specification or has a name that begins with "`TNC_XXX_`" where `XXX` is a valid vendor ID. For the latter case, the test IMC will verify that the vendor ID is composed of ASCII numbers using a decimal representation whose initial digit is not zero. This test case is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-1-M] and [CTNC-IFIMC1.1-TNCC-REQ-7-M].

[CTNC-IFIMC1.1-TNCC-TC-2] Several test IMCs will be created. When a connection starts, these IMCs will check that they all get the same connection ID for that connection. This test case is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-2-M].

[CTNC-IFIMC1.1-TNCC-TC-3] The test IMC will send a zero length message and the test IMV will verify that it is delivered properly. This test case is for the following TNCC requirement:[CTNC-IFIMC1.1-TNCC-REQ-3-M].

[CTNC-IFIMC1.1-TNCC-TC-4] The test IMC will implement no vendor-specific functions. The TNCC must follow the IF-IMC specification and must not crash. This test case is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-5-M].

[CTNC-IFIMC1.1-TNCC-TC-5] The test IMC will implement some extra vendor-specific functions that the TNCC does not understand. The TNCC must follow the IF-IMC specification and must not crash. This test case is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-6-M].

[CTNC-IFIMC1.1-TNCC-TC-6] The test IMC will wait until the TNCC calls TNC_IMC_BatchEnding. While that call is in progress, the test IMC will call TNC_TNCC_SendMessage. This test case is for the following TNCC requirement:[CTNC-IFIMC1.1-TNCC-REQ-8-M]

[CTNC-IFIMC1.1-TNCC-TC-7] The test IMC sends messages whose message types include boundary message types (0x00000000, 0x000000FE 0x00000100, 0x000001FE, 0xFFFFFE00, 0xFFFFFFFE). The

test IMV verifies that all messages are delivered. The test IMV sends messages whose types include boundary message types (0x00000000, 0x000000FE, 0x00000100, 0x000001FE, 0xFFFFFE00, 0xFFFFFEFE). The test IMC verifies that all messages are delivered. This test case is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-11-M].

[CTNC-IFIMC1.1-TNCC-TC-8] The test case for the TNCS requirement [CTNC-IFIMC1.1-TNCC-REQ-11-M] will be expanded. The test IMC sends two messages including one message whose message type includes the reserved TNC_VENDORID_ANY (0xFFFFFF). The test IMV verifies that only one message whose message type does not include the reserved TNC_VENDORID_ANY (0xFFFFFF) is received. This test case is expanded so that the test IMC sends two messages including one message whose message type includes the reserved TNC_SUBTYPE_ANY (0xFF). The test IMV verifies that only one message whose message type does not include the reserved TNC_SUBTYPE_ANY (0xFF) is received. This test case is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-12-M] and [CTNC-IFIMC1.1-TNCC-REQ-13-M].

[CTNC-IFIMC1.1-TNCC-TC-9] The test IMC will return the values of 0x0, 0xFFFFFFFF, and 0xA as result codes during individual calls for each of the six IMC functions used by the TNCC, as noted below.

TNC_IMC_Initialize()
TNC_IMC_BeginHandshake()
TNC_IMC_NotifyConnectionChange()
TNC_IMC_ReceiveMessage()
TNC_IMC_BatchEnding()
TNC_IMC_Terminate()

The TNCS under test must operate normally without hanging or crashing. This test case is for the following IMC requirement: [CTNC-IFIMC1.1-TNCC-REQ-14-M].

[CTNC-IFIMC1.1-TNCC-TC-10] The test IMC will iterate through all mandatory IF-IMC API functions

TNC_TNCC_ReportMessageTypes,
TNC_TNCC_SendMessage,
TNC_TNCC_RequestHandshakeRetry

and verify that the TNCC under test implements these functions. This test case is for the following TNCC requirements: [CTNC-IFIMC1.1-TNCC-REQ-16-M], [CTNC-IFIMC1.1-TNCC-REQ-37-M] and [CTNC-IFIMC1.1-TNCC-REQ-42-M].

