TCG Compliance_TNC IF-IMV Compliance Test Plan

Version 1.0
Revision 0.08
29 October 2007
Published

Contact:

admin@trustedcomputinggroup.org

TCG PUBLISHED

Copyright © 2006-2007 Trusted Computing Group, Incorporated.

Disclaimer

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-IMV 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-IMV 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-IMV compliance tests. Readers should be familiar with the TNC Architecture [1], with the Compliance_TNC Compliance and Interoperability Principles specification [3] and with IF-IMV v1.1.

2 Specifications and Components

2.1 Specifications

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

2.2 Components

There are two IF-IMV compliance tests that test the two kinds of components that interface with IF-IMV: Integrity Measurement Verifiers and TNC Servers.

2.2.1 IF-IMV Compliance Test for Integrity Measurement Verifiers (IMVs)

The IF-IMV Compliance test for Integrity Measurement Verifiers (IMVs) tests that an IMV properly implements IF-IMV. The Test Target for this test (the component under test) is an IMV.

One difficult aspect of this test is that most IMVs require an active IMC 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 IMV, allowing the IMV to communicate with the IMC.

2.2.2 IF-IMV Compliance Test for TNC Servers (TNCSs)

The IF-IMV Compliance test for TNC Servers (TNCSs) tests that a TNCS properly implements IF-IMV. The Test Target for this test (the component under test) is a TNCS.

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

3 Specifications Requirements

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

This section has two subsections. The first one lists requirements upon Integrity Measurement Verifiers, which are tested by the IF-IMV compliance test for IMVs. The second one lists requirements upon TNC Servers, which are tested by the IF-IMV compliance test for TNCSs.

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), "IFIMV1.1" (indicating that these are requirements from IF-IMV v1.1), "IMV" or "TNCS" depending on which component the requirement applies to, a requirement number unique within the preceding prefix, and compliance classifier ("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 IMVs

[CTNC-IFIMV1.1-IMV-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-IMV section 2.5 and section 3.2.4)

[CTNC-IFIMV1.1-IMV-REQ-2-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-IMV section 2.6.4)

[CTNC-IFIMV1.1-IMV-REQ-3-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 TNCSs and IMVs that support different version numbers interact safely. (IF-IMV section 3.2.2) On platforms that don't define a Dynamic Function Binding mechanism, all optional [IF-IMV API] functions MUST be implemented. (IF-IMV section 3.6)

[CTNC-IFIMV1.1-IMV-REQ-4-M]   An IMV or TNC Server MUST work properly if a vendor-specific function is not implemented by the other party […]. (IF-IMV section 3.2.4)

[CTNC-IFIMV1.1-IMV-REQ-5-M]   An IMV or TNC Server […] MUST ignore vendor-specific functions that it does not understand. (IF-IMV section 3.2.4)

[CTNC-IFIMV1.1-IMV-REQ-6-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-IMV section 3.2.4)

[CTNC-IFIMV1.1-IMV-REQ-7-M]   IMV DLLs also MUST be reentrant. (IF-IMV section 3.3)

Version 1.0

[CTNC-IFIMV1.1-IMV-REQ-8-M]	The TNC Server and all IMV DLLs MUST be thread-safe. (IF-IMV section 3.3)
[CTNC-IFIMV1.1-IMV-REQ-9-M]	The TNCS can choose any value for the IMV ID and the IMV MUST NOT attach any significance to the value chosen. (IF-IMV section 3.4.2.1)
[CTNC-IFIMV1.1-IMV-REQ-10-M]	As described in section 2.6.3 above, it is sometimes desirable to retry an Integrity Check Handshake (when remediation is complete, for instance). Some TNCSs will not support this but all IMVs MUST do so. (IF-IMV section 3.4.2.2)
[CTNC-IFIMV1.1-IMV-REQ-11-M]	[The TNC Server] can even choose a network connection ID that was used by a previous network connection that has now been deleted and is invalid. The IMV MUST NOT attach any significance to the value chosen. (IF-IMV section 3.4.2.2)
[CTNC-IFIMV1.1-IMV-REQ-12-M]	[For the handshake retry reason value,] The IMV MUST pass one of the values listed in section 3.5.6. The IMV MUST NOT use any other handshake retry reason value with this version of the IF-IMV API. (IF-IMV section 3.4.2.4) [Section 3.5.6 Handshake Retry Reason Values: This is the complete set of permissible values for the TNC_Retry_Reason type in this version of the IF-IMV API. TNC_RETRY_REASON_IMV_IMPORTANT_POLICY_CHANGE, TNC_RETRY_REASON_IMV_MINOR_POLICY_CHANGE, TNC_RETRY_REASON_IMV_SERIOUS_EVENT, TNC_RETRY_IMV_MINOR_EVENT, AND TNC_RETRY_REASON_IMV_PERIODIC]
[CTNC-IFIMV1.1-IMV-REQ-13-M]	[For the IMV action recommendation value,] The IMV MUST pass one of the values listed in section 3.5.7. The IMV MUST NOT use any other IMV Action Recommendation value with this version of the IF-IMV API. (IF-IMV section 3.4.2.5) [Section 3.5.7 IMV Action Recommendation Values: This is the complete set of permissible values for the TNC_IMV_Action_Recommendation type in this version of the IF-IMV API. TNC_IMV_ACTION_RECOMMENDATION_ALLOW, TNC_IMV_ACTION_RECOMMENDATION__NO_ACCESS, TNC_IMV_ACTION_RECOMMENDATION_ISOLATE, TNC_IMV_ACTION_RECOMMENDATION_NO_RECOMMENDATION]
[CTNC-IFIMV1.1-IMV-REQ-14-M]	[For the IMV evaluation result value,] The IMV MUST pass one of the values listed in section 3.5.8. The IMV MUST NOT use any other IMV Evaluation Result value with this version of IF-IMV API. (IF-IMV Section 3.4.2.6) [Section 3.5.8 IMV Evaluation Result Values: This is the complete set of permissible values for the TNC_IMV_Evaluation_Result type in this version [v1.1] of the IF-IMV API. TNC_IMV_EVALUATION_RESULT_COMPLIANT, TNC_IMV_EVALUATION_RESULT_NONCOMPLIANT_MINOR, TNC_IMV_EVALUATION_RESULT_NONCOMPLIANT_MAJOR, TNC_IMV_EVALUATION_RESULT_ERROR, TNC_IMV_EVALUATION_RESULT_DONT_KNOW]

[CTNC-IFIMV1.1-IMV-REQ-15-M] 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 IMV MUST NOT send messages whose message type includes one of these reserved values. (IF-IMV section 3.4.2.7)

[CTNC-IFIMV1.1-IMV-REQ-16-M] The message type TNC_VENDORID_ANY (0xffffff) is reserved as a wild card as described in section 3.8.1. IMVs may request messages with this vendor ID to indicate that they want to receive messages whose message type includes any vendor ID. However, an IMV MUST NOT send messages whose message type includes this reserved value and a TNCS MUST NOT deliver such messages. (IF-IMV section 3.4.2.9)

[CTNC-IFIMV1.1-IMV-REQ-17-M] The message subtype TNC_SUBTYPE_ANY (0xff) is reserved as a wild card as described in section 3.8.1. IMVs may request messages with this message subtype to indicate that they want to receive messages whose message subtype has any value. However, an IMV MUST NOT send messages whose message subtype includes this reserved value and a TNCS MUST NOT deliver such messages. (IF-IMV section 3.4.2.10)

