MSE Project Report

A Posture Broker Protocol

Compatible with Trusted Network Connect

Sansar Choinyambuu schoinya@hsr.ch

Advisor: Prof. Dr. Andreas Steffen

January 25, 2011

Hochschule Rapperswil

Oberseestrasse 10

CH-8640 Rapperswil

Table of Contents

Abstract

Network security is only guaranteed when each individual endpoint, connected to the network is safe and secure. It is certainly not a simple task to control whether the network nodes meet the security requirements. One approach to accomplish such control is to check whether the endpoint masters all the security prerequisites before it is admitted to the network. Only those devices which passed the tests are granted the network access. The post-admission controls are eventually carried out when the security policy gets changed or the endpoint posture gets affected.

Network Endpoint Assessment (NEA) is the term which is defined by the NEA working group at IETF for the Network Access Control that is shortly described above. A series of standards have been released by the NEA Working Group, including the RFC 5793 for PB-TNC protocol. As a deliverable of the current semester project, the PB-TNC protocol was implemented as a plugin of the strongSwan software and the present paper serves as a report of the project as well as a documentation for the implementation.

1. Introduction

In today's networked world, it is becoming increasingly important to protect the networks from security threats. The corporate policy defines the security requirements, which have to be fulfilled by each device that is attempting to access the network. With the help of this approach, certain protection level could be enforced on all network nodes. Network Endpoint Assessment architecture has been implemented in the industry to assess the posture of endpoint devices for the purposes of controlling compliance with a corporate security policy and optionally restricting access until the endpoint has been updated to satisfy the requirements.

NEA concept has been intensively researched in recent years, as a consequence both market giants such as Cisco and Microsoft offer their own products and a number of open source solutions are developed. Controversy begins at the point that these products and solutions are developed within various incompatible frameworks and often lacks at the interoperability.

The effort to standardize the technology was initially made by the Trusted Computing Group (TCG), which established a Trusted Network Connect (TNC) working group (TNC-WG) that is publishing specifications for an open architecture of for Network Access Control [5-12].

IETF's NEA Working Group has been organized to agree on and publish series of critical standards for Network Endpoint Assessment. TCG has submitted its TNC specifications as proposals for NEA standards. Consequently, a series of RFC's are released by NEA working group which are compatible with TNC standards. The NEA Reference model defines the components involved in the endpoint assessment and the protocols they employ to communicate with each other. RFC 5793 [2] specifies

PB-TNC, a Posture Broker (PB) protocol identical to the Trusted Computing Group's IF-TNCCS 2.0 protocol [5].

1.1 Objectives

The goal of the present project is to implement the PB protocol [2] as a plugin for the strongSwan software in order the endpoints which installed and appropriately configured the strongSwan be able to take roles of either Access Requestor or Policy Enforcement Point or Policy Decision Point and carry out the Network Endpoint Assessment [1].

1.2 Structure of the report

The research chapter of the paper is dedicated for the study of existing frameworks and solutions for NEA, particularly the open source implementation from TNC@FHH [18] at Fachhochschule Hannover and the libtnc [17], the library used there are taken into consideration. The vendors use various different terminologies for NEA framework entities, components and protocols. In the present paper the terms defined by the IETF RFCs and TNC specifications are mainly used. For a complete list for explanation of terms and abbreviations please refer to the Glossary and Acronyms sections of the paper respectively.

The task of the project is analyzed subsequently in detail and the architecture of the implementation is described followed by the implementation chapter of the report. The implementation part documents the tnccs-20 plugin code of the *strongSwan git branch* **tnccs_20** dating on 30.November.2010. Finally, the experiences we collected as well as the challenges we faced and the outlook into the future are discussed in conclusion.

2. Research

Frost and Sullivan predicted in 2010 that Network Access Control vendors will sell 7,500 appliances and rake in at least $250 million, with a steady growth rate of about 25% every year [23]. The technological hype and financial gains have attracted number of vendors to develop their own NEA solutions.

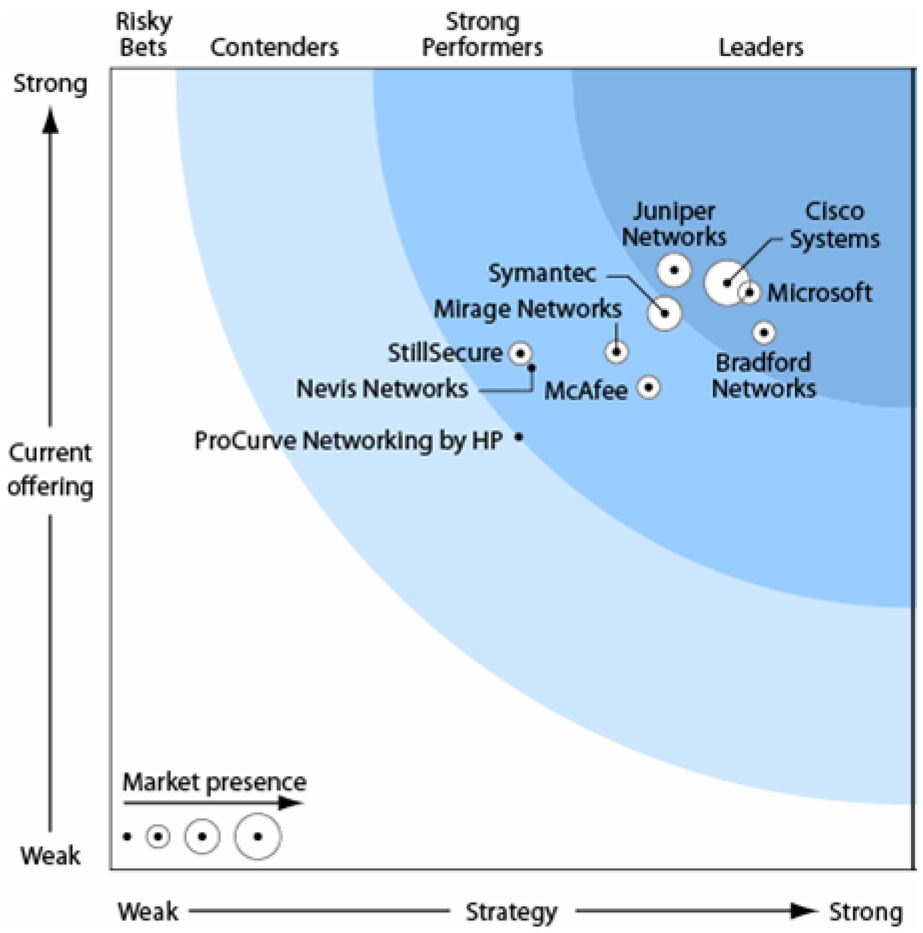


*Figure 1.*The Forrester Wave: Network Access Control, Q3 2008 [25]

Several NEA products are introduced in this section of the paper and more details are given for the TNC@FHH project, an open source NEA implementation and the library used in the project called libtnc.

2.1 *Network Admission Control NAC by Cisco Systems*

The Cisco Network Admission Control System, composed of the Cisco NAC Manager and Server and NAC Agent as client, use the Cisco NAC Appliance to authenticate, authorize, evaluate and remediate wired, wireless and remote users before they can access the network. [27]

Cisco hardware is responsible for enforcing the access policy sent by the Access Control Server. Cisco NAC Appliance extends NAC to all network access methods, including access through LANs, remote-access gateways, and wireless access points. It also supports posture assessment for guest users. According to the preconfigured rules, NAC checks critical windows updates (for Windows 98,XP, 2000,ME), anti-virus software (from most of the big vendors) updates, Anti spyware updates and other third party checks. Cisco claims that customers can easily add customized checks themselves.

The customers who decided to deploy the NAC by Cisco Systems will probably encounter interoperability issues with their already installed network equipments from other vendors. Either all the components which will take part in assessment have to be products by Cisco or they have to be one of the vendors from Cisco's partner program. Cisco and Microsoft both claim that they will be supporting their own partner programs as well as the NAC/NAP program.

2.2 Network Access Protection NAP by Microsoft

NAP enforces health requirements by monitoring and assessing the health of client computers when they attempt to connect or communicate on a network. Client computers that are not in compliance with the health policy can be provided with restricted network access until their configuration is updated and brought into compliance with policy. [25]

Administrator can set polices to restrict or control access to the network based on the health of the computer. Devices that are compliant are allowed access to the network and devices that fail the compliance check are restricted. Non-compliant devices can be automatically updated or specific updates can also be pushed to the device via manual remediation. Compliant computers connected to the network lose connection if they lose health status. Any change of state in the device is reported back to the policy server for post-connect compliance.

Microsoft provides the Windows Security Health Agent (WSHA) with Windows Vista and Windows XP with Service Pack 3 and the Windows Security Health Validator (WSHV) with Windows Server 2008. The WSHA monitors the operational status of the Windows Security Center (WSC) on NAP client computers. The WSHA monitors the Firewall, Virus Protection, Spyware Protection, Automatic Updating and Security Update Protection. There is also number of additional SHA and SHV pairs developed by NAP Partners from Microsoft [28].

2.3 Trusted Network Connect by Trusted Computing Group (TNC by TCG)

The Trusted Computing Group (TCG) is an industry standards body formed to develop, define, and promote open standards for trusted computing and security technologies. TCG has developed an open architecture and standards for Network Access Control called Trusted Network Connect (TNC) [30].

The most important aspects of the TNC architecture are vendor-independence through open standards.

2.3.1 TNC Architecture

Within the three basic entities, namely Access Requestor (AR), Policy Enforcement Point (PEP) and Policy Decision Point (PDP), TNC defines certain components for open framework of Network Access Control. The Access Requestor contains a Network Access Requestor, the software that is used by the client to connect to the network – an 802.1X supplicant, a VPN client, or similar. The Access Requestor also contains a TNC Client (software that manages the overall NAC process) and Integrity Measurement Collectors (IMCs, plugin software modules specialized for reporting the status of AV software, patches, or other things).

The Policy Decision Point contains a Network Access Authority, software that makes the final decision on whether network access should be granted. The Policy Decision Point also contains a TNC Server (software that manages the NAC process on the server) and Integrity Measurement Verifiers (IMVs, plugin software modules that compare reports from IMCs against policy, supply access recommendations to the TNC Server, and send remediation instructions to the IMCs). The Policy Enforcement Point doesn't have any internal components. The following figure illustrates the TNC architecture.

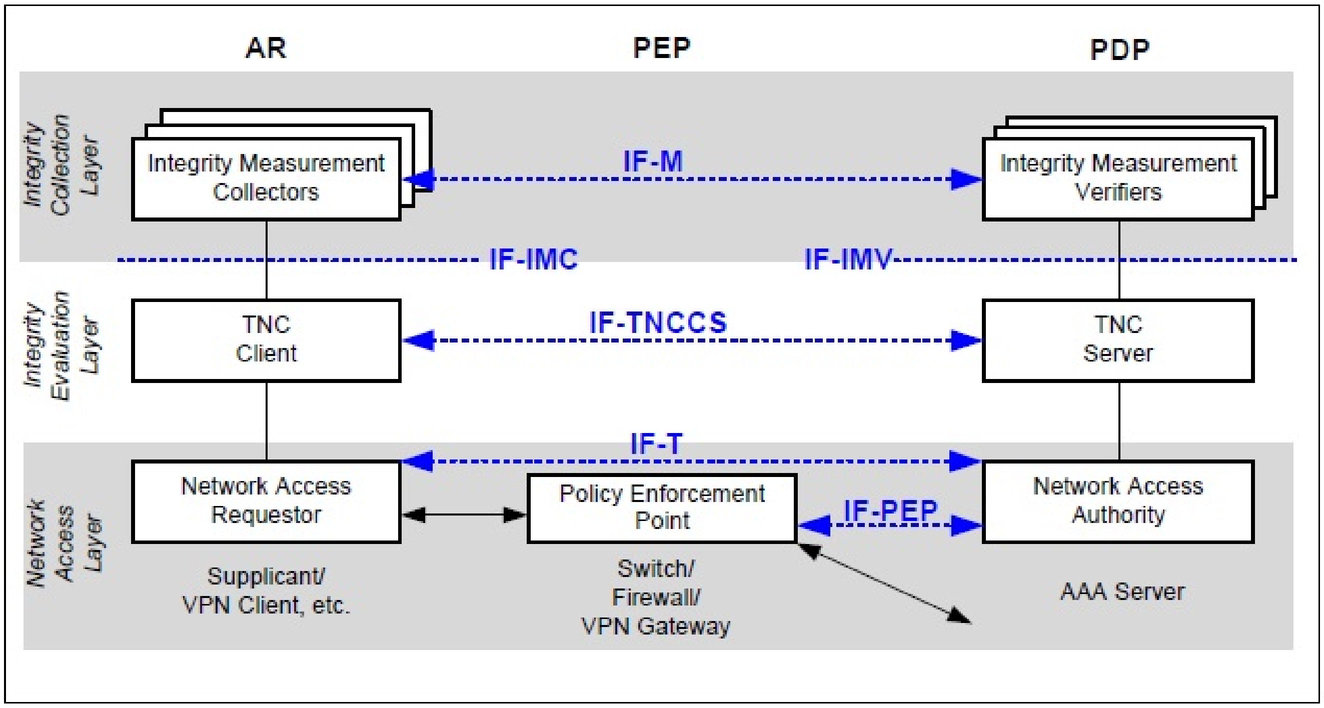

AR
PEP
PDP
Integrity Collection Layer
Integrity Measurement Collectors
IF-M
Integrity Measurement Verifiers
IF-IMC
IF-IMV
Integrity Evaluation Layer
TNC Client
IF-TNCCS
TNC Server
IF-T
Network Access Layer
Network Access Requestor
Policy Enforcement Point
IF-PEP
Network Access Authority
Supplicant/ VPN Client, etc.
Switch/ Firewall/ VPN Gateway
AAA Server

Figure 2.TNC Archtecture [5]

Specifying the interface between the components TNC made it possible to use the hardware from different vendors as separate components and still be able to interoperate with each other seamlessly. Some of the TNC interfaces are protocols and some are Application Programming Interfaces (APIs).