[CTNC-IFIMC1.1-TNCC-TC-11] The test IMC will not register for any messages via the TNC_TNCC_ReportMessageTypes function. The test IMC will validate that the TNCC never delivers any messages to it. This test case is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-41-M].

[CTNC-IFIMC1.1-TNCC-TC-12] The test IMC (either the Windows or Linux/UNIX Platform Bindings) will create eight (8) concurrent threads and each thread will make overlapping calls to the TNCC under test. The

test IMC will validate that TNCC under test does not crash, hang, or return inconsistent results. This test case is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-47-M]

[CTNC-IFIMC1.1-TNCC-TC-13] [Windows and Unix/Linux Platform Binding] The test IMC will call TNC_TNCC_BindFunction four (4) times to validate correct implementation. On the first call the test IMC will set the functionName parameter to "TNC_TNCC_ReportMessageTypes". On the second call the functionName parameter will be set to "TNC_TNCC_SendMessage", On third call the functionName parameter will be set to "TNC_TNCC_RequestHandshakeRetry". On the fourth call the functionName parameter will be set to "TNC_1234_UndefinedFunction". For the first three calls the pOutFunctionPointer is to be validated to be non-NULL, while in the last call the parameter value should be NULL. This test case is for the following TNCC requirements: [CTNC-IFIMC1.1-TNCC-REQ-51-M], [CTNC-IFIMC1.1-TNCC-REQ-52-M], [CTNC-IFIMC1.1-TNCC-REQ-59-M] and [CTNC-IFIMC1.1-TNCC-REQ-60-M].

[CTNC-IFIMC1.1-TNCC-TC-14] [Windows Platform Binding] Test case [CTNC-IFIMC1.1-TNCC-TC-13] will be expanded by repeating it twice and verifying that the pOutFunctionPointer values returned for each function do not change between the first and second cycle. This test case is for the following TNCC requirements: [CTNC-IFIMC1.1-TNCC-REQ-53-M].

[CTNC-IFIMC1.1-TNCC-TC-15] [Windows Platform Binding] The test IMC will populate the well-known registry key with the following bogus values and ensure that the TNCC under test does not crash or hang:

HKEY_LOCAL_MACHINE

→ SOFTWARE

→ *BOGUS_VALUE_1*

→ TRUSTED COMPUTING GROUP

→ *BOGUS_VALUE_2*

→ TNC

→ *BOGUS_VALUE_3*

→ IMC

→ *BOGUS_VALUE_4*

This test case is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-54-M].

[CTNC-IFIMC1.1-TNCC-TC-16] [Unix/Linux Platform Binding] The operator will run 'ldd' against the TNCC's .so file. The output from 'ldd' will be searched for the inclusion of the string "pthread", which is indicative of compliancy. This test case is for the following TNCC requirement: [CTNC-IFIMC1.1-TNCC-REQ-55-M].

Notes:

- There is no test case for requirements [CTNC-IFIMC1.1-TNCC-REQ-4-M] as all platform bindings defined in the IF-IMC 1.1 specification include Dynamic Function Binding.
- There is no test case for requirements [CTNC-IFIMC1.1-TNCC-REQ-17-M] as there are no optional to implement IF-IMC API functions currently defined in TNC standards.
- There is no test case for requirement [CTNC-IFIMC1.1-TNCC-REQ-32-M] as it is not possible in a simple fashion to determine whether a message originated from an IMV.

5 References

This section lists specifications and other documents that are referred to in the document. Since this document is informative (not normative), all of these references are informative with respect to this document.

Informative References

[1] Trusted Computing Group, *TNC Architecture for Interoperability*, Specification Version 1.1, May 2006.

[2] Trusted Computing Group, *TNC IF-IMC*, Specification Version 1.1, May 2006.

[3] Trusted Computing Group, *Compliance_TNC Compliance and Interoperability Principles*, Specification Version 1.0, Draft Specification, October 2006.