[CTNC-IFIMV1.1-IMV-REQ-18-M] IMVs and TNCSs MUST be prepared for any function to return any result code. (IF-IMV section 3.4.2.12 and 3.5.2)

[CTNC-IFIMV1.1-IMV-REQ-19-M] The reserved value TNC_CONNECTIONID_ANY MUST NOT be used as a normal network connection ID. Instead, it may be passed to TNC_TNCS_RequestHandshakeRetry to indicate that handshake retry is requested for all current network connections. (IF-IMV section 3.5.4)

[CTNC-IFIMV1.1-IMV-REQ-20-M] Some of the functions in the IF-IMV API are marked as mandatory below. Mandatory [IF-IMV API] functions MUST be implemented. (IF-IMV section 3.6)

[CTNC-IFIMV1.1-IMV-REQ-21-M] An IMV or TNC Server MUST work properly if one or more optional [IF-IMV API] functions are not implemented by the other party. (IF-IMV section 3.6) There aren't any optional TNCS functions in IF-IMV 1.1. No test case is necessary for this requirement.

[CTNC-IFIMV1.1-IMV-REQ-22-M] The TNC Server calls this function [TNC_IMV_Initialize] to initialize the IMV and agree on the API version number to be used. It also supplies the IMV ID, an IMV identifier that the IMV must use when calling TNC Server callback functions. All IMVs MUST implement TNC_IMV_Initialize (IF-IMV section 3.7.1)

[CTNC-IFIMV1.1-IMV-REQ-23-M] The IMV MUST check these [minVersion and maxVersion] to determine whether there is an API version number that it supports in this range. (IF-IMV section 3.7.1)

[CTNC-IFIMV1.1-IMV-REQ-24-M] If not [if the API version number specified is not supported], the IMV MUST return TNC_RESULT_NO_COMMON_VERSION. (IF-IMV section 3.7.1)

[CTNC-IFIMV1.1-IMV-REQ-25-M] [TNC_IMV_NotifyConnectionChange is an optional function.] If the state is TNC_CONNECTION_STATE_DELETE, the IMV

MUST NOT pass this network connection ID to the TNC Server after this function [TNC_IMV_NotifyConnectionChange] returns (unless the TNCS later creates another network connection with the same network connection ID). (IF-IMV section 3.7.2)

[CTNC-IFIMV1.1-IMV-REQ-26-M] [TNC_IMV_ReceiveMessage is an optional function] The IMV MUST NOT ever modify the buffer contents [passed in the TNC_IMV_ReceiveMessage function] […]. (IF-IMV section 3.7.3)

[CTNC-IFIMV1.1-IMV-REQ-27-M] [TNC_IMV_ReceiveMessage is an optional function] The IMV MUST NOT access the buffer [passed in TNC_IMV_ReceiveMessage function] after TNC_IMV_ReceiveMessage has returned. If the IMV wants to retain the message, it should copy it before returning from TNC_IMV_ReceiveMessage. (IF-IMV section 3.7.3)

[CTNC-IFIMV1.1-IMV-REQ-28-M] [TNC_IMV_SolicitRecommendation] All IMVs MUST implement this function [TNC_IMV_SolicitRecommendation]. (IF-IMV section 3.7.4)

[CTNC-IFIMV1.1-IMV-REQ-29-M] [TNC_TNCS_ReportMessageTypes] The imvID [passed into the TNC_TNCS_ReportMessageTypes function] MUST contain the value provided to TNC_IMV_Initialize. (IF-IMV section 3.8.1)

[CTNC-IFIMV1.1-IMV-REQ-30-M] [TNC_TNCS_SendMessage] The imvID [passed into the TNC_TNCS_SendMessage function] MUST contain the value provided to TNC_IMV_Initialize […]. (IF-IMV section 3.8.2)

[CTNC-IFIMV1.1-IMV-REQ-31-M] [TNC_TNCS_SendMessage] […] The connectionID parameter [passed into the TNC_TNCS_SendMessage] MUST contain a valid network connection ID. (IF-IMV section 3.8.2)

[CTNC-IFIMV1.1-IMV-REQ-32-M] [TNC_TNCS_SendMessage] The IMV MUST NOT call this function [TNC_TNCS_SendMessage] unless it has received a call to TNC_IMV_ReceiveMessage, or TNC_IMV_BatchEnding for this connection and the IMV has not yet returned from that function. (IF-IMV Section 3.8.2)

[CTNC-IFIMV1.1-IMV-REQ-33-M] [TNC_TNCS_SendMessage] The IMV MUST NOT specify a message type whose vendor ID is 0xffffff or whose subtype is 0xff. (IF-IMV Section 3.8.2)

[CTNC-IFIMV1.1-IMV-REQ-34-M] [TNC_TNCS_RequestHandshakeRetry] An IMV calls this function [TNC_TNCS_RequestHandshakeRetry] to ask a TNCS to retry an Integrity Check Handshake. The IMV MUST pass its IMV ID as the imcID parameter […]. (IF-IMV section 3.8.3)

[CTNC-IFIMV1.1-IMV-REQ-35-M] [TNC_TNCS_RequestHandshakeRetry] An IMV calls this function [TNC_TNCS_RequestHandshakeRetry] to ask a TNCS to retry an Integrity Check Handshake. The IMV MUST pass […] a network connection ID as the connectionID parameter […]. (IF-IMV section 3.8.3)

[CTNC-IFIMV1.1-IMV-REQ-36-M] [TNC_TNCS_RequestHandshakeRetry] An IMV calls this function [TNC_TNCS_RequestHandshakeRetry] to ask a TNCS to retry an Integrity Check Handshake. The IMV MUST pass […] one of the handshake retry reasons listed in section

3.5.6, as the reason parameter. (IF-IMV section 3.8.3) [Section 3.5.6 Handshake Retry Reason Values: This is the complete set of permissible values for the TNC_Retry_Reason type in this version of the IF-IMC API: . TNC_RETRY_REASON_IMV_IMPORTANT_POLICY_CHANGE, TNC_RETRY_REASON_IMV_MINOR_POLICY_CHANGE, TNC_RETRY_REASON_IMV_SERIOUS_EVENT, TNC_RETRY_REASON_IMV_MINOR_EVENT, TNC_RETRY_REASON_IMV_PERIODIC.]

[CTNC-IFIMV1.1-IMV-REQ-37-M] [TNC_TNCS_ProvideRecommendation] The IMV MUST pass its IMV ID as the imvID parameter […]. (IF-IMV section 3.8.4)

[CTNC-IFIMV1.1-IMV-REQ-38-M] [TNC_TNCS_ProvideRecommendation] The IMV MUST pass […] a valid network connection ID as the connectionID parameter […]. (IF-IMV section 3.8.4)

[CTNC-IFIMV1.1-IMV-REQ-39-M] [TNC_TNCS_ProvideRecommendation] The IMV MUST pass […] one of the IMV Action Recommendation values listed in section 3.5.7 as the recommendation parameter […]. (IF-IMV section 3.8.4) [Section 3.5.7 IMV Action Recommendation Values: This is the complete set of permissible values for the TNC_IMV_Action_Recommendation type in this version of the IF-IMV API: TNC_IMV_ACTION_RECOMMENDATION_ALLOW, TNC_IMV_ACTION_RECOMMENDATION_NO_ACCESS, TNC_IMV_ACTION_RECOMMENDATION_ISOLATE, TNC_IMV_ACTION_RECOMMENDATION_NO_RECOMMENDATION.]