IF-IMC – API that allows a TNC Client to load IMCs and allows IMCs to exchange messages with IMVs

IF-IMV – API that allows a TNC Server to load IMVs and allows IMVs to supply recommendations to the TNC Server and exchange messages with IMCs

IF-M– protocol for messages sent between IMCs and IMVs. This protocol is transported by IF-TNCCS.

IF-TNCCS – protocol for messages sent from TNC Client to TNC Server and vice versa, containing IF-M messages, session management messages, etc. This protocol is transported by IF-T so that it remains transport-independent. IF-TNCCS Version 2.0 [5] protocol which uses TLV binding is identical with PB-TNC protocol specified in RFC 5793 [2]. So this project is the implementation of both IF-TNCCS 2.0 and PB-TNC protocols.

IF-T – the transport protocol. TNC plans to provide several standard options for IF-T. So far, the only method standardized is an EAP method that can be carried over 802.1X or IPsec's IKEv2 protocol.

IF-PEP – protocol for the Policy Decision Point to communicate decisions to the Policy Enforcement Point

2.4 TNC@FHH

The TNC@FHH [18] is a project, maintained by Trust@FHH research group of the Fachhochschule Hannover, the University of Applied Sciences and Arts, in Lower Saxony, Germany. Within scope of this project, the open source implementation of TNC framework was implemented. TNC@FHH implementation was certified by Trusted Computing Group. Within these certification programs, products are tested in order to assess their compliance to the TCG specifications. In the area of TNC, the certification program currently covers the interfaces IF-IMC, IF-IMV and IF-PEP for RADIUS.

The following TNC components and their respective interfaces have been implemented by TNC@FHH:

- IMC's (complies with specification IF-IMC 1.2 and IF-IMV 1.2 [9,10])
- TNCS (complies with IF-TNCCS 1.1 [6]) shared library that is used by the naaeap module
- NAA (complies with IF-T EAP 1.1) naaeap is a shared library that is used by the EAP TNC module.

At the starting phase of the project, the TNC@FHH v0.7.0 was installed and studied to gain initial insights into the NEA concept. Since the goal of the project was to implement Posture Broker protocol, we had no intention to implement new IMV and IMC pairs and used the IMV's and IMC's developed by TNC@FHH for our continual testing during the implementation phase.

FreeRADIUS AAA server software has to be running on the server side as PDP with the patch provided by TNC@FHH applied. The patch is a new EAP module which handles TNC traffic and forwards TNC specific data to corresponding module. The FreeRADIUS can be configured to assign the access requestors to different VLANs depending on the endpoint assessment results provided by IMV's.

Wpa_supplicant [21] or ***XSupplicant*** can be used as 802.1x client which is going to be authenticated by RADIUS server, thus playing the role of AR in TNC reference model. In addition to that an 802.1x compatible switch has to be used as PEP. TNC@FHH provides the documentation on how to install, configure and run the AR and PDP and PEP [29].

The following IMV's and IMC's are implemented by TNC@FHH and also can be used with our implementation. Their Vendor ID is set to FHH IANA PEN 0x0080ab, and each IMV and IMC pair has its own message type.

- example 0xfe: Hello world example
- dummy 0x31:another hello world example, recommended to use for testing purpose
- clamav 0x41:checks the status of the antivirus software clamav
- platid 0x33: simple challenge response protocol, uses RSA private key and X.509 certificate
- attestation 0x34: simple protocol for binary attestation, uses TPM module if available
- hostscanner 0x30: scans for opened ports on an endpoint

Within the scope of this project also a framework called ***imunit*** is developed, with the help of which new IMV and IMC pairs can be developed.

2.4.1 libtnc

The libtnc library is OS independent implementation [17] of the Trusted Network Connect (TNC) specification from Trusted Computing Group (TCG). It contains functions for loading and communicating with TNC plugins, sample IMV and IMC plugins, TNCCS XML support on Windows, Linux, Mac and BSD.

The author of the library is Mike McCauley (mikem@open.com.au). libtnc complies with the TNC specifications IF-IMC 1.2, IF-IMV 1.2 and IF-TNCCS 1.1 [9,10,6].

3. Analysis

This chapter discusses what has been foreseen for the implementation of the project in the beginning phase and what the requirements for the deliverables of the project are.

3.1 Standards

As mentioned in Introduction, the Trusted Computing Group has developed Open architecture and standards for the Trusted Network Connect. First specifications were released in the year of 2007, and open source solutions such as TNC@FHH and libtnc have been implemented in compliance with these TNC specifications.

IETF has seen the insufficiency of the framework standardization for the Network Access Technology and organized the NEA working group with the efforts included from parties which have own strong interest in the subject such as Cisco, Juniper, Microsoft and TCG. In June 2008, the working group has published the overview and requirements document for NEA reference model [1]. The TCG has submitted its TNC specifications as proposals to NEA working group and IETF has approved TCG's IF-M 1.0 and IF-TNCCS 2.0 specifications, subsequently the PA-TNC [3] and PB-TNC [2] protocols were published as RFC's in March 2010.

The goal of the present project is to implement the PB-TNC protocol which is compatible with the TNC framework. Today's stand point for the standards for this implementation is to the certain level interoperable between what is defined by IETF and TGC, that means PB-TNC protocol RFC5793 [2] is identical with the IF-TNCCS 2.0. IF-TNCCS protocol is specified concisely and refers to the RFC5793 of PB-TNC protocol for most of the details. We predict that the Network Access Control products and solutions from vendors would have to respect the RFC's from IETF in order to solve the interoperability issues in the coming years.

3.2 Requirements

The main requirement for the project deliverable is that the implementation of PB-TNC protocol complies with the RFC 5793. Since, at the moment of this writing, there is no open source or commercial implementation of RF5793, we had an advantage to make our own design decisions at our convenience.

The common requirements C1-C11, that are defined by NEA Requirement document [1] and PB specific requirements PB1-PB6 have to be fulfilled by the implementation.

3.3 Deliverables

The implementation has to be fully functional strongSwan plugin which implements the RFC 5793 PB-TNC protocol. When the appropriate plugins are activated with the correct configuration, the strongSwan VPN client can act as NEA Client and strongSwan VPN gateway can take the role of either PEP only or PEP integrated within NEA server alternatively.

4. Architecture

This chapter of the project report will explain the architecture concepts based on the brief introduction of strongSwan software followed by the description of NEA reference model and illustration of NEA Protocol stack within strongSwan.

4.1 strongSwan

strongSwan is an IPSec based open source VPN solution for Linux, with the focus on X.509 certificates. It is a descendent of discontinued FreeS/WAN project which was the first complete implementation for IPSec for Linux. Besides the Openswan, which is another continuation of FreeS/WAN project, strongSwan is one of the most important IPSec solutions for Linux.

The strongSwan software is modular and offers dozens of plugins which enhance the functionality. The current project should deliver one further plugin **tnccs-20** which activated together with other appropriate plugins gives the software the ability of carrying out NEA on endpoints running strongSwan. The **tnccs-20** plugin is the part of the Charon daemon which implements IKEv2.

The tnccs-20 plugin is written with an object oriented programming style for C. This allows the modern programming paradigms to be employed but still use the standard c complier and toolset. This object oriented design is achieved by heavy use of function pointers. Interfaces of classes are defined as a struct with function pointers, and an implementation extends the struct by including the Interface as its first struct member [31].

4.2 NEA Reference model

Identical with TNC Architecture in the Figure 2, NEA reference model contains 3 layered protocols.

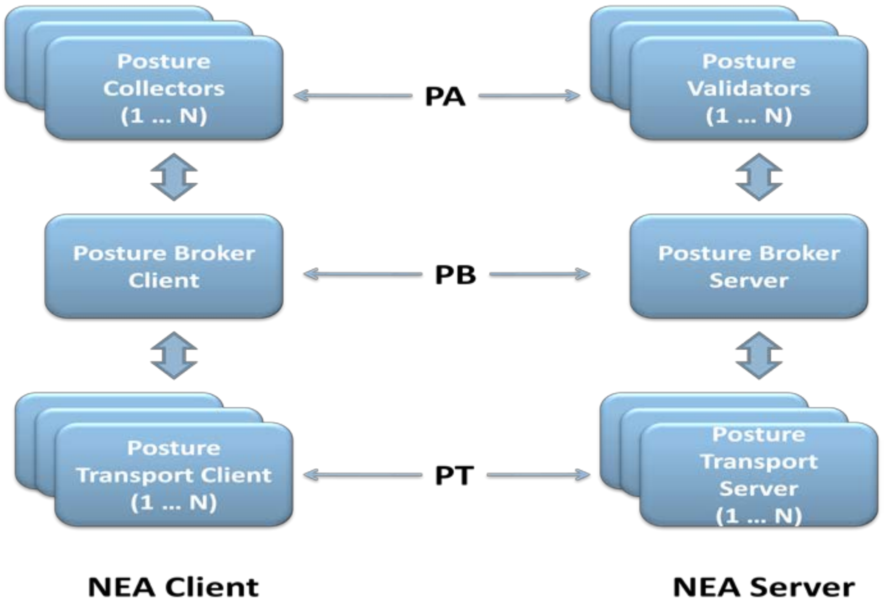

Posture Collectors (1 … N)
PA
Posture Validators (1 … N)
Posture Broker Client
PB
Posture Broker Server
Posture Transport Client (1 … N)
PT
Posture Transport Server (1 … N)
NEA Client
NEA Server

Figure 3. NEA Reference Model [1]

PA protocol is a message wrapper around a set of attributes that is batched and encapsulated in PB. PB is primarily a lightweight message batching protocol and the PT has the responsibility to transport the batches between the NEA Client and NEA Server. The vertical lines in the model represent APIs and/or protocols between components within the NEA Client or Server.

4.2.1 NEA Client

The NEA Client has the responsibility to respond to the requests for attributes describing the present state of the local operating domain (e.g. OS, Anti Virus software) and handling the assessment results sent by the server. In case it receives the remediation instruction from the server, the NEA client has to execute the instructions in order to become compliant with the policy. A single NEA Client can have several Posture Collectors capable of collecting standard and/or vendor-specific Posture Attributes for particular features of the endpoint.

Posture Broker Client multiplexes the PA messages which contain the posture attributes, into the batch. It also demultiplexes the batch sent by the Posture Broker Server and distributes encapsulated PA messages to the corresponding collectors. Handling and eventually interacting with the user upon the global assessment decision from the PB Server is one of the main functionalities of PB Client.

4.2.2 NEA Server

NEA Server is responsible for handling the posture attributes received on request from NEA Client. To be precise, the attributes are first extracted from PB Batch by the Posture Broker Server and are distributed to the responsible Posture Validators. PB Server is then in charge for computing the global assessment decision based on individual posture assessment results from the each individual Posture Validator.

4.3 Protocol stack

The protocols and components which are stacked in our implementation for NEA assessment within strongSwan are depicted in the Figure 4.

IMC and IMV pairs developed by TNC@FHH are used in our implementation. Particularly, all the Collectors and Validators which respect the specification IF-IMC v1.2 and IF-IMV v1.2 by TCG can be used with the strongSwan for the endpoint assessment.