[CTNC-IFIMV1.1-IMV-REQ-40-M] [TNC_TNCS_ProvideRecommendation] The IMV MUST pass […] one of the IMV Evaluation Result values listed in section 3.5.8 as the evaluation parameter. (IF-IMV section 3.8.4) [Section 3.5.8 IMV Evaluation Result Values: This is the complete set of permissible values for the TNC_IMV_Evaluation_Result type in this version of the IF_IMV API: TNC_IMV_EVALUATION_RESULT_COMPLAINT, TNC_IMV_EVALUATION_RESULT_NONCOMPLAINT_MINOR, TNC_IMV_EVALUATION_RESULT_NONCOMPLAINT_MAJOR, TNC_IMV_EVALUATION_RESULT_ERROR, TNC_IMV_EVALUATION_RESULT_DONT_KNOW.]

[CTNC-IFIMV1.1-IMV-REQ-41-M] [TNC_TNCS_ProvideRecommendation] However, a TNCS MAY continue to deliver messages after an IMV calls TNC_TNCS_ProvideRecommendation, especially if other IMVs continue the dialog after the one IMV has rendered its decision. The IMV MUST be prepared for this. (IF-IMV section 3.8.4) This requirement is not testable. Therefore, there will be no test case for it.

[CTNC-IFIMV1.1-IMV-REQ-42-M] [Windows Platform Binding] The Microsoft Windows DLL platform binding for the IF_IMV API defines one additional function that MUST be implemented by IMVs implementing this platform binding. (IF-IMV Section 4.1.7) IMVs implementing the Microsoft Windows DLL platform binding

MUST define this additional platform-specific function [TNC_IMV_ProviderBindFunction]. (IF-IMV Section 4.1.7.1)

[CTNC-IFIMV1.1-IMV-REQ-43-M] [Windows Platform Binding] The IMV MUST set the imvID parameter [passed to TNC_TNCS_BindFunction function] to the IMV ID value provided to TNC_IMV_Initialize. (IF-IMV Section 4.1.8.1)

[CTNC-IFIMV1.1-IMV-REQ-44-M] [Windows Platform Binding] The IMV MUST set the functionName parameter [passed to TNC_TNCS_BindFunction function] to a pointer to a C string identifying the function whose pointer is desired (i.e., "TNC_TNCS_SendMessage") (IF-IMV Section 4.1.8.1)

[CTNC-IFIMV1.1-IMV-REQ-45-M] [Windows Platform Binding] The IMV MUST set the pOutFunctionPointer parameter [passed to the TNC_TNCS_BindFunction function] to a pointer to storage into which the desired function pointer will be stored. (IF-IMV Section 4.1.8.1)

[CTNC-IFIMV1.1-IMV-REQ-46-M] [Windows Platform Binding] A well-known registry key is used by the TNCS to load IMVs. […] TNC Servers and IMVs MUST ignore unrecognized values and keys. […] The only requirement, as stated above, is that TNC Servers and IMVs MUST ignore unrecognized values and keys. (IF-IMV Section 4.1.9)

[CTNC-IFIMV1.1-IMV-REQ-47-M] [UNIX/Linux Platform Binding] The UNIX/Linux Dynamic Linkage platform binding for the IF-IMV API defines one additional function that MUST be implemented by IMVs implementing this platform binding. (IF-IMV Section 4.2.8) IMVs implementing the UNIX/Linux Dynamic Linkage platform binding MUST define this additional platform-specific function [TNC_IMV_ProvideBindFunction]. (IF-IMV Section 4.2.8.1)

[CTNC-IFIMV1.1-IMV-REQ-48-M] [UNIX/Linux Platform Binding] The IMV MUST set the imvID parameter [passed to the TNC_TNCS_BindFunction] to the IMV ID value provided to TNC_IMV_Initialize. (IF-IMV Section 4.2.9.1)

[CTNC-IFIMV1.1-IMV-REQ-49-M] [UNIX/Linux Platform Binding] The IMV MUST set the functionName parameter [passed to the TNC_TNCS_BindFunction] to a pointer to a C string identifying the function whose pointer is desired (i.e., "TNC_TNCS_SendMessage"). (IF-IMV Section 4.2.9.1)

[CTNC-IFIMV1.1-IMV-REQ-50-M] [UNIX/Linux Platform Binding] The IMV MUST set the pOutFunctionPointer parameter [passed to the TNC_TNCS_BindFunction] to a pointer to storage into which the desired function pointer will be stored. (IF-IMV Section 4.2.9.1)

3.2 Requirements on TNC Servers

[CTNC-IFIMV1.1-TNCS-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-IMV section 2.5 and section 3.2.4)

[CTNC-IFIMV1.1-TNCS-REQ-2-M]	The TNCS MUST use the same connection ID for all IMVs when referring to a particular connection. (IF-IMV section 2.6.2)
[CTNC-IFIMV1.1-TNCS-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-IMV section 2.6.4)
[CTNC-IFIMV1.1-TNCS-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 TNCSs and IMVs that support different version numbers interact safely. (IF-IMV section 3.2.2) On platforms that don't define a Dynamic Function Binding mechanism, all optional [IF-IMV API] functions MUST be implemented. (IF-IMV section 3.6) All platform bindings defined in the IF-IMV 1.1 specification include Dynamic Function Binding so we don't need a test case for this requirement.
[CTNC-IFIMV1.1-TNCS-REQ-5-M]	An IMV or TNC Server MUST work properly if a vendor-specific function is not implemented by the other party […]. (IF-IMV section 3.2.4)
[CTNC-IFIMV1.1-TNCS-REQ-6-M]	An IMV or TNC Server […] MUST ignore vendor-specific functions that it does not understand. (IF-IMV section 3.2.4)
[CTNC-IFIMV1.1-TNCS-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-IMV section 3.2.4)
[CTNC-IFIMV1.1-TNCS-REQ-8-M]	The TNCS MUST be reentrant (able to receive and process a function call even when one is already underway). (IF-IMV section 3.3)
[CTNC-IFIMV1.1-TNCS-REQ-9-M]	The TNC Server and all IMV DLLs MUST be thread-safe. (IF-IMV section 3.3) [Windows Platform Binding] Unlike IMCs, IMV DLLs are required to be thread-safe. The IMV DLL MAY create threads. The TNC Server MUST be thread-safe. (IF-IMV Section 4.1.3) [UNIX/Linux Platform Binding] The IMV MAY create threads. The TNC Server MUST be thread-safe. (IF-IMV Section 4.2.4)
[CTNC-IFIMV1.1-TNCS-REQ-10-M]	[For the network connection state value,] the TNCS MUST pass one of the values listed in section 3.5.5[1]. The TNCS MUST NOT use any other network connection state value with this version of the IF-IMV API. (IF-IMV section 3.4.2.3) [TNC_IMV_NotifyConnectionChange is an optional function.] The TNC Server calls this function [TNC_IMV_NotifyConnectionChange] to inform the IMV that the state of the network connection identified by connectionID has changed to newState. Section 3.5.5 lists all the possible values of newState for this version of the IF-IMV API. The TNCS MUST NOT use any other values with this version of IF-IMV (IF-IMV section 3.7.2) The newState parameter MUST contain one of the values listed in section

---

[1] IF-IMV v1.1 specification references section 3.5.4. However, section 3.5.5 lists the complete set of permissible values for TNC_Connection_State type.