PA-TNC [3] protocol is claimed to be identical with TNC specified IF-M v1 protocol. It has not been however in the scope of the project to implement this protocol. The IF-IMC and IF-IMV specifications were followed to enable the communication with IMV’s and IMV’s. The implementation of the PB-TNC protocol, tnccs-20 strongSwan plugin complies with the RFC 5793 [2].

The TNC architecture doesn't define any particular protocol to be used as IF-T transport protocol. IF-T Protocol Bindings for Tunneled EAP Methods 1.1 [12] provides the specification of protocol bindings when tunneled EAP method is used. While the cryptographically protected outer tunnel is responsible for secure communication, the non-secure inner protocol can exchange the messages within the created tunnel. EAP-TNC inner protocol was implemented by Dr.Andreas Steffen as well as outer transport protocol IKEv2-EAP-TTLS, which provides the authenticated and encrypted tunnel.

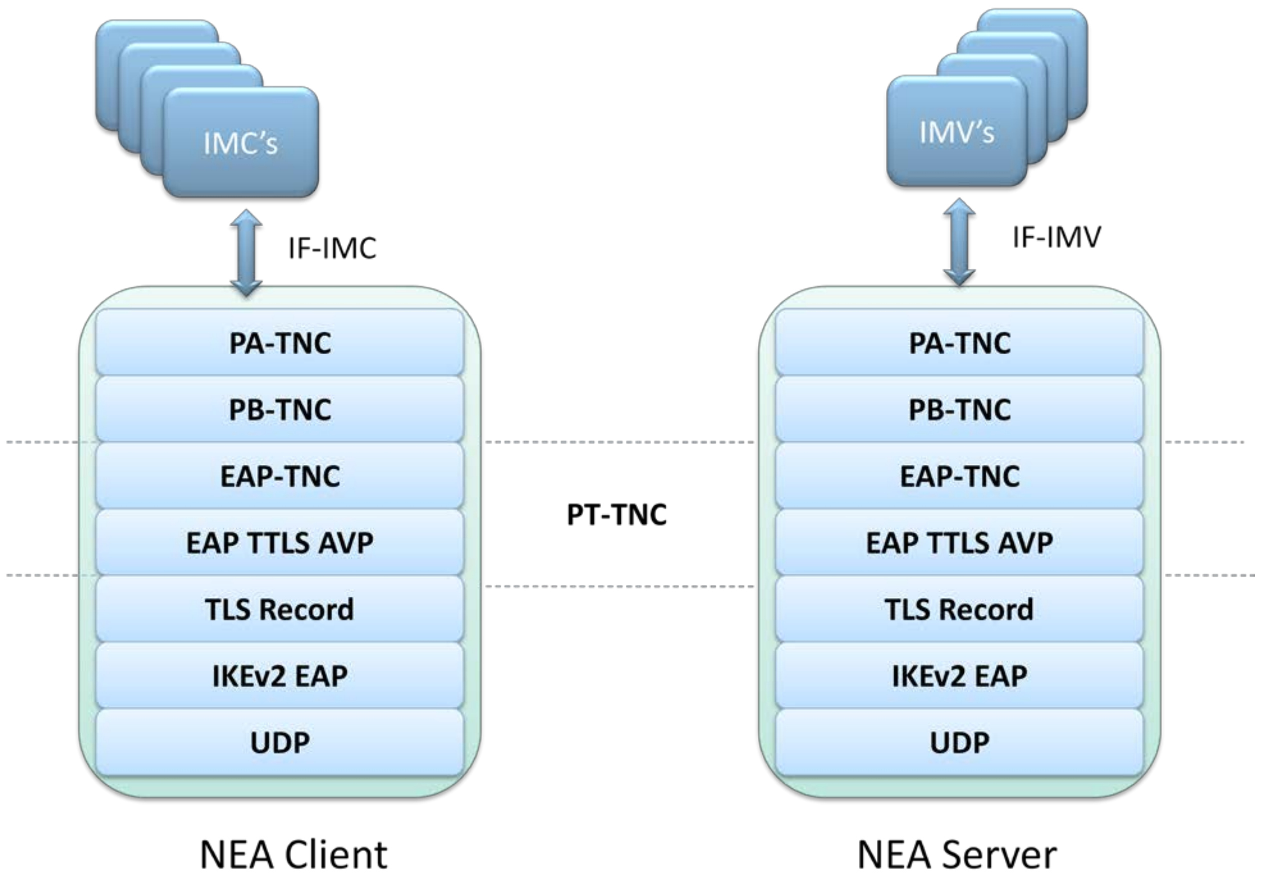


Figure 4. NEA Protocol Stack within strongSwan

The tunneled EAP bindings make it possible to implement IF-T transport protocol over number of existing access protocols that use EAP at the access level. One of such access protocol, 802.1x was used in the various other implementations of TNC framework such as TNC@FHH while IKEv2 was used in our implementation.

5 Implementation

The initial plan for the implementation of the project was to use libtnc [35] library to communicate with the IMC's and IMV's and enhance the strongSwan with the plugin which implements the IF-TNCCS interface functionalities according to the RFC 5793.

However during the implementation phase it became apparent that it was to our benefit if we implement the API's to facilitate the interaction with IMV's and IMC's as strongSwan plugin as well. Therefore the API's specified in IF-IMC and IF-IMV version 1.2 [9][10] are implemented by Dr.Andreas Steffen for the strongSwan.

Since the IETF hasn't yet released the internet draft on PT protocol as RFC at the time of the writing, the outer and inner transport protocols are implemented as defined in IF-T Protocol Bindings for Tunneled EAP Methods version 1.1 [12]. The inner non-secure transport protocol EAP-TNC is encapsulated in an outer encrypted and authenticated IKEv2-EAP-TTLS tunnel.

The PB-TNC protocol [2] introduces the new TLV based batches, which are exchanged between the NEA Client and NEA server. In the previous version of specification from TCG, TNCCS v1.1, the batches exchanged between PB Broker Server and Client was XML based. Therefore the handling of the PB batches as defined in PB-TNC protocol differs mostly if not completely from the implementations of the TNCCS v1.1 protocol in terms of encoding. The Server side has to implement both versions of the protocol whenever possible, so that any client can be assessed regardless of the IF-TNCCS interface protocol version in use. IF-TNCCS 1.1 support was introduced in October 2010 with the strongSwan 4.5.0 stable release. The tnccs-11 Charon plugin uses Mike McCauley's libtnc library [35].

5.1 PB-TNC Encapsulation

An example PB-TNC encapsulation of the messages exchanged between Posture Broker Client and Posture Broker is illustrated in the Figure 5. For instance as the figure shows, several PA messages from or to IMV's and IMC's are multiplexed into the PB batch which is preceded by the Batch Header.

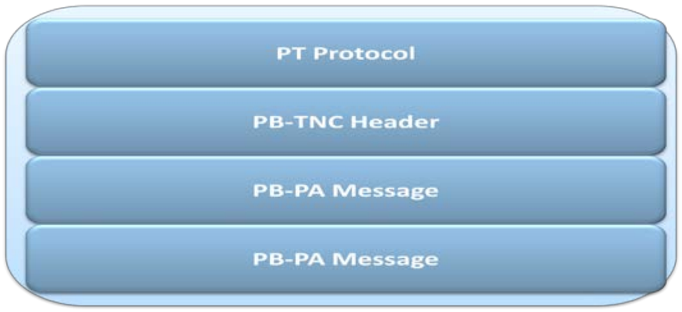


Figure 5. PB-TNC Encapsulation [2]

The PB batch is transported through the secure tunnel provided by PT protocol to the communicating endpoint. The PB batch may not only contain the PA messages but number of other messages which are built and handled within the PB-TNC protocol.

5.2 PB Batch and Messages

For the detailed explanation of fields for messages and batches please refer to the RFC 5793 and RFC 5792 [2][3]. This section demonstrates how the PB batch and messages were implemented for the tnccs-20 plugin.

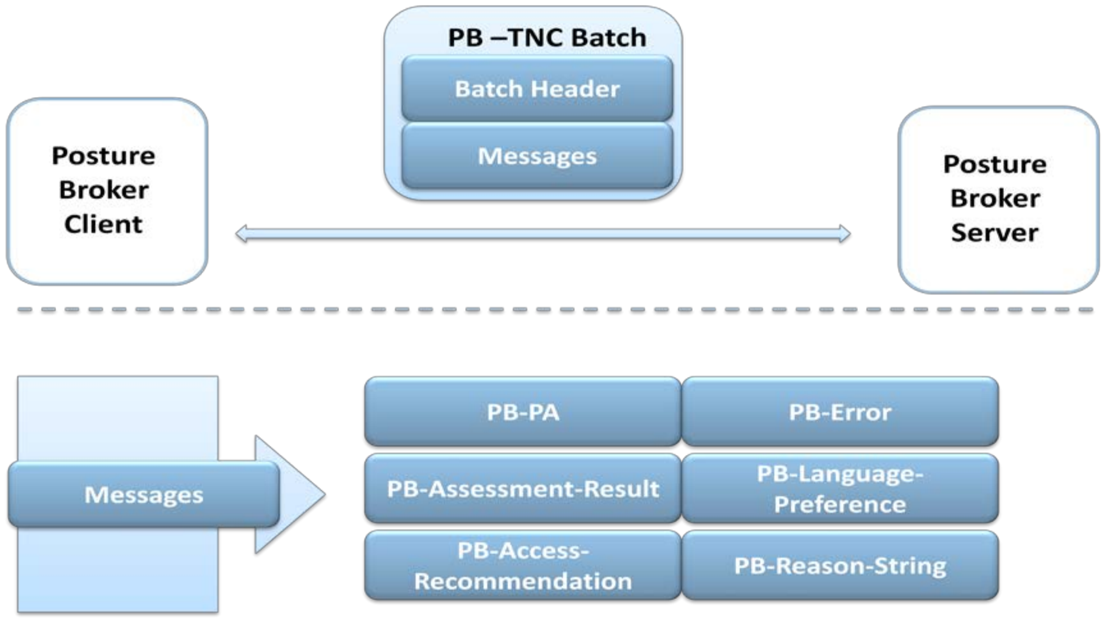


Figure 6. Batch exchange between PB Server and Client

The constructing and processing of the batches and the messages have been implemented using TLS Writer and TLS Reader which is the part of the tlslib of strongSwan, developed by Martin Willi.

5.2.1 Vendor ID

In our implementation, the Vendor ID field for all the message types, message subtypes and error codes are set to the value of zero (“0”) which is IETF SMI PEN (Private Enterprise Number). Hence, the message types and subtypes and error codes have the meaning of the standardized values by RFC’s [1][2][3].

```
#define PB_TNC_IETF_VENDOR_ID       0x000000
```

5.2.2 PB Batch Header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Version    |D|     Reserved                        | B-Type|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Batch Length                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7. PB-TNC Batch Header [2]

The PB batches serve as transport means for the entire communication between the PB Server and PB Client. Each batch has to be preceded by the header, shown in the Figure 7. Version field of the PB Batch Header always has the decimal value of “2” in our implementation, which stands for the TNCCS protocol version number 2. It is foreseen in the standard [2] that by looking at this field one could determine which version of TNCCS protocol is going to be employed for certain assessment session.

Directionality bit defines whether the batch is sent from Server to Client or the opposite course. The following Batch Types are defined in the specification RFC 5793[2]:

CDATA, SDATA – May only be sent from Client or Server respectively, usually contains the PB-PA messages for IMV or IMC. Non fatal error messages might also be delivered within the batch of this type.

CRETRY, SRETRY – Can only be sent from the Client or Server respectively. In our implementation, only the client side is able to trigger the handshake retry, thus the CRETRY batch is replied with SRETRY batch and the handshake retry action is taken thereupon at the Client side. These batches contain no messages in our implementation.

RESULT – Can be sent from Server side only. In our implementation it always contains the PB-Assessment-Result, PB-Access-Recommendation messages and additionally PB-Result-String message if available.

CLOSE – Can be sent from both sides, it should contain the Fatal PB-Error message if the connection is to be closed down due to the fatal error. Otherwise it could also be empty in case the NEA was done successfully and the connection is to be closed down.

Batch length field is assigned the value of the whole batch length in bytes which includes the header and messages. When the new batch is constructed, we write the default value of 8 in this field which is the length of the header solely. Each time when the new message is appended to the batch, the length is calculated again including the length of new message and Batch Length field is rewritten as implemented in the following piece of code.

```
htoun32(header.ptr + 4,header.len + pb_tnc_msg.len);
```

The creation of the new batch header is done within the function:

```
static chunk_t build_batch_header(pb_tnc_batch_type_t batch_type,
                                            bool is_server);
```

The processing of the header of received batch is implemented within the function:

```
static status_t process_batch(private_tnccs_20_t *this, void*  batch_buffer,
                                 size_t batch_buffer_length);
```

5.2.3 PB-TNC Message

```
   0                   1                   2                   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |     Flags     |               PB-TNC Vendor ID                |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                       PB-TNC Message Type                     |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                      PB-TNC Message Length                    |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |               PB-TNC Message Value (Variable Length)          |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 8. PB-TNC message [2]

Regardless of the message types, every PB-TNC message has the same structure which is depicted in the Figure 8. The interface ***pb_tnc_message_t*** was created based on this general structured feature. This generic interface is implemented by all the PB-TNC message types. Thus giving the opportunity to create (***build*** function), process (***process*** function) and destroy the message (***destroy*** function) as well as to get type of message (***get_type*** function) using the generic object of type ***pb_tnc_message_t***, independent of the message types.

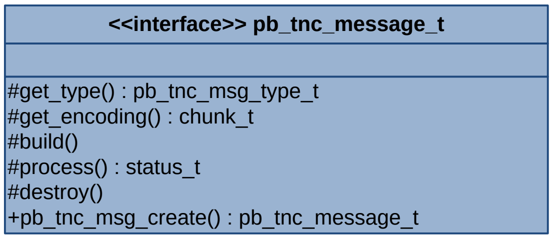

<<interface>> pb_tnc_message_t
#get_type() : pb_tnc_msg_type_t
#get_encoding() : chunk_t
#build()
#process() : status_t
#destroy()
+pb_tnc_msg_create() : pb_tnc_message_t

Figure 9. Interface pb_tnc_message_t

The variable field Message Value contains the actual different PB-TNC messages. This value of the field can be retrieved with the ***get_encoding*** function of the ***pb_tnc_message_t*** generic object. The preceding fields remain constant for all the message types, hence can be called as the header of the message.

The only defined bit of the flag field is most significant field for No-Skip flag. Depending on the message type this flag is either cleared or set to 1. Each message type has its own structure for the variable field PB-TNC Message Value, which is going to be explained in the following sections.

The creation of the PB-TNC message is done within the function:

```
static chunk_t build_pb_tnc_msg(pb_tnc_msg_type_t msg_type, chunk_t msg_value);
```

The processing of the single PB-TNC message is implemented within the function:

```
static status_t process_pb_tnc_msg(tls_reader_t *reader,
                                   bool *vendor_id_reserved,
                                   bool *message_type_reserved,
                                   bool *no_skip,
                                   u_int32_t *message_type,
                                   pb_tnc_message_t **pb_tnc_msg);
```

5.2.4 PB-PA Message

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Flags      |               PA Message Vendor ID            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           PA Subtype                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Posture Collector Identifier | Posture Validator Identifier  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 PA Message Body (Variable Length)             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 10. PB-PA Message [2]

PB-PA message construction is triggered either from either the IMV or IMC, it contains various attribute requests and responses to those requests and has the message type value of "1". PB-PA message is usually contained in the CDATA or SDATA batch respectively, based on its directionality. The IMC or IMV calls the IF-IMC or IF-IMV API function SendMessage in order to give a message to PB Server or PB Client for delivery.

The SendMessage function definition in IF-IMC API [10]:

```
TNC_Result TNC_TNCC_SendMessage( /*in*/ TNC_IMCID imcID,
                                    /*in*/ TNC_ConnectionID connectionID,
                                    /*in*/ TNC_BufferReference message,
                                    /*in*/ TNC_UInt32 messageLength,
                                    /*in*/ TNC_MessageType messageType);
```

The SendMessage function definition in IF-IMV API [9]:

```
TNC_Result TNC_TNCS_SendMessage(/*in*/ TNC_IMVID imvID,
                                    /*in*/ TNC_ConnectionID connectionID,
                                    /*in*/ TNC_BufferReference message,
                                    /*in*/ TNC_UInt32 messageLength,
                                    /*in*/ TNC_MessageType messageType);
```

The SendMessage implementation of tnccs-20 plugin:

```
METHOD(tnccs_t, send_message, void, private_tnccs_20_t* this,
                                    TNC_IMCID imc_id, TNC_IMVID imv_id,
                                    TNC_BufferReference msg,
                                    TNC_UInt32 msg_len,
                                    TNC_MessageType msg_type)
```

On the receiving side the PB Server or PB Client has the responsibility to distribute the PA messages to the corresponding IMV or IMC by calling IF-IMV or IF-IMC API function ReceiveMessage.

The ReceiveMessage function definition in IF-IMV API [9]:

```
TNC_Result TNC_IMV_ReceiveMessage( /*in*/ TNC_IMVID imvID,
                                    /*in*/ TNC_ConnectionID connectionID,
                                    /*in*/ TNC_BufferReference message,
                                    /*in*/ TNC_UInt32 messageLength,
                                    /*in*/ TNC_MessageType messageType);
```

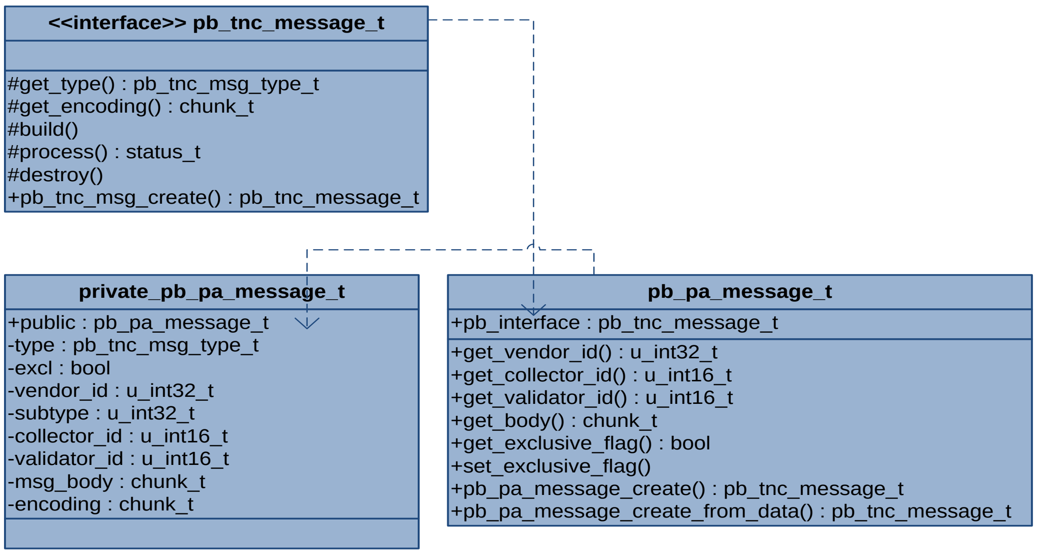
The ReceiveMessage function definition in IF-IMC API [10]:

```
TNC_Result TNC_IMC_ReceiveMessage(/*in*/ TNC_IMCID imcID,
                                    /*in*/ TNC_ConnectionID connectionID,
                                    /*in*/ TNC_BufferReference message,
                                    /*in*/ TNC_UInt32 messageLength,
                                    /*in*/ TNC_MessageType messageType);
```

The receive message function call implemented in tnccs-20 plugin:

```
charon->imvs->receive_message(charon->imvs,
                                    this->connection_id, msg_body.ptr,
                                    msg_body.len, msg_type);
charon->imcs->receive_message(charon->imcs,
                                    this->connection_id, msg_body.ptr,
                                    msg_body.len,msg_type);
```

Each IMV and IMC has to register the message types they are interested in receiving when they are loaded during the strongSwan startup (tnc-imc and tnc-imv plugins). With the help of msg_type variable, the imv or imc manager decides which imv's or imc's ReceiveMessage function to call.



<<interface>> pb_tnc_message_t
#get_type() : pb_tnc_msg_type_t
#get_encoding() : chunk_t
#build()
#process() : status_t
#destroy()
+pb_tnc_msg_create() : pb_tnc_message_t
private_pb_pa_message_t
+public : pb_pa_message_t
-type : pb_tnc_msg_type_t
-excl : bool
-vendor_id : u_int32_t
-subtype : u_int32_t
-collector_id : u_int16_t
-validator_id : u_int16_t
-msg_body : chunk_t
-encoding : chunk_t
pb_pa_message_t
+pb_interface : pb_tnc_message_t
+get_vendor_id() : u_int32_t
+get_collector_id() : u_int16_t
+get_validator_id() : u_int16_t
+get_body() : chunk_t
+get_exclusive_flag() : bool
+set_exclusive_flag()
+pb_pa_message_create() : pb_tnc_message_t
+pb_pa_message_create_from_data() : pb_tnc_message_t

Figure 11. Class diagram for PB-PA Message

As could be observed in Figure 11, ***pb_pa_message_t*** class has implemented the generic PB-TNC message interface ***pb_tnc_message_t***. There is one private member variable for each field of the PB-PA message, and those private members of ***private_pb_pa_message_t*** class could only be accessed through the public function pointers of ***pb_pa_message_t***.

The ***process*** function defined by PB-TNC message generic interface sets the values to the private members for the PB-PA message fields. In this way, it is made possible to use the getter functions of ***pb_pa_message_t*** class for the handling of the field values in the received message. The similar structure applies for all the type of messages and is not explained in the following sections repeatedly.

For the creation of PB-PA message, the function ***pb_pa_message_create*** is used with the necessary parameters, which are supplied with the SendMessage function call from either IMV or IMC.

```
pb_tnc_message_t *pb_pa_message_create(u_int32_t vendor_id, u_int32_t subtype,
                                       u_int16_t collector_id,
                                       u_int16_t validator_id,
                                       chunk_t msg_body)
```

The ***pb_pa_message_create_from_data*** function takes the message value field of the PB-TNC message as a whole chunk in the argument and assigns the chunk value to private member called *encoding*.

```
pb_tnc_message_t* pb_pa_message_create_from_data(chunk_t data);
```

The Exclusive flag which is the most significant bit in the flag field is cleared as default, due to the fact that IF-IMC and IF-IMV [9][10] specifications do not define the feature to deliver the PA message to a certain IMV or IMC exclusively. Every IMV and IMC which reported the interest in receiving the message with certain PA subtype, would be provided with the PA message with the help of ReceiveMessage API function call, introduced earlier in this section. Nevertheless, the setter function for Exclusive flag is implemented and can be used in future when the IF-IMC and IF-IMV API's provide the corresponding functionality.

5.2.5 PB-Assessment-Result Message

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Assessment Result                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 12. PB-Assessment-Result Message [2]

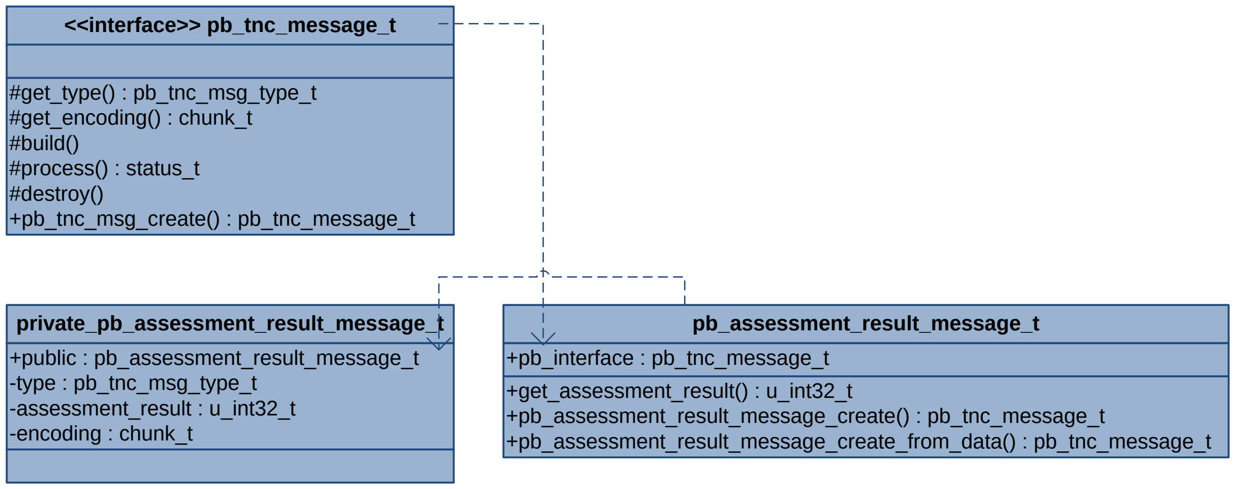
PB-Assessment-Result message could only be sent from PB server, after the global assessment decision was made. It has very simple structure and only contains the Assessment Result code. The message type value for PB-Assessment-Result message is “2”. The term “Assessment” is called “Evaluation” in the TNC framework as it could be read in Platform Binding C Header file [14], which was used in our implementation, defines the codes for assessment.