3.5.5. (IF-IMV section 3.7.2) [Section 3.5.5 Network Connection State Values: This is the complete set of permissible values for the TNC_Connection_State type in this version [v1.1] of IF-IMV API: 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-IFIMV1.1-TNCS-REQ-11-M] TNC Clients and TNC Servers MUST properly deliver messages with any message type (as described in section 2.6.4). (IF-IMV section 3.4.2.7)

[CTNC-IFIMV1.1-TNCS-REQ-12-M] The message type TNC_VENDORID_ANY (0xffffff) is reserved as a wild card as described in section 3.8.1. IMVs may request messages with this vendor ID to indicate that they want to receive messages whose message type includes any vendor ID. However, an IMV MUST NOT send messages whose message type includes this reserved value and a TNCS MUST NOT deliver such messages. (IF-IMV section 3.4.2.9)

[CTNC-IFIMV1.1-TNCS-REQ-13-M] The message subtype TNC_SUBTYPE_ANY (0xff) is reserved as a wild card as described in section 3.8.1. IMVs may request messages with this message subtype to indicate that they want to receive messages whose message subtype has any value. However, an IMV MUST NOT send messages whose message subtype includes this reserved value and a TNCS MUST NOT deliver such messages. (IF-IMV section 3.4.2.10)

[CTNC-IFIMV1.1-TNCS-REQ-14-M] IMVs and TNCSs MUST be prepared for any function to return any result code. (IF-IMV section 3.4.2.12 and 3.5.2)

[CTNC-IFIMV1.1-TNCS-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_TNCS_RequestHandshakeRetry to indicate that handshake retry is requested for all current network connections. (IF-IMV section 3.5.4)

[CTNC-IFIMV1.1-TNCS-REQ-16-M] Some of the functions in the IF-IMV API are marked as mandatory below. Mandatory [IF-IMV API] functions MUST be implemented. (IF-IMV section 3.6)

[CTNC-IFIMV1.1-TNCS-REQ-17-M] An IMV or TNC Server MUST work properly if one or more optional [IF-IMV API] functions are not implemented by the other party. (IF-IMV section 3.6)

[CTNC-IFIMV1.1-TNCS-REQ-18-M] The TNC Server MUST NOT call any other IF-IMV API functions for an IMV until it has successfully completed a call to TNC_IMV_Initialize() (IF-IMV section 3.7.1) [Windows Platform Binding] The TNCS MUST always call the TNC_IMV_Initialize function first. (IF-IMV section 4.1.1) [UNIX/Linux Platform Binding] The TNCS MUST always call the TNC_IMV_Initialize function first. (IF-IMV Section 4.2.1)

[CTNC-IFIMV1.1-TNCS-REQ-19-M] Once a call to this function [TNC_IMV_Initialize] completed successfully, this function MUST NOT be called again for a

particular IMV-TNCS pair until a call to TNC_IMV_Terminate has completed successfully. (IF-IMV section 3.7.1)

[CTNC-IFIMV1.1-TNCS-REQ-20-M] The TNC Server MUST set minVersion to the minimum IF-IMV API version number that it supports […] (IF-IMV section 3.7.1.)

[CTNC-IFIMV1.1-TNCS-REQ-21-M] [The TNC Server] MUST set maxVersion to the maximum API version number that it supports. (IF-IMV section 3.7.1)

[CTNC-IFIMV1.1-TNCS-REQ-22-M] The TNC Server also MUST set pOutActualVersion so that the IMV 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-IMV section 3.7.1)

[CTNC-IFIMV1.1-TNCS-REQ-23-M] [TNC_IMV_NotifyConnectionChange is an optional function.] The imvID parameter (passed in to TNC_IMV_NotifyConnectionChange function) MUST contain the IMV ID value provided to TNC_IMV_Initialize. (IF-IMV section 3.7.2)

[CTNC-IFIMV1.1-TNCS-REQ-24-M] [TNC_IMV_NotifyConnectionChange is an optional function.] The connectionID parameter (passed in to TNC_IMV_NotifyConnectionChange function) MUST contain a valid network connection ID. (IF-IMV section 3.7.2)

[CTNC-IFIMV1.1-TNCS-REQ-25-M] [TNC_IMV_ReceiveMessage is an optional function] The TNC Server calls this function [TNC_IMV_ReceiveMessage] to deliver a message to the IMV. 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 IMC (or a TNCC or other party acting as an IMC). The (IF-IMV section 3.7.3) This requirement is not automatically testable; therefore, there will be no test case corresponding to this requirement.

[CTNC-IFIMV1.1-TNCS-REQ-26-M] [TNC_IMV_ReceiveMessage is an optional function.] The imvID parameter [passed into TNC_IMV_ReceiveMessage function] MUST contain the IMV ID value provided to TNC_IMV_Initialize. (IF-IMV section 3.7.3)

[CTNC-IFIMV1.1-TNCS-REQ-27-M] [TNC_IMV_ReceiveMessage is an optional function.] The connectionID parameter [passed into TNC_IMV_ReceiveMessage function] MUST contain a valid network connection ID. (IF-IMV section 3.7.3)

[CTNC-IFIMV1.1-TNCS-REQ-28-M] [TNC_IMV_ReceiveMessage is an optional function.] The message parameter [passed into TNC_IMV_ReceiveMessage function] MUST contain a reference to a buffer containing the message being delivered to the IMV. (IF-IMV section 3.7.3)

[CTNC-IFIMV1.1-TNCS-REQ-29-M] [TNC_IMV_ReceiveMessage is an optional function.] The messageLength parameter [passed into TNC_IMV_ReceiveMessage function] MUST contain the number of octets in the message. (IF-IMV section 3.7.3)

[CTNC-IFIMV1.1-TNCS-REQ-30-M] [TNC_IMV_ReceiveMessage is an optional function.] The messageType parameter [passed into

TNC_IMV_ReceiveMessage function] MUST contain the type of the message. It MUST match one of the TNC_MessageType values previously supplied by the IMV to the TNCS in the IMV's most recent call to TNC_TNCS_ReportMessageTypes. (IF-IMV section 3.7.3)

[CTNC-IFIMV1.1-TNCS-REQ-31-M] [TNC_IMV_SolicitRecommendation] The imvID parameter [passed into TNC_IMV_SolicitRecommendation function] MUST contain the IMV ID value provided to TNC_IMV_Initialize. (IF-IMV section 3.7.4)

[CTNC-IFIMV1.1-TNCS-REQ-32-M] [TNC_IMV_SolicitRecommendation] The connectionID parameter [passed into TNC_IMV_SolicitRecommendation function] MUST contain a valid network connection ID. (IF-IMV section 3.7.4)

[CTNC-IFIMV1.1-TNCS-REQ-33-M] [TNC_IMV_BatchEnding is an optional function.] The imvID parameter [passed into TNC_IMV_BatchEnding function] MUST contain the IMV ID value provided to TNC_IMV_Initialize. (IF-IMV section 3.7.5)

[CTNC-IFIMV1.1-TNCS-REQ-34-M] [TNC_IMV_BatchEnding is an optional function.] The connectionID parameter [passed into TNC_IMV_BatchEnding function] MUST contain a valid network connection ID. (IF-IMV section 3.7.5)