```
/* IMV Evaluation Result Values */

#define TNC_IMV_EVALUATION_RESULT_COMPLIANT 0
#define TNC_IMV_EVALUATION_RESULT_NONCOMPLIANT_MINOR 1
#define TNC_IMV_EVALUATION_RESULT_NONCOMPLIANT_MAJOR 2
#define TNC_IMV_EVALUATION_RESULT_ERROR 3
#define TNC_IMV_EVALUATION_RESULT_DONT_KNOW 4
```

PB-Assessment-Result message has to be contained in the RESULT batch together with the PB-Access-Recommendation message and if available the PB-Reason-String message as well. As described in the RFC 5793 [2], it is up to the developer how to handle the received Assessment result. In our implementation, the received Assessment result is logged on the strongSwan Debug level 1.

The following Figure demonstrates the class diagram of PB-Assessment-Result message. As described in the previous PB-PA message section, the main structures for all type of message implementations are identical. During the processing of the received PB-Assessment-Result message, the ***get_assessment_result*** function is used to retrieve the value for Assessment Result Code field.



<<interface>> pb_tnc_message_t
#get_type() : pb_tnc_msg_type_t
#get_encoding() : chunk_t
#build()
#process() : status_t
#destroy()
+pb_tnc_msg_create() : pb_tnc_message_t
private_pb_assessment_result_message_t
+public : pb_assessment_result_message_t
-type : pb_tnc_msg_type_t
-assessment_result : u_int32_t
-encoding : chunk_t
pb_assessment_result_message_t
+pb_interface : pb_tnc_message_t
+get_assessment_result() : u_int32_t
+pb_assessment_result_message_create() : pb_tnc_message_t
+pb_assessment_result_message_create_from_data() : pb_tnc_message_t

Figure 13. Class Diagram for PB-Assessment-Result message

5.2.6 PB-Access-Recommendation Message

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Reserved           |   Access Recommendation Code  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

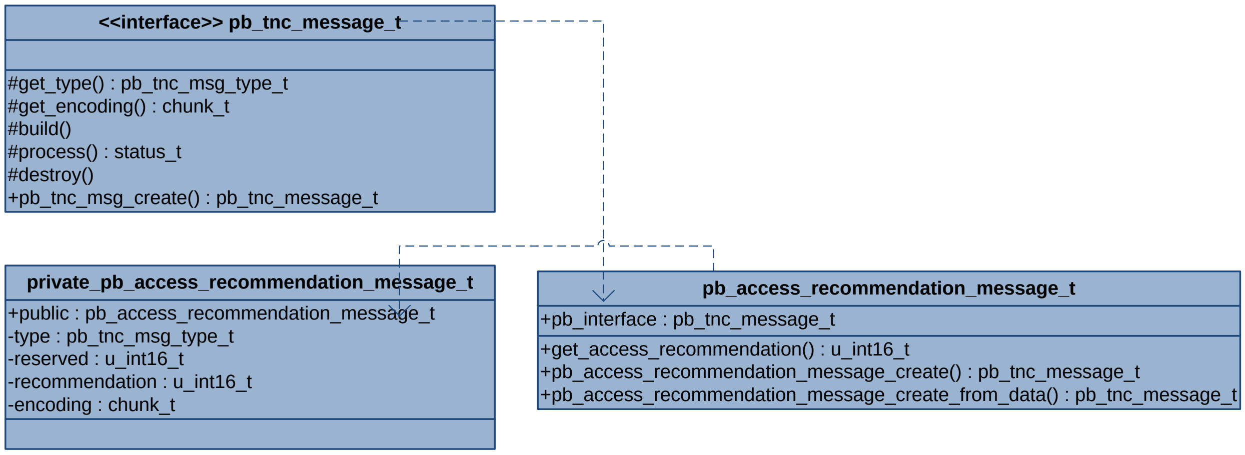
Figure 14. PB-Access-Recommendation message [2]

PB-Access-Recommendation message is sent from PB server side only, after the global assessment decision was made. The structure of the message is incomplex as it simply contains the reserved field filled with zero and Access Recommendation code. The message type value for PB-Access-Recommendation message is “3”. The Platform Binding C Header file [14], which was used in our implementation, defines the codes for access recommendation.

```
/* IMV Action Recommendation Values */

#define TNC_IMV_ACTION_RECOMMENDATION_ALLOW 0
#define TNC_IMV_ACTION_RECOMMENDATION_NO_ACCESS 1
#define TNC_IMV_ACTION_RECOMMENDATION_ISOLATE 2
#define TNC_IMV_ACTION_RECOMMENDATION_NO_RECOMMENDATION 3
```

As defined in the RFC 5793 [2] the PB-Access-Recommendation message has to be contained in RESULT batch whenever possible, the requirement which our implementation respects.



<<interface>> pb_tnc_message_t
#get_type() : pb_tnc_msg_type_t
#get_encoding() : chunk_t
#build()
#process() : status_t
#destroy()
+pb_tnc_msg_create() : pb_tnc_message_t
private_pb_access_recommendation_message_t
+public : pb_access_recommendation_message_t
-type : pb_tnc_msg_type_t
-reserved : u_int16_t
-recommendation : u_int16_t
-encoding : chunk_t
pb_access_recommendation_message_t
+pb_interface : pb_tnc_message_t
+get_access_recommendation() : u_int16_t
+pb_access_recommendation_message_create() : pb_tnc_message_t
+pb_access_recommendation_message_create_from_data() : pb_tnc_message_t

Figure 15. Class Diagram for PB-Access-Recommendation message

Similar to the other message types, PB-Access-Recommendation message implementation is structured in the way shown above. The handling of the received access recommendation is also the decision of PB client. The received access recommendation is logged on strongSwan debug level 1.

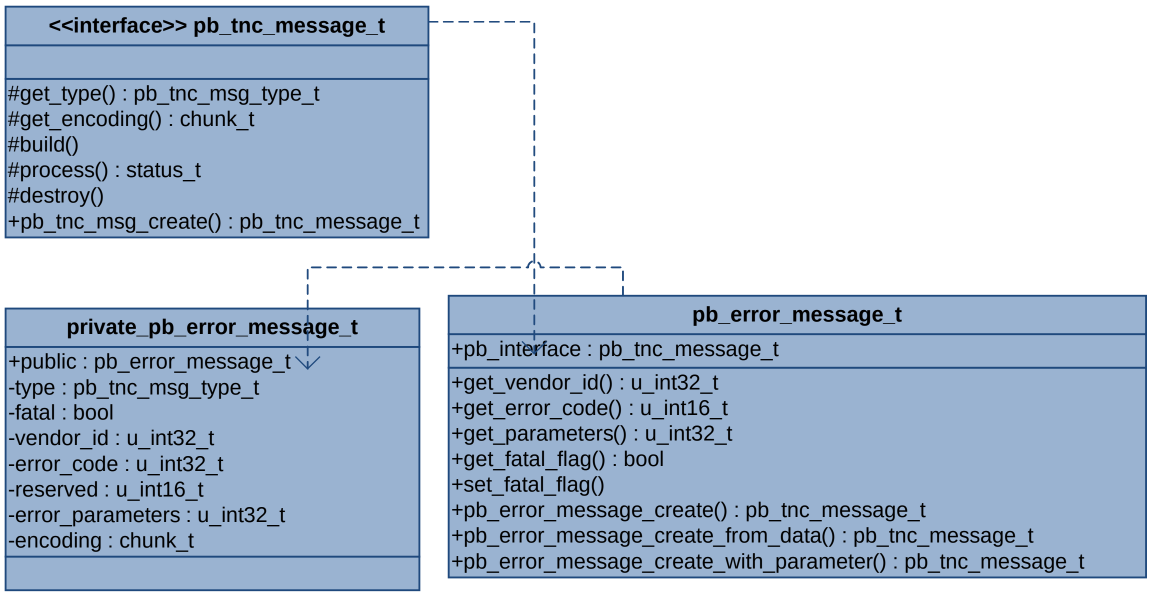
5.2.7 PB-Error Message

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |            Error Code Vendor ID               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Error Code          |           Reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Error Parameters (Variable Length)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 16. PB-Error message [2]

There are number of situations which are defined in RFC 5793 that certain error message with specified error code has to be constructed and sent. The description of these circumstances and which error message is created thereupon are explained in the section Error Handling. PB-Error message has the message type value of "5".

Only the use of most significant bit in Flags field is defined in the standard, which presents the Fatal error if set to decimal value of “1”.The PB-Error message can be enclosed in CLOSE batch if the Error is fatal and in any type of batch if not. The RFC 5793 defines that the all of the Error codes are actually fatal with the exception of ***INVALID PARAMETER*** error, where the developer can decide whether to clear the fatal flag depending on the situation. The value for Error Parameters field varies depending on the Error Code and there is specified structure for this field for every Error Code defined by RFC 5793 [2].



<<interface>> pb_tnc_message_t
#get_type() : pb_tnc_msg_type_t
#get_encoding() : chunk_t
#build()
#process() : status_t
#destroy()
+pb_tnc_msg_create() : pb_tnc_message_t
private_pb_error_message_t
+public : pb_error_message_t
-type : pb_tnc_msg_type_t
-fatal : bool
-vendor_id : u_int32_t
-error_code : u_int32_t
-reserved : u_int16_t
-error_parameters : u_int32_t
-encoding : chunk_t
pb_error_message_t
+pb_interface : pb_tnc_message_t
+get_vendor_id() : u_int32_t
+get_error_code() : u_int16_t
+get_parameters() : u_int32_t
+get_fatal_flag() : bool
+set_fatal_flag()
+pb_error_message_create() : pb_tnc_message_t
+pb_error_message_create_from_data() : pb_tnc_message_t
+pb_error_message_create_with_parameter() : pb_tnc_message_t

Figure 17. Class Diagram for PB-Error message

There is one additional method, called ***pb_error_message_create_with_parameter*** for creating the PB-Error message with the parameter.

5.2.8 PB-Language-Preference Message

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Language Preference (Variable Length)           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 18. PB-Language-Preference message [2]

PB-Language-Preference message (with message type “6”) could be sent from both PB Client and PB Server, but as mentioned in the RFC 5793, it makes more sense if the Language preference of the Client is sent to the server. In this way the IMV’s and PB Server can adjust the language of the plain strings they’re sending to the IMC’s and PB Client accordingly. For this reason, the first CDATA batch from the PB Client contains the PB-Language-Preference message in our implementation.

The Language Preference could be set as the part of the strongSwan configuration in the format defined in RFC 4646. The default preferred language is set to English, if no language is configured. For the PB-Language-Preference message to be sent to PB Server, the prefix "`Accept-Language: `" is prepended to the configured setting string in accordance with the RFC 5793.

On the PB Server side the received language preference value is assigned to the attribute ***TNC_ATTRIBUTEID_REASON_LANGUAGE*** via function call:

```
this->recs->set_preferred_language(this->recs, language);
```

On the account of which it is possible for IMV's to retrieve the value of this attribute for adjusting their reason strings' language accordingly when they provide access recommendations.

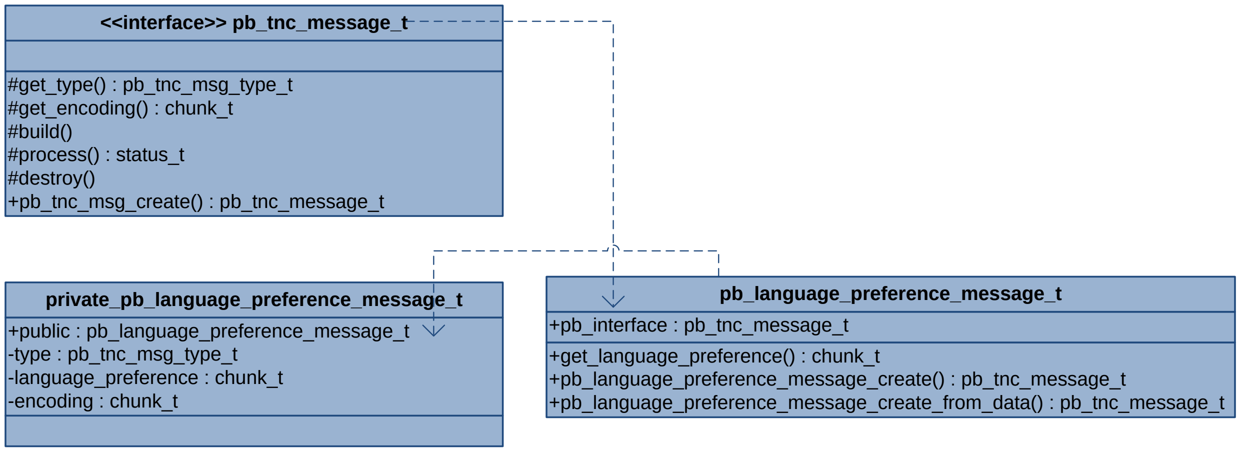

<<interface>> pb_tnc_message_t
#get_type() : pb_tnc_msg_type_t
#get_encoding() : chunk_t
#build()
#process() : status_t
#destroy()
+pb_tnc_msg_create() : pb_tnc_message_t
private_pb_language_preference_message_t
+public : pb_language_preference_message_t
-type : pb_tnc_msg_type_t
-language_preference : chunk_t
-encoding : chunk_t
pb_language_preference_message_t
+pb_interface : pb_tnc_message_t
+get_language_preference() : chunk_t
+pb_language_preference_message_create() : pb_tnc_message_t
+pb_language_preference_message_create_from_data() : pb_tnc_message_t

Figure 19. Class Diagram for PB-Language-Preference message

5.2.9 PB-Reason-String Message

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Reason String Length                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Reason String (Variable Length)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Lang Code Len | Reason String Language Code (Variable Length) |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 20. PB-Reason-String message [2]

The PB message with the message type value of "7" is PB-Reason-String message. It could only be sent from the PB Server side and it should contain the text, which explains the reasoning of the access recommendation from an IMV. The Language code field helps to send the same reason string texts in different languages so that Client can select what it understands and can handle it accordingly.

This occasion to send several PB-Reason-String messages together is applied in different concept for our implementation. The argumentation is that there should be reason for every recommendation made by the IMV. We can't possibly aggregate those reasons for the global access recommendation decision solely based on the contents. Therefore we are collecting the reason strings from each IMV if they provide any, create separate PB-Reason-String messages and append them together. As suggested in the RFC 5793, those reason strings are then sent together with the PB-Assessment-Result and PB-Access-Recommendation messages in the RESULT batch. The providing of the reason string and reason language has to be done when the IMV makes the Recommendation with the help of the following function defined in the IF-IMV API [9] as implemented in ***tnccs_manager_t class***:

```
METHOD(tnccs_manager_t, set_attribute, TNC_Result,
                 private_tnccs_manager_t *this, TNC_IMVID imv_id,
                 TNC_ConnectionID id,
                 TNC_AttributeID attribute_id,
                 TNC_UInt32 buffer_len,
                 TNC_BufferReference buffer);
```

This function calls the following functions of ***recommendations_t*** class which is the part of ***tnc-imv*** plugin to set the reason string and reason language to the recommendation:

```
METHOD(recommendations_t, set_reason_string, TNC_Result,
      private_tnc_imv_recommendations_t *this, TNC_IMVID id, chunk_t reason);
METHOD(recommendations_t, set_reason_language, TNC_Result,
      private_tnc_imv_recommendations_t *this, TNC_IMVID id,
                                          chunk_t reason_lang);
```

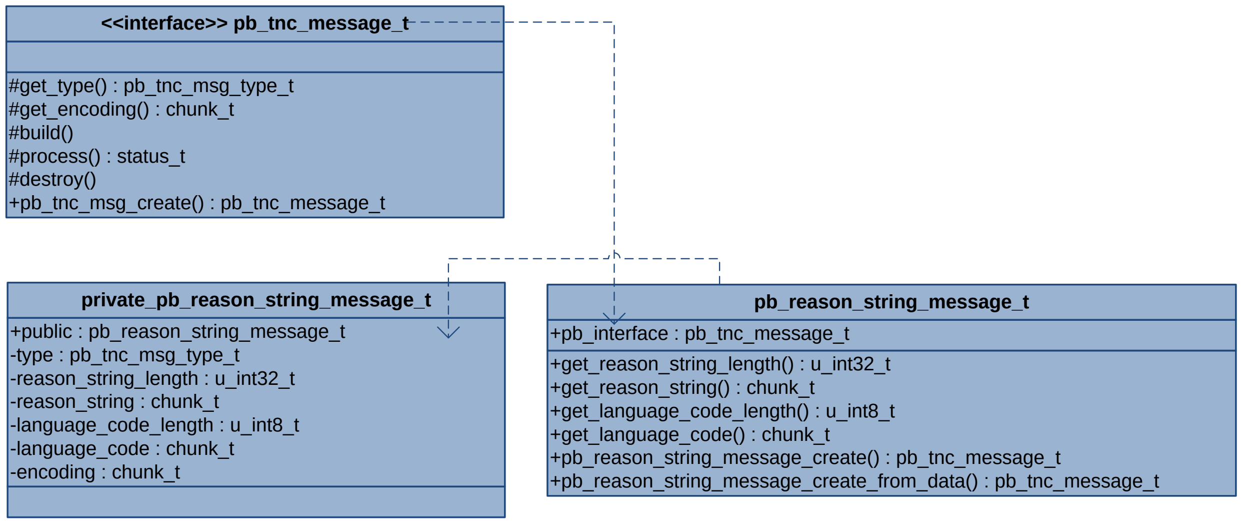

<<interface>> pb_tnc_message_t
#get_type() : pb_tnc_msg_type_t
#get_encoding() : chunk_t
#build()
#process() : status_t
#destroy()
+pb_tnc_msg_create() : pb_tnc_message_t
private_pb_reason_string_message_t
+public : pb_reason_string_message_t
-type : pb_tnc_msg_type_t
-reason_string_length : u_int32_t
-reason_string : chunk_t
-language_code_length : u_int8_t
-language_code : chunk_t
-encoding : chunk_t
pb_reason_string_message_t
+pb_interface : pb_tnc_message_t
+get_reason_string_length() : u_int32_t
+get_reason_string() : chunk_t
+get_language_code_length() : u_int8_t
+get_language_code() : chunk_t
+pb_reason_string_message_create() : pb_tnc_message_t
+pb_reason_string_message_create_from_data() : pb_tnc_message_t

Figure 21. Class Diagram for PB-Reason-String message

5.3 Tnccs-20

The implementation of PB-TNC protocol is demonstrated in this section. The primary responsibility of the PB-TNC protocol is to carry the PA messages between IMV's and IMC's. In addition to that, as introduced in the previous sections, there is also number of messages which PB-TNC protocol carries between PB Client and PB Server, constructs and processes them. The state of the assessment is also managed by PB-TNC protocol which is changed usually on Batch exchange between PB Server and PB Client.

5.3.1 Plugin

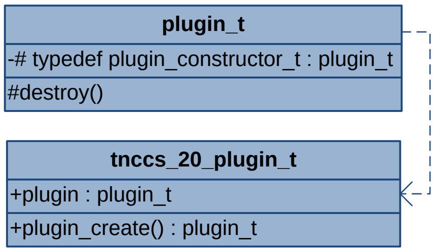

plugin_t
-# typedef plugin_constructor_t : plugin_t
#destroy()
tnccs_20_plugin_t
+plugin : plugin_t
+plugin_create() : plugin_t

Figure 22. Class Diagram for tnccs-20 plugin

As all plugins of charon daemon does, the tnccs-20 plugin implements the ***plugin_t*** interface. The daemon loads the plugins at strongSwan startup; each plugin registers itself at the daemon to hook in functionality. A separate section follows on how to configure the strongSwan in order to test the plugin.

5.3.2 Implementation

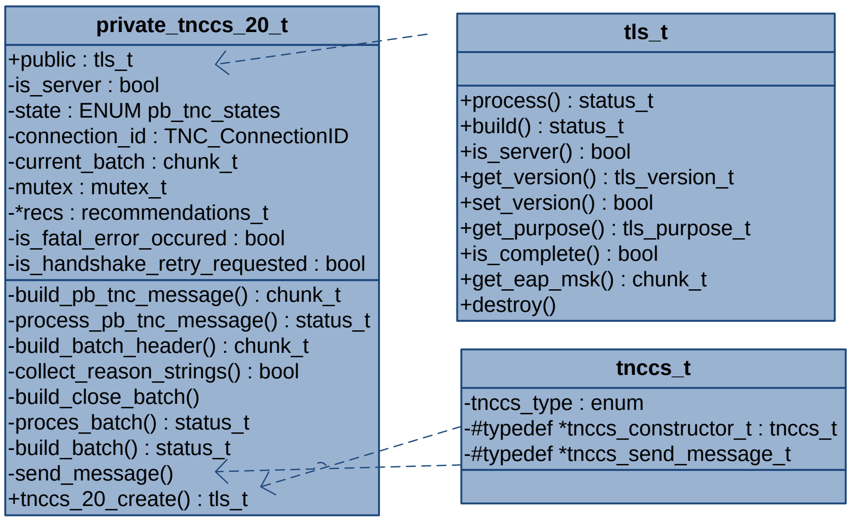

private_tnccs_20_t
+public : tls_t
-is_server : bool
-state : ENUM pb_tnc_states
-connection_id : TNC_ConnectionID
-current_batch : chunk_t
-mutex : mutex_t
-*recs : recommendations_t
-is_fatal_error_occured : bool
-is_handshake_retry_requested : bool
-build_pb_tnc_message() : chunk_t
-process_pb_tnc_message() : status_t
-build_batch_header() : chunk_t
-collect_reason_strings() : bool
-build_close_batch()
-proces_batch() : status_t
-build_batch() : status_t
-send_message()
+tnccs_20_create() : tls_t
tls_t
+process() : status_t
+build() : status_t
+is_server() : bool
+get_version() : tls_version_t
+set_version() : bool
+get_purpose() : tls_purpose_t
+is_complete() : bool
+get_eap_msk() : chunk_t
+destroy()
tnccs_t
-tnccs_type : enum
-#typedef *tnccs_constructor_t : tnccs_t
-#typedef *tnccs_send_message_t

Figure 23. Class Diagram for private data of tnccs_20_t class

5.3.2.1 Public Member

The instance of TNCCS 2.0 protocol handler is created with the help of the method ***tnccs_20_create***, which returns its first struct member of type interface ***tls_t***. By using this member called ***public***, the below layer protocol ***eap_tls*** can call the functions: ***process*** for processing the received batch, ***build*** for creating the batch to be sent, ***is_server*** for getting whether this TLS stack is working as server, ***get_purpose*** for getting the purpose this TLS stack instance, ***is_complete*** to see whether the TLS negotiation is completed successfully, ***get_eap_msk*** for getting the Master Session Key for EAP (our implementation just sends empty chunk_t value) and finally the ***destroy*** function to free the memory for the tnccs_20_t object.

5.3.2.2 Private Members

is_server boolean value indicates whether the endpoint is acting as PB Server.

state variable has one of the values in the enumeration illustrated below, depending on the state of assessment.

```
/**
 * PB-TNC States (State machine) as defined in section 3.2 of RFC 5793
 */
 enum pb_tnc_states_t {
      PB_STATE_INIT =                     0,
      PB_STATE_SERVER_WORKING =           1,
      PB_STATE_CLIENT_WORKING =           2,
      PB_STATE_DECIDED =                  3,
      PB_STATE_END =                      4,
};
```

connection_id – a variable holds the integer ID for each connection between PB Server and PB Client, as more than one connection might exist at the same time.

current_batch – a variable holds the batch that is being constructed and is going to be sent

mutex – this locks the access to the current_batch variable when the variable is being written

recs – is the holder for the recommendations which are provided from IMV's for the particular assessment

is_fatal_error_occured – a boolean value that takes TRUE if the fatal PB-Error message was delivered while processing the received batch, and is checked when the next batch is ready to be sent. Should it be set to TRUE the connection is closed immediately.

Is_handshake_retry_requested – a boolean value which takes TRUE if the function RequestHandshakeRetry, defined in IF-IMV or IF-IMC [9][10] API was called from an IMV or IMC and it is checked when the next batch is ready to be sent. In the case it is set to TRUE, the constructed batch is dropped if exists and the CRETRY or SRETRY batch is sent depending on whether an IMV or IMC has triggered the handshake retry.