[CTNC-IFIMV1.1-TNCS-REQ-35-M] [TNC_IMV_Terminate is an optional function.] Once a call to TNC_IMV_Terminate is made, the TNC Server MUST NOT call the IMV except to call TNC_IMV_Initialize (which may not succeed if the IMV cannot initialize itself). (IF-IMV section 3.7.6)

[CTNC-IFIMV1.1-TNCS-REQ-36-M] [TNC_IMV_Terminate is an optional function.] The imvID parameter [passed into TNC_IMV_Terminate function] MUST contain the IMV ID value provided to TNC_IMV_Initialize. (IF-IMV section 3.7.6)

[CTNC-IFIMV1.1-TNCS-REQ-37-M] [TNC_TNCS_ReportMessageTypes] All TNC Servers MUST implement this function [TNC_TNCS_ReportMessageTypes]. (IF-IMV section 3.8.1)

[CTNC-IFIMV1.1-TNCS-REQ-38-M] [TNC_TNCS_ReportMessageTypes] The TNC Server MUST NOT ever modify the list of message types […] (IF-IMV section 3.8.1)

[CTNC-IFIMV1.1-TNCS-REQ-39-M] [TNC_TNCS_ReportMessageTypes] The TNC Server […] MUST NOT access this list after TNC_TNCS_ReportMessageTypes has returned. (IF-IMV section 3.8.1)

[CTNC-IFIMV1.1-TNCS-REQ-40-M] [TNC_TNCS_ReportMessageTypes] TNC Servers MUST support any message type [for the TNC_TNCS_ReportMessageTypes]. (IF-IMV section 3.8.1)

[CTNC-IFIMV1.1-TNCS-REQ-41-M] [TNC_TNCS_ReportMessageTypes] Note that although all TNC Servers must implement this function, some IMVs may never call it if they don't support receiving any message types. This is acceptable. In such a case, the TNC Server MUST NOT deliver any messages to the IMV. (IF-IMV section 3.8.1)

[CTNC-IFIMV1.1-TNCS-REQ-42-M] [TNC_TNCS_SendMessage] All TNC Servers MUST implement this function [TNC_TNCS_SendMessage]. (IF-IMV section 3.8.2)

[CTNC-IFIMV1.1-TNCS-REQ-43-M] [TNC_TNCS_SendMessage] The TNC Server MUST NOT ever modify the buffer contents […]. (IF-IMV section 3.8.2)

[CTNC-IFIMV1.1-TNCS-REQ-44-M] [TNC_TNCS_SendMessage] The TNC Server […] MUST NOT access the buffer after TNC_TNCS_SendMessage has returned. (IF-IMV section 3.8.2)

[CTNC-IFIMV1.1-TNCS-REQ-45-M] [TNC_TNCS_SendMessage] The TNC Server MUST support any message type [for the TNC_TNCS_SendMessage function]. (IF-IMV Section 3.8.2)

[CTNC-IFIMV1.1-TNCS-REQ-46-M] [Windows Platform Binding] The TNCS MUST listen for changes to the well-known registry key so that it can load and unload IMVs dynamically. However, the TNCS SHOULD delay before making changes based on registry key changes since it is common for these changes to come in batches within a few seconds during an install process. Unlike a TNCC, a TNCS MUST NOT ignore such changes. (IF-IMV section 4.1.1)

[CTNC-IFIMV1.1-TNCS-REQ-47-M] [Windows Platform Binding] IMVs implementing the Microsoft Windows DLL platform binding MUST define this additional platform-specific function [TNC_IMV_ProvideBindFunction]. The TNC Server MUST call this function immediately after calling TNC_IMV_Initialize to provide a pointer to the TNCS bind function. (IF-IMV Section 4.1.7.1)

[CTNC-IFIMV1.1-TNCS-REQ-48-M] [Windows Platform Binding] The imvID parameter [passed to the TNC_IMV_ProvideBindFunction function] MUST contain the value provided to TNC_IMV_Initialize. (IF-IMV Section 4.1.7.1)

[CTNC-IFIMV1.1-TNCS-REQ-49-M] [Windows Platform Binding] The bindFunction parameter [passed to the TNC_IMV_ProvideBindFunction function] MUST contain a pointer to the TNCS bind function. (IF-IMV Section 4.1.7.1)

[CTNC-IFIMV1.1-TNCS-REQ-50-M] [Windows Platform Binding] The Microsoft Windows DLL platform binding for the IF-IMV API defines one additional function that MUST be implemented by TNC Servers implementing this platform binding. (IF-IMV Section 4.1.8) TNC Servers implementing the Microsoft Windows DLL platform binding MUST define this additional platform-specific function [TNC_TNCS_BindFunction]. (IF-IMV Section 4.1.8.1)

[CTNC-IFIMV1.1-TNCS-REQ-51-M] [Windows Platform Binding] If the TNCS 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-IMV Section 4.1.8.1)

[CTNC-IFIMV1.1-TNCS-REQ-52-M] [Windows Platform Binding] Once an IMV obtains a pointer to a particular function [through TNC_TNCS_BindFunction], the TNCS MUST always return the same function pointer

value to that IMV for that function name. (IF-IMV Section 4.1.8.1)

[CTNC-IFIMV1.1-TNCS-REQ-53-M] [Windows Platform Binding] A well-known registry key is used by the TNCS to load IMCs. […] TNC Servers and IMVs MUST ignore unrecognized values and keys. […] The only requirement, as stated above, is that TNC Servers and IMVs MUST ignore unrecognized values and keys. (IF-IMV Section 4.1.9)

[CTNC-IFIMV1.1-TNCS-REQ-54-M] [UNIX/Linux Platform Binding] The TNC Server MUST call the function [TNC_IMV_ProvideBindFunction] immediately after calling TNC_IMV_Initialize to provide a pointer to the TNCS bind function. (IF-IMV Section 4.2.8.1)

[CTNC-IFIMV1.1-TNCS-REQ-55-M] [UNIX/Linux Platform Binding] The imvID parameter [passed to the TNC_IMV_ProvideBindFunction function] MUST contain the value provided to TNC_IMV_Initialize. (IF-IMV section 4.2.8.1)

[CTNC-IFIMV1.1-TNCS-REQ-56-M] [UNIX/Linux Platform Binding] The bindFunction parameter [passed to the TNC_IMV_ProvideBindFunction function] MUST contain a pointer to the TNCS bind function. (IF-IMV section 4.2.8.1)

[CTNC-IFIMV1.1-TNCS-REQ-57-M] [UNIX/Linux Platform Binding] The UNIX/Linux Dynamic Linkage platform binding for the IF-IMV API defines one additional function that MUST be implemented by TNC Servers implementing this platform binding. (IF-IMV section 4.2.9) TNC Servers implementing the UNIX/Linux Dynamic Linkage platform binding MUST define this additional platform-specific function [TNC_TNCS_BindFunction]. (IF-IMV section 4.2.9.1)

[CTNC-IFIMV1.1-TNCS-REQ-58-M] [UNIX/Linux Platform Binding] An IMV can use this function [TNC_TNCS_BindFunction] to obtain pointers to other TNCS functions. […] The IMV MUST set the pOutFunctionPointer parameter to a pointer to storage into which the desired function pointer will be stored. If the TNCS 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-IMV section 4.2.9.1)