```
TNC_Result TNC_TNCS_RequestHandshakeRetry(/*in*/ TNC_IMVID imvID,
                                    /*in*/ TNC_ConnectionID connectionID,
                                    /*in*/ TNC_RetryReason reason);
TNC_Result TNC_TNCC_RequestHandshakeRetry(/*in*/ TNC_IMCID imcID,
                                    /*in*/ TNC_ConnectionID connectionID,
                                    /*in*/ TNC_RetryReason reason);
```

The implementation of RequestHandshakeRetry function in tnccs_manager_t class:

```
TNC_Result (*request_handshake_retry)(tnccs_manager_t *this, bool is_imc,
                                    TNC_UInt32 imcv_id,
                                    TNC_ConnectionID id,
                                    TNC_RetryReason reason);
```

5.3.2.3 Private static functions

The responsibilities of the most of private functions are related to the PB-TNC messages and therefore were explained in the sections of corresponding messages in this chapter. There is an additional helper function ***build_close_batch*** for creating the new Batch with Error Message in case error is occurred during the processing of the received batch. This function will be explained in the Error Handling section of this chapter.

5.3.2.4 Connection

The connection is created for PB Server and PB Client respectively in the following code:

```
charon->tnccs->create_connection(charon->tnccs, (tnccs_t*)this,_send_message,
                             &this->is_handshake_retry_requested, &this->recs);
```

```
charon->tnccs->create_connection(charon->tnccs,(tnccs_t*)this, _send_message,
                             NULL, NULL);
```

These function calls return the assigned Connection ID and from then on, the connection is always referred by that ID.

5.4 State Machine

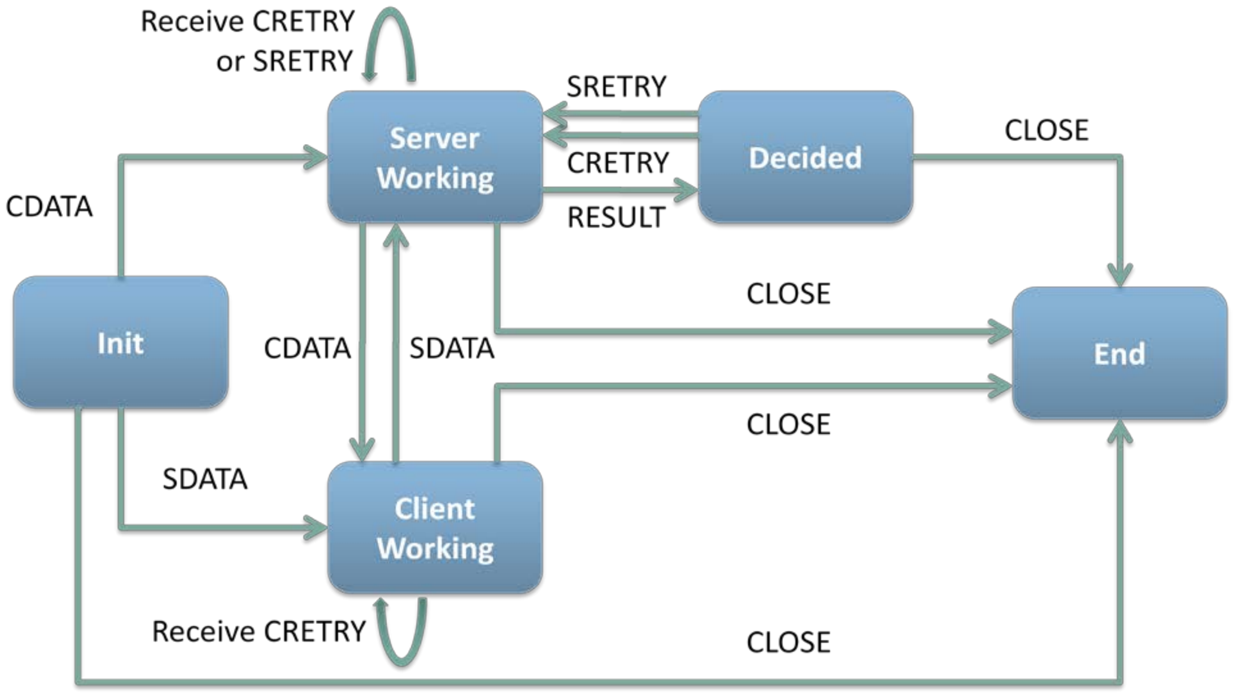


Figure 24. PB-TNC State Machine [2]

The Figure above illustrates the set of states that PB-TNC assessment session can have and the possible transitions between them. The texts in the boxes are the states and the labels for the arrows indicate the type of Batches which trigger the state transitions upon sending or reception.

The explanation on how the state transitions during the assessment implemented follows in the subsequent paragraphs. As mentioned in the Private Members section for tnncs-20 Implementation, the member variable ***state*** is the holder for the assessment state at the given time.

Init –This is the initial value assigned for the state when the instance of TNCCS 2.0 protocol handler is first created.

Server working – The state variable takes this value after the batch is sent from the PB Client to the PB Server. This is the first state transition in our implementation, because the Client starts the handshake by sending the first CDATA batch.

Client working – When the batch is sent from the PB Server to the PB Client that state variable takes this value. This is most probably the second transition except the case when PB Server has already made the Access Recommendation and sends the RESULT batch as the second exchange batch.

Decided – The state variable takes this value when the PB Server sends the RESULT batch containing the PB-Assessment-Result message. This would happen when the PB-Server has not received anything to be delivered from the IMV's, so it assumes that the global access decision is anticipated to be made and calculates it based on the individual Access Recommendations from the IMV's.

End – Upon the reception of the RESULT batch with PB-Assessment-Result message PB Client creates the empty CLOSE batch and sends it if no errors occur in between. When the PB Server correspondingly receives that empty CLOSE batch it closes the connection and TNCCS 2.0 protocol handler instance is destroyed just after the state variable gets the value of END.

The case explained above is the assumedly best case of Endpoint Assessment triggered by the NEA Client. The Reassessment possibility exists in our implementation, but could not be tested with the IMV and IMC pairs from TNC@FHH, due to the fact that these do not use this very functionality.

5.5 Error handling

There are number of circumstances defined in the RFC 5793 [2], in which the PB Server or PB Client has to send the CLOSE batch with fatal Error message. The following table lists these certain failure situations as well as the circumstances that are checked additionally in our implementation:

No.	Circumstance	Action taken	Reference (RFC 5793)
1	A Batch, shorter than 8 bytes is received (Batch header is 8 byte long)	Connections is closed with failure	4.1
2	A Batch with Unsupported Version is received	Close batch sent with fatal PB-Error message with Version Not Supported Error code	4.1
3	PB Server receives a Batch with Directionality bit that doesn't equal to 0	Connections is closed with failure	4.1
4	PB Client receives a Batch with Directionality bit that doesn't equal to 1	Connections is closed with failure	4.1
5	PB Client receives CDATA, CRETRY batch or PB Server receives SDATA, SRETRY, RESULT batch	Connections is closed with failure	4.1
6	Empty RESULT batch is received	Connections is closed with failure	4.1
7	Batch with Unknown Batch Type is received	Close batch sent with fatal PB-Error message with Unexpected Batch Type Error code	4.1
8	PB-TNC Message with reserved Vendor ID or reserved Message Type value or	Close batch sent with fatal PB-Error	4.2

	reserved PA Message Subtype is received	message with Invalid Parameter Error code	
9	PB-TNC Message with No-Skip flag set and Unknown Message Type is received	Close batch sent with fatal PB-Error message with Unsupported Mandatory Message Error code	4.2
10	PB-TNC Message is received which is not long enough	Close batch sent with fatal PB-Error message with Invalid Parameter Error code	All message types sections
11	PB-PA message without No-Skip flag set	Close batch sent with fatal PB-Error message with Invalid Parameter Error code	4.5
12	PB Server receives either PB-Assessment-Result or PB-Access-Recommendation or PB-Reason-String message which is not contained in batch of type RESULT	Ignore the message	Corresponding message types sections
13	PB Server receives either PB-Assessment-Result or PB-Access-Recommendation or PB-Reason-String message	Close batch sent with fatal PB-Error message with Invalid Parameter Error code	Corresponding message types sections
14	PB-Assessment-Result or PB-Access-Recommendation message with unknown Assessment Result code or unknown Access Recommendation code is received	Close batch sent with fatal PB-Error message with Invalid Parameter Error code	4.6 and 4.7
15	PB-Access-Recommendation or PB-Remediation-Parameters or PB-Language-Reference or PB-Reason-String message with No-Skip flag set is received	Close batch sent with fatal PB-Error message with Invalid Parameter Error code	Corresponding message types sections
16	PB-Remediation-Parameters message is not contained in RESULT batch	Ignore the message	4.8
17	PB-Error message with unknown error code is received	Ignore the message	4.9
18	PB-Language-Reference message is received with language preference that is not formed according to the standard	Connections is closed with failure	4.10

19	PB-Reason-String message with null terminated Reason String or Language Code is received	Close batch sent with fatal PB-Error message with Invalid Parameter Error code	4.11

The most of these circumstances happen during the processing action of the received batch, therefore the helper function ***build_close_batch*** was implemented in order to create close batch with given error message and drop the unprocessed part of the received batch and if exists, the batch that has been constructed for transmission as well.

The errors that are explicitly defined in the RFC 5703 [2] forces the ***process_batch*** private function to be returned with the code ***VERIFY_ERROR*** (type status_t) and we would send the CLOSE batch that is constructed in the next ***build_batch*** function execution.

On the other hand the local errors get the ***process_batch*** function to be returned with status ***FAILED***; consequently the EAP-TNC will fail immediately. This is for example the case when the received message has enough bytes to pass the minimum length check but not long enough to contain all the necessary fields as expected.

5.6 Logging

The logging is done through the strongSwan debugging feature as demonstrated in the following piece of code:

```
#include <debug.h>
DBG1(DBG_TNC, "TNCCS Unknown version");
DBG2(DBG_TNC, "creating batch header");
DBG3(DBG_TNC,"%B", &msg);
```

The Error situations and important messages are logged on level 1 whereas the explanatory messages have the second debugging level and binary print out is assigned the level 3.

The debugging level can be set through ***ipsec.conf*** configuration file of strongSwan as follows:

```
# /etc/ipsec.conf - strongSwan IPsec configuration file

config setup
      charondebug="tnc 3"
```

5.6 Decision Policy

There is small conceptual difference between the IETF standards and TNC specifications on the issue: whether the Assessment result or Access Recommendation is important. According to the IF-

IMV [9] each IMV has to provide the action recommendation and evaluation result based on the attribute values it has received so far from the corresponding IMV on client side. This is done by IMV calling the function defined in IF-IMV API:

```
TNC_Result TNC_TNCS_ProvideRecommendation(/*in*/ TNC_IMVID imvID,
                       /*in*/ TNC_ConnectionID connectionID,
                       /*in*/ TNC_IMV_Action_Recommendation recommendation,
                       /*in*/ TNC_IMV_Evaluation_Result evaluation);
```

Since the access decision is made based on the individual Action Recommendations from the IMV's it is essential that the client side also receives the Global Access Recommendation. But as defined in the RF 5793 the server is not obliged to send the Access Recommendation message but must sent the Assessment Result message within the RESULT batch. The No-Skip flag is cleared for the Access-Recommendation message and set for the Assessment-Result message.

So from the standards from IETF point of view the global access decision has to be made based on the evaluation results from the IMV's. But since the IMV's we're using are developed according to TNC specifications, some of them do not provide the appropriate evaluation result and just concentrates on giving the correct access recommendation. As a result it could happen that the client receives the assessment result "don't know" but the access request will be accepted.

The strongSwan plugin tnc-imv implements the recommendations_t class, which contains the functions concerning the recommendations.