3.3 Other Requirements

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

[CTNC-IFIMV1.1-OTHER-REQ-1-M] [UNIX/Linux Platform Binding] A line that begins with "IMV" (U+0049, U+004D, U+0043, U+0020) specifies an IMV that may be loaded. The next character MUST be U+0022 (QUOTATION MARK). This MUST be followed by a human-readable IMV name (potentially zero length) and another U+0022 character (QUOTATION MARK). Of course, the IMV 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 IMV executable

file (up to but not including the U+000A that terminates the line). The path to the IMV executable file MUST NOT be a partial path. The /etc/tnc_config file must not contain IMVs with the same human-readable name.   (IF-IMV 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-IMV Compliance Test for IMVs

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

[CTNC-IFIMV1.1-IMV-AA-1]   Once loaded and initialized, an IMV may initiate a handshake retry at anytime and the test program must have code that verifies the IMV under test only uses valid imvID, connection ID, and handshake retry reason values and no others. This entry covers the following IMV requirements: [CTNC-IFIMV1.1-IMV-REQ-12-M], [CTNC-IFIMV1.1-IMV-REQ-34-M], [CTNC-IFIMV1.1-IMV-REQ-35-M], and [CTNC-IFIMV1.1-IMV-REQ-36-M].

[CTNC-IFIMV1.1-IMV-AA-2]   The test program that loads the IMV must have code that detects if the IMV under test passes only permissible TNC_IMV_Action_Recommendation type and TNC_IMV_Evaluation_Result type to the TNC_TNCS_ProvideRecommendation function. This entry covers the following IMV requirements: [CTNC-IFIMV1.1-IMV-REQ-13-M], [CTNC-IFIMV1.1-IMV-REQ-14-M], [CTNC-IFIMV1.1-IMV-REQ-39-M], and [CTNC-IFIMV1.1-IMV-REQ-40-M].

[CTNC-IFIMV1.1-IMV-AA-3]   Once loaded and initialized, an IMV can send messages at anytime and the test program must have code that detects if the IMV under test ever sends messages with reserved message type values where NOT allowed. This entry covers the following IMV requirements: [CTNC-IFIMV1.1-IMV-REQ-15-M], [CTNC-IFIMV1.1-IMV-REQ-16-M], [CTNC-IFIMV1.1-IMV-REQ-17-M], and [CTNC-IFIMV1.1-IMV-REQ-33-M].

[CTNC-IFIMV1.1-IMV-AA-4]   Once loaded and initialized, an IMV uses a network connection ID when communicating with the TNCS. The test program that loads IMVs will contain code to detect if the IMV under test ever uses an invalid value for the network connection ID, such as one that is in "delete" state or the reserved value (TNC_CONNECTIONID_ANY other than in the instance a handshake retry is being requested for all current network connections). This entry covers the following IMV requirements: [CTNC-IFIMV1.1-IMV-REQ-19-M], [CTNC-IFIMV1.1-IMV-REQ-25-M], and [CTNC-IFIMV1.1-IMV-REQ-31-M].

[CTNC-IFIMV1.1-IMV-AA-5]   The receive message function passes content to an IMV via receive message buffer. The test program that loads IMVs will contain code to verify that the contents of the receive message buffer are unmodified on return of the receive message function. This test case will be expanded by employing memory protection in the test program to detect if the IMV under test ever attempts to access a receive message buffer after the IMV returns from

the receive message function. This entry covers the following IMV requirements [CTNC-IFIMV1.1-IMV-REQ-26-M] and [CTNC-IFIMV1.1-IMV-REQ-27-M].

[CTNC-IFIMV1.1-IMV-AA-6]   Once loaded and initialized, an IMV uses an IMV ID when communicating with the TNCS. The test program that loads IMVs will contain code to detect if the IMV under test uses its IMV ID that was provided to TNC_IMV_Initialize. This entry covers the following IMV requirements: [CTNC-IFIMV1.1-IMV-REQ-29-M], [CTNC-IFIMV1.1-IMV-REQ-30-M], [CTNC-IFIMV1.1-IMV-REQ-37-M], [CTNC-IFIMV1.1-IMV-REQ-43-M], and [CTNC-IFIMV1.1-IMV-REQ-48-M].

[CTNC-IFIMV1.1-IMV-AA-7]   The test program for Windows and Unix/Linux that loads IMVs will have code to verify that the IMV under test always sends a valid function name string that conforms to the "TNC_TNCS_" or "TNC_XXX_" format (where XXX is a vendor ID) when the IMV calls the TNCS platform bind function. This test case is for the following IMV requirements: [CTNC-IFIMV1.1-IMV-REQ-44-M] and [CTNC-IFIMV1.1-IMV-REQ-49-M].

[CTNC-IFIMV1.1-IMV-AA-8]   The test program for Windows and Unix/Linux that loads IMVs will have code to verify that an IMV always passes function pointer storage when the IMV calls the TNCS 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 IMV requirements: [CTNC-IFIMV1.1-IMV-REQ-45-M] and [CTNC-IFIMV1.1-IMV-REQ-50-M].

[CTNC-IFIMV1.1-IMV-AA-9]   The test program that loads IMVs will contain code to detect if the IMV under test ever attempts to call the send message function for a connection for which the IMV is not servicing a receive message, or batch ending function call and has yet to return. This test case is for the following IMV requirement: [CTNC-IFIMV1.1-IMV-REQ-32-M].

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

[CTNC-IFIMV1.1-IMV-TC-1]   The test program that loads the IMV under test will iterate through all the functions defined by the IMV and ensure that each of these is either an IMV function defined in the IF-IMV 1.1 specification or has a name that begins with "TNC_XXX_" where XXX is a valid vendor ID. This case will be expanded to 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 IMV requirements: [CTNC-IFIMV1.1-IMV-REQ-1-M] and [CTNC-IFIMV1.1-IMV-REQ-6-M].

[CTNC-IFIMV1.1-IMV-TC-2]   The test program that loads the IMC will load two copies of the IMC so that two copies of each IMC message will be sent to the IMV under test. The IMV 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 IMV must follow the IF-IMV specification and must not crash. This test case is for the following IMV requirement: [CTNC-IFIMV1.1-IMV-REQ-2-M].

[CTNC-IFIMV1.1-IMV-TC-3]   The test program that loads the IMV will implement no vendor-specific functions. The IMV must follow the IF-IMV specification and must not crash. This test case is for the following IMV requirement: [CTNC-IFIMV1.1-IMV-REQ-4-M].

[CTNC-IFIMV1.1-IMV-TC-4]   The test program that loads the IMV will implement some extra vendor-specific functions that the IMV does not understand. The IMV must follow the IF-IMV specification and must not crash. This test case is for the following IMV requirement: [CTNC-IFIMV1.1-IMV-REQ-5-M].

[CTNC-IFIMV1.1-IMV-TC-5]   The test program that loads the IMV calls into the IMV while the IMV is calling the TNCS. For instance, the test program waits until the IMV calls TNC_TNCS_ProvideRecommendation and then calls TNC_IMV_NotifyConnectionChange to notify the IMV about the creation of a different connection. This test case is for the following IMV requirement: [CTNC-IFIMV1.1-IMV-REQ-7-M].

[CTNC-IFIMV1.1-IMV-TC-6]   The test program that loads the IMV starts up two threads. Each thread runs a series of handshakes, inserting short and varying (but not random) delays into the handshakes. The test program verifies that IMV does not crash or otherwise violate the spec when running multiple threads. This test case is for the following IMV requirement: [CTNC-IFIMV1.1-IMV-REQ-8-M].

[CTNC-IFIMV1.1-IMV-TC-7]   The test program that loads IMVs will try loading the IMV several times with different IMV ID values (including edge cases like 0, 1, and the maximum TNC_UInt32 value) and verify that the IMV functions properly with all of these values. This test case is for the following IMV requirement: [CTNC-IFIMV1.1-IMV-REQ-9-M].

[CTNC-IFIMV1.1-IMV-TC-8]   The test program that loads IMVs will run a handshake retry and verify that the IMV under test functions properly during this handshake retry (in particular, that it does not crash or hang). This test case is for the following IMV requirement: [CTNC-IFIMV1.1-IMV-REQ-10-M].

[CTNC-IFIMV1.1-IMV-TC-9]   The test program that loads IMVs 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 IMV under test functions properly during the second handshake (in particular, that it does not crash or hang). This test case is for the following IMV requirement: [CTNC-IFIMV1.1-IMV-REQ-11-M].

[CTNC-IFIMV1.1-IMV-TC-10]   The test program that loads IMVs will return the values of 0x0, 0xFFFFFFFF, and 0xA as result codes during individual calls for each of the five TNCS functions used by the IMV: TNC_TNCS_BindFunction, TNC_TNCS_ProvideRecommendation, TNC_TNCS_ReportMessageTypes, TNC_TNCS_RequestHandshakeRetry, TNC_TNCS_SendMessage , . This test case is for the following IMV requirement [CTNC-IFIMV1.1-IMV-REQ-18-M].

[CTNC-IFIMV1.1-IMV-TC-11]   The test program that loads IMVs, for both Windows and Unix/Linux, will iterate through all mandatory IF-IMV API functions including TNC_IMV_ProvideBindFunction and verify that the IMV under test implements these functions. This test case is for the following IMV requirements: [CTNC-IFIMV1.1-

IMV-REQ-20-M], [CTNC-IFIMV1.1-IMV-REQ-22-M], [CTNC-IFIMV1.1-IMV-REQ-28-M], [CTNC-IFIMV1.1-IMV-REQ-42-M], and [CTNC-IFIMV1.1-IMV-REQ-47-M].

[CTNC-IFIMV1.1-IMV-TC-12]   The test case [CTNC-IFIMV1.1-IMV-TC-11] will be expanded by iteratively calling the IMV'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 IMV 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 IMV requirements: [CTNC-IFIMV1.1-IMV-REQ-23-M] and [CTNC-IFIMV1.1-IMV-REQ-24-M].

[CTNC-IFIMV1.1-IMV-TC-13]   The test program for Windows that loads IMVs will populate the well known registry location with optional values and verify that an IMV ignores unknown optional values correctly and loads without hanging or crashing. This test case is for the following IMV requirement: [CTNC-IFIMV1.1-IMV-REQ-46-M].

NOTES:

- There is no test case for [CTNC-IFIMV1.1-IMV-REQ-3-M] requirement because all platform bindings defined in the IF-IMV 1.1 specification include Dynamic Function Binding.

4.2 Test Cases for IF-IMV Compliance Test for TNCSs

There are several asynchronous actions that a TNCS may initiate and which the test program must be capable of handling during all normative test cases. For example, a TNCS may call TNC_IMV_NotifyConnectionChange to provide a notification of a new network connection at any time. The test program must handle TNCS asynchronous actions appropriately and ensure the TNCS is generating correct data/messages/etc for the given action. Following is the list of asynchronous actions and validation requirements for the test program.

[CTNC-IFIMV1.1-TNCS-AA-1]   The test IMV must implement TNC_IMV_NotifyConnectionChange and verify that the TNCS under test sends only one of the valid network connection state values listed in section 3.5.5 of IF-IMV v1.1, The test IMV will check the network connection state value to make sure it is one of the values listed in IF-IMV section 3.4.2.3. This test case is for the following TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-10-M].

[CTNC-IFIMV1.1-TNCS-AA-2]   The test IMV verifies that the imvID parameter passed by the TNCS is the IMV ID value provided to TNC_IMV_Initialize. The test IMV also verifies that the connection ID also contains a valid network connection ID. The network connection ID must never have the reserved value TNC_CONNECTIONID_ANY. Further, the connection ID must never have the value TNC_CONNECTIONID_DELETE except when passed to the TNC_IMV_NotifyConnectionChange function. This test case is for the following TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-15-M], [CTNC-IFIMV1.1-TNCS-REQ-23-M], [CTNC-IFIMV1.1-TNCS-REQ-24-M], [CTNC-IFIMV1.1-TNCS-REQ-26-M], [CTNC-IFIMV1.1-TNCS-REQ-27-M], [CTNC-IFIMV1.1-TNCS-REQ-31-M], [CTNC-IFIMV1.1-TNCS-REQ-32-M], [CTNC-IFIMV1.1-TNCS-REQ-33-M], [CTNC-IFIMV1.1-TNCS-REQ-34-

M], [CTNC-IFIMV1.1-TNCS-REQ-36-M], [CTNC-IFIMV1.1-TNCS-REQ-48-M], and [CTNC-IFIMV1.1-TNCS-REQ-55-M].

[CTNC-IFIMV1.1-TNCS-AA-3]   The test IMV calls TNC_TNCS_BindFunction for all TNCS functions and verifies pOutFunctionPointer is set to either a pointer to the requested function or NULL when the requested function is not defined. The test IMV also verifies that the same function pointer is returned once the test IMV obtains a pointer to a particular function. This test case is for the following TNCS requirements: [CTNC-IFIMV1.1-TNCS-REQ-51-M], [CTNC-IFIMV1.1-TNCS-REQ-52-M], and [CTNC-IFIMV1.1-TNCS-REQ-58-M]

[CTNC-IFIMV1.1-TNCS-AA-4]   The test IMV will have code to detect if the TNCS under test ever calls TNC_IMV_Initialize again after having called it successfully before, unless TNC_IMV_Terminate() has completed successfully. This entry is for the following TNCS requirements: [CTNC-IFIMV1.1-TNCS-REQ-18-M] and [CTNC-IFIMV1.1-TNCS-REQ-19-M].

[CTNC-IFIMV1.1-TNCS-AA-5]   The test IMV will have code to detect if the TNCS under test ever advertises an invalid or incorrect minVersion and maxVersion argument with TNC_IMV_Initialize. This entry is for the following TNCS requirements: [CTNC-IFIMV1.1-TNCS-REQ-20-M], and [CTNC-IFIMV1.1-TNCS-REQ-21-M].

[CTNC-IFIMV1.1-TNCS-AA-6]   The test IMV will have code to verify that a TNCS always passes valid function pointer storage in pOutActualVersion when the TNCS calls the IMV initialization function TNC_IMV_Initialize. Verification of function pointer storage can be done by verifying that function pointer is not NULL. This entry is for the following TNCS requirement:[CTNC-IFIMV1.1-TNCS-REQ-22-M].

[CTNC-IFIMV1.1-TNCS-AA-7]   The test IMV will have code to verify that always when a TNCS is calling the [optional] TNC_IMV_ReceiveMessage() function:

    (a) message parameter contains a valid reference (e.g. memory pointer) to the buffer containing the message being delivered to the test IMV. 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 buffer reference.