```
METHOD(recommendations_t, provide_recommendation, TNC_Result,
                 private_tnc_imv_recommendations_t* this, TNC_IMVID id,
                 TNC_IMV_Action_Recommendation rec,
                 TNC_IMV_Evaluation_Result eval);
METHOD(recommendations_t, have_recommendation, bool,
                 private_tnc_imv_recommendations_t *this,
                 TNC_IMV_Action_Recommendation *rec,
                 TNC IMV Evaluation Result *eval);
```

On the other hand, it is important to have appropriately defined decision policy in the strongSwan configuration. Based on this policy the ***have_recommendation*** function above determines the global decision.

We adhered the two already existed policies from libtnc library from Mike McCauley. ***RECOMMENDATION_POLICY_ALL*** sets the global decision to "allow" if all the IMV's recommend "allow", to "isolate" if all the IMV's recommend "isolate" in all other cases global access recommendation is set to "no access". ***RECOMMENDATION_POLICY_ANY*** sets the global access recommendation the best case, to be exact, if any IMV recommends "allow" the final decision is "allow", else if any IMV recommends "isolate" then the decision is "isolate", else the final recommendation is "no access".

One additional policy was added in our implementation, which applies as the default policy if nothing is set in strongSwan configuration. ***RECOMMENDATION_POLICY_DEFAULT*** takes the worst case out of all IMV recommendations, that is if any of the IMV's recommended "no access" the final decision is "no access", else if any of the IMV's recommend "isolate" then the final recommendation is "isolate", else if any of the IMV's provide recommendation "allow" then the global recommendation is "allow", else the global decision is "no recommendation".

In case not all of the IMV's have provided their recommendations at the time of ***have_recommendations*** function call, the ***solicit_recommendation*** function has to be called for each IMV. This demands every IMV to give their access recommendation so that the final decision could be calculated once again with the ***have_recommendations*** function call. The definition for the function in IF-IMV [9] API is follows:

```
TNC_Result TNC_IMV_SolicitRecommendation(
                  /*in*/ TNC_IMVID imvID,
                  /*in*/ TNC_ConnectionID connectionID);
```

The solicit recommendation function, implemented by **tnc-imv** plugin is called from tnccs-20 plugin code as follows:

```
charon->imvs->solicit_recommendation(charon->imvs, this->connection_id);
```

This function cal triggers the ***solicit_recommendation*** function of all the IMV's to be called and makes sure that all the IMV's provide their respective recommendations.

5.7 Policy Enforcement

When the TNCCS 2.0 protocol handler instance is destroyed and connection between PB Server and PB Client is finished, the policy enforcement has to be made based on the global access recommendation made by PB Server. NEA clients are consequently connected by VPN gateway to the subnet "***rw-allow***" if the access recommendation made by PB Server is "allow", to the subnet "***rw-isolate***" if the recommendation was "isolate".

On the other hand, for any reason the PB Server has not made the final access recommendation yet at the time of connection closing or recommended "no access" or "no recommendation", the access request of NEA Client fails.

5.8 Constraints and limitations

The PB-TNC message of type PB-Remediation-Parameters was not implemented in our project. Although the remediation feature is one of the important aspects of NEA, we are currently unable to distinguish the PA messages from IMV's whether the content includes remediation instructions. The IF-M and PA-TNC [11][3] standards enables this feature, but PA-TNC protocol was not implemented within the scope of the current semester project.

5.9 Testing

There are two testing scenarios deployed by Dr.Andreas Steffen on website of strongSwan [33][34]. The UML testing environment is necessary for preparation of these testing scenarios which could be found with the instruction also on strongSwan.org [19].

The necessary configurations and preparations are explained in the following sections. For installation instructions of TNC@FHH please refer to [18] and for the installation instruction of strongSwan please refer to [19].

5.9.1 IMV and IMC Configuration

Example configuration for NEA Server with all the available IMV's enabled from TNC@FHH:

```
#/etc/tnc_config
# IMV-Configuration file of TNC@FHH-TNC-Server
# path to IMV libraries depends on where the TNC@FHH package was installed

IMV "Dummy" /usr/local/lib/libdummyimv.so.0.7.0
IMV "Example" /usr/local/lib/libexampleimv.so.0.7.0
IMV "Platid" /usr/local/lib/libplatidimv.so.0.7.0
IMV "Clamav" /usr/local/lib/libclamavimv.so.0.7.0
IMV "Attestation" /usr/local/lib/libattestationimv.so.0.7.0
IMV "Hostscanner" /usr/local/lib/libhostscannerimv.so.0.7.0
```

Example configuration for NEA Client with all the available IMC's enabled from TNC@FHH:

```
#/etc/tnc_config
# IMC-Configuration file of TNC@FHH-TNC-Client
# path to IMC libraries depends on where the TNC@FHH package was installed

IMC "Dummy" /usr/local/lib/libdummyimc.so.0.7.0
IMC "Example" /usr/local/lib/libexampleimc.so.0.7.0
IMC "Platid" /usr/local/lib/libplatidimc.so.0.7.0
IMC "Clamav" /usr/local/lib/libclamavimc.so.0.7.0
IMC "Attestation" /usr/local/lib/libattestationimc.so.0.7.0
IMC "Hostscanner" /usr/local/lib/libhostscannerimc.so.0.7.0
```

5.9.2 strongSwan Configuration

By activating the appropriate plugins, a strongSwan VPN Client can act as a TNC Client and a strongSwan VPN Gateway can take on either the role of a "Policy Enforcement Point" (PEP) only which forwards all EAP-TTLS packets via EAP-RADIUS to an external AAA-Server or alternatively can additionally act as a TNC Server [32]. For detailed configuration examples please refer to the strongSwan Wiki on Trusted Network Connect HowTo guide.

In addition to these, the following are the supplemental configurations that are specific for TNC implementation in strongSwan:

- The Recommendation Policy – The Policy to adhere when making global Access Recommendation
- The Preferred Language – The language preference of the NEA Client

```
#/etc/strongSwan.conf
charon {
    plugins {
    tnc-imv {
      #RECOMMENDATION_POLICY_DEFAULT
      recommendation_policy = 0
      # recommendation_policy = 1  RECOMMENDATION_POLICY_ANY
      # recommendation_policy = 2  RECOMMENDATION_POLICY_ALL
    }
    tnc-imc {
      preffered_language = de
    }

  }
}
```

6 Conclusion

With the implementation of PB-TNC protocol RFC 5793, the strongSwan has become the first Open Source VPN software that is capable of carrying out Network Endpoint Assessment. From the strongSwan developers release version 4.5.1.dr2 on, the tnccs-20 plugin codes are integrated to the git master branch of strongSwan, thus could be downloaded and tested.

The proposed standard for PT transport protocol is being developed as Internet Draft and not yet released. Therefore the refactoring of transport protocol implementation in strongSwan might be necessary in order to conform to the to-be-released specification from IETF NEA Working Group. The tnccs-11 plugin which implements TNC IF-TNCCS 1.1 specification should be fixed so that the any client can be assessed independent of the version of IF-TNCCS protocol they're using.

The insights and experiences gained from this project can be helpful for implementing an IMC and IMV pair that assesses the posture of endpoints using Trusted Platform Module which could be the topic of the next semester project.

Glossary

The terms that are not included here are explained in more detail in corresponding specifications. RFC 5209 [1] contains the list of the Terminology for NEA.

Access requestor (Network Access Requestor) - AR is the endpoint that is attempting to access the network and may be any device that is managed by the NEA system, including workstations, servers, printers, cameras and other IP-enabled devices.

Policy Enforcement Point – The PEP could be a network device, like a switch, firewall or router; an out-of-band device that enforces the endpoint security assessment for all AR’s.

Policy Decision Point - Based on the AR's posture and a network’s defined policy, the PDP determines what access should be granted. Once the PDP determines which policy to apply, it communicates the access control decision to the PEP for enforcement.

Integrity Measurement Collector (Posture Collector) - is responsible for collecting standard and/or vendor-specific Posture Attributes for particular features of the endpoint, it is also responsible for handling assessment decisions and remediation instructions.

Integrity Measurement Verifier (Posture Validator) – is responsible for handling Posture Attributes from corresponding Posture Collector(s). It performs the posture assessment for one or more features of the endpoint (e.g., anti-virus software) and creates the result and, if necessary, the remediation instructions, or it may choose to request additional attributes

Acronyms

NEA – Network Endpoint Assessment

PB – Posture Broker

PA – Posture Attribute

PT – Posture Transport

TCG – Trusted Computing Group

TNC – Trusted Network Connect

IMV – Integrity Measurement Verifier

IMC – Integrity Measurement Collector

AR – Access Requestor

PEP – Policy Enforcement Point

PDP – Policy Decision Point

AAA – Authentication Access Authorization

TLV – Type Length Value

AVP – Attribute Value Pair

IETF – Internet Engineering Task Force

RFC – Request for Comments

EAP – Extensible Authentication Protocol

TPM – Trusted Platform Module

NAC – Network Admission Protocol

ACS – Access Control Server

NAP – Network Access Protection

SHA – System Health Agent

SHV – System Health Validator

Table of Figures

Table of Source Code References

The following source code references are based on strongSwan git branch ***tnccs_20*** (30.Nov.2010) The references source codes could be modified or removed in the stable version of strongSwan after the integration of the plugin code into the master branch.

Bibliography

[1] P.Sangster, H. Khosravi, M. Mani, K. Narayan, J. Tardo , RFC 5209 „Network Endpoint Assessment (NEA): Overview and Requirements“ , IETF Network Working Group, June 2008

[2] R. Sahita, S. Hanna, R. Hurst, K. Narayan, RFC 5793 “PB-TNC: A Posture Broker (PB) Protocol Compatible with Trusted Network Connect (TNC)” ISSN: 2070-1721 , IETF, March 2010

[3] P. Sangster, K. Narayan, RFC 5792 “PA-TNC: A Posture Attribute (PA) Protocol Compatible with Trusted Network Connect (TNC)”, ISSN: 2070-1721, IETF, March 2010

[4] P. Sangster, Internet draft “PT-TLS: A Posture Transport (PT) Protocol Compatible with TNC Using Transport Layer Security (TLS draft-sangster-nea-pt-tls-01.txt”, IETF, August 24, 2010

[5] Trusted Computing Group, Specification Version 2.0 “TNC IF-TNCCS: TLV Binding”, January 22, 2010

[6] Trusted Computing Group, Specification Version 1.1 “TNC IF-TNCCS”, February 5, 2007

[7] Trusted Computing Group, Specification “TNC IF-TNCCS: Protocol Bindings for SoH”, May 21, 2007

[8] Trusted Computing Group, Specification Version 1.4 “TNC Architecture for Interoperability”, May 18, 2009

[9] Trusted Computing Group, Specification Version 1.2 “TNC IF-IMV”, February 5, 2007

[10] Trusted Computing Group, Specification Version 1.2 “TNC IF-IMC”, February 5, 2007

[11] Trusted Computing Group, Specification Version 1.0 “TNC IF-M: TLV Binding”, March 10, 2010

[12] Trusted Computing Group, Specification Version 1.1 “TNC IF-T: Protocol Bindings for Tunneled EAP Methods”, May 21, 2007

[14] Trusted Computing Group, Trusted Network Connect IF-IMV API version 1.20 “*tncif.h*, Microsoft Windows DLL Platform Binding C Header”, February 5, 2007

[14] Trusted Computing Group, TNC IF-IMV API version 1.20 Revision 8 “*tncifimv.h,* Microsoft Windows DLL Platform Binding C Header”, February 5, 2007

[15] Trusted Computing Group, TNC IF-IMC API version 1.20 Revision 8 “*tncifimc.h,* Microsoft Windows DLL Platform Binding C Header”, February 5, 2007

[16] Trusted Computing Group, “FAQ on Trusted Computing Group and the Internet Engineering Task Force”, March 2010

[17] Mike McCauley, Documentation for “libtnc”, A TNC implementation for Windows and Unix, for libtnc version 1.19, May 8, 2008

[18] Ingo Bente, Bastian Hellmann, Jan Bernhardt, Arne Welzel, TNC@FHH An Open Source TNC Implementation, Documentation Version 0.7.0, Trust@FHH Research Group, May 3, 2010

[19] strongSwan http://www.strongswan.org/, Documentation, Wiki

[20] FreeRADIUS http://freeradius.org/, Documentation, Wiki, FAQ

[21] wpa_supplicant http://hostap.epitest.fi/wpa_supplicant/

[22] Mike Fratto, "Tutorial: Network Access Control (NAC)", July 17, 2007

[23] Joel Snyder, "NAC: What went wrong?", Network World, May 24, 2010

[24] Wikipedia,"Network Access Control", December 2010

[24] Wikipedia,"Network Access Protection", December 2010

[24] Wikipedia,"Trusted Network Connect", December 2010

[25] Microsoft, Network Access Protection (NAP) in Windows Server 2008

[26] Trusted Computing Group, Trusted Network Connect (TNC)

[27] Cisco Systems, Network Admission Control (NAC)

[28] Microsoft, System Health Agents (SHAs) and System Health Validators (SHVs) that are available from NAP partners, September 30. 2008

[29] TNC@FHH, Trust@FHH Wiki TNC@FHH User Documentation

[30] Interop Labs, What is TCG's Trusted Network Connect?, Trusted Coputing Group

[31] strongSwan Wiki, Object Oriented Programming Style, strongSwan Developer Documentation