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

    This entry is for the following TNCS requirements: [CTNC-IFIMV1.1-TNCS-REQ-28-M], [CTNC-IFIMV1.1-TNCS-REQ-29-M], and [CTNC-IFIMV1.1-TNCS-REQ-30-M]

[CTNC-IFIMV1.1-TNCS-AA-8]   The test IMV will have code to verify that a TNCS never calls any IMV function other than TNC_IMV_Initialize after calling the [optional] TNC_IMV_Terminate function. This entry is for the

following TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-35-M].

[CTNC-IFIMV1.1-TNCS-AA-9]   The test IMV has code to detect that the TNCS under test calls TNC_IMV_ProvideBindFunction immediately after calling TNC_IMV_Initialize. The test IMV also has code to detect that the bindFunction parameter contains a pointer to the TNCS bind function. This test case is for the following TNCS requirements: [CTNC-IFIMV1.1-TNCS-REQ-47-M], [CTNC-IFIMV1.1-TNCS-REQ-49-M], [CTNC-IFIMV1.1-TNCS-REQ-54-M], and [CTNC-IFIMV1.1-TNCS-REQ-56-M].

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

[CTNC-IFIMV1.1-TNCS-TC-1]   The test IMV will iterate through all the functions defined by the TNCS and ensure that each of these is either a TNCS function defined by the IF-IMV 1.1 specification or has a name that begins with "TNC_XXX_" where XXX is a valid vendor ID. This test case will be expanded to 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 TNCS requirements: [CTNC-IFIMV1.1-TNCS-REQ-1-M] and [CTNC-IFIMV1.1-TNCS-REQ-7-M].

[CTNC-IFIMV1.1-TNCS-TC-2]   Several test IMVs will be created. When a connection starts, these IMVs will check that they all get the same connection ID for that connection. This test case is for the following TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-2-M].

[CTNC-IFIMV1.1-TNCS-TC-3]   The test IMV will send a zero length message and the test IMC will verify that it is delivered properly. This test case is for the following TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-3-M]

[CTNC-IFIMV1.1-TNCS-TC-4]   The test IMV will implement no vendor-specific functions. The TNCS must follow the IF-IMV specification and must not crash. This test case is for the following TNCS requirement: [CTNC-IFIMV1.1-IMV-REQ-5-M].

[CTNC-IFIMV1.1-TNCS-TC-5]   The test IMV will implement some extra vendor-specific functions that the TNCS does not understand. The TNCS must follow the IF-IMV specification and must not crash. This test case is for the following TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-6-M].

[CTNC-IFIMV1.1-TNCS-TC-6]   The test IMV will wait until the TNCS calls TNC_IMV_BatchEnding. While that call is in progress, the test IMV will call TNC_TNCS_SendMessage. This test case is for the following TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-8-M].

[CTNC-IFIMV1.1-TNCS-TC-7]   The test IMV (either the Windows or Linux/UNIX Platform Bindings) will create eight (8) concurrent threads and each thread will make overlapping calls to the TNCS under test. The test IMV will validate that TNCS under test does not crash, hang, or return inconsistent results. This test case is for the following TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-9-M].

[CTNC-IFIMV1.1-TNCS-TC-8]   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 TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-11-M].

[CTNC-IFIMV1.1-TNCS-TC-9]   The test case for the TNCS requirement [CTNC-IFIMV1.1-TNCS-REQ-11-M] will be expanded. The test IMV sends two messages including one message whose message type includes the reserved TNC_VENDORID_ANY (0xFFFFFF). The test IMC 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 IMV sends two messages including one message whose message type includes the reserved TNC_SUBTYPE_ANY (0xFF). The test IMC 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 TNCS requirements: [CTNC-IFIMV1.1-TNCS-REQ-12-M] and [CTNC-IFIMV1.1-TNCS-REQ-13-M].

[CTNC-IFIMV1.1-TNCS-TC-10]   The test IMV implements IMV functions (TNC_IMV_Initialize, TNC_IMV_NotifyConnection Change, TNC_IMV_ReceiveMessage, TNC_IMV_SolicitRecommendation, TNC_IMV_BatchEnding, and TNC_IMV_Terminate) and has these functions return values of 0, 10, and FFFFFFFF. The TNCS under test must operate normally without hanging or crashing. This test case is for the following TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-14-M].

[CTNC-IFIMV1.1-TNCS-TC-11]   The test IMV calls all mandatory functions (TNC_TNCS_ReportMessageTypes, TNC_TNCS_SendMessage, TNC_TNCS_ProvideRecommendation, and TNC_TNCS_RequestHandshakeRetry) and verifies that they are implemented. This test case is for the following TNCS requirements: [CTNC-IFIMV1.1-TNCS-REQ-16-M], [CTNC-IFIMV1.1-TNCS-REQ-37-M], and [CTNC-IFIMV1.1-TNCS-REQ-42-M].

[CTNC-IFIMV1.1-TNCS-TC-12]   The test IMV does not implement any of the optional IMV functions (TNC_IMV_NotifyConnectionChange, TNC_IMV_ReceiveMessage, TNC_IMV_BatchEnding, and TNC_IMV_Terminate). The TNCS under test must operate properly (not crash or hang). This test case is for the following TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-17-M].

[CTNC-IFIMV1.1-TNCS-TC-13]   The test IMV has code to verify that the list of message types passed in supportedTypes has not been modified or accessed after the call returns. One method of performing this validation is to use protected memory. Additionally, the test IMV passes in boundary cases for the TNC_UINT32 type as a list of message types to check if the TNCS under test handles any message type. This test case is for the following TNCS requirements:, [CTNC-IFIMV1.1-TNCS-REQ-38-M], [CTNC-IFIMV1.1-TNCS-REQ-39-M], [CTNC-IFIMV1.1-TNCS-REQ-40-M], and [CTNC-IFIMV1.1-TNCS-REQ-41-M]

[CTNC-IFIMV1.1-TNCS-TC-14] The test IMV has codes to verify that the buffer content passed into TNC_TNCS_SendMessage as message argument has not been modified or otherwise accessed after the call returns. The test suite also has codes to compare that the message sent by TNCS is the same as the message the test IMV sent. This test case is for the following TNCS requirements: [CTNC-IFIMV1.1-TNCS-REQ-43-M], and [CTNC-IFIMV1.1-TNCS-REQ-44-M].

[CTNC-IFIMV1.1-TNCS-TC-15] The test suite updates the well-known registry key to change the list of IMVs to load and unload. The test suite verifies that the TNCS under test reflects such changes within 2 minutes by loading and unloading IMVs as specified in the well-known registry key. This test case is for the following TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-46-M].

[CTNC-IFIMV1.1-TNCS-TC-16] The test IMV calls TNC_TNCS_BindFunction to verify that the TNCS under test implements this function. The test case is for the following TNCS requirements: [CTNC-IFIMV1.1-TNCS-REQ-50-M] and [CTNC-IFIMV1.1-TNCS-REQ-57-M].

[CTNC-IFIMV1.1-TNCS-TC-17] For the testing of TNCS for Windows platforms, the test IMVs will populate the well known registry location with optional values and verify that the TNCS ignores unknown optional values correctly and loads without hanging or crashing. This test case is for the following TNCS requirement: [CTNC-IFIMV1.1-TNCS-REQ-53-M].

NOTES:

- There is no test case for [CTNC-IFIMV1.1-TNCS-REQ-25-M] requirement because it is not automatically testable.

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-IMV*, Specification Version 1.1, May 2006.

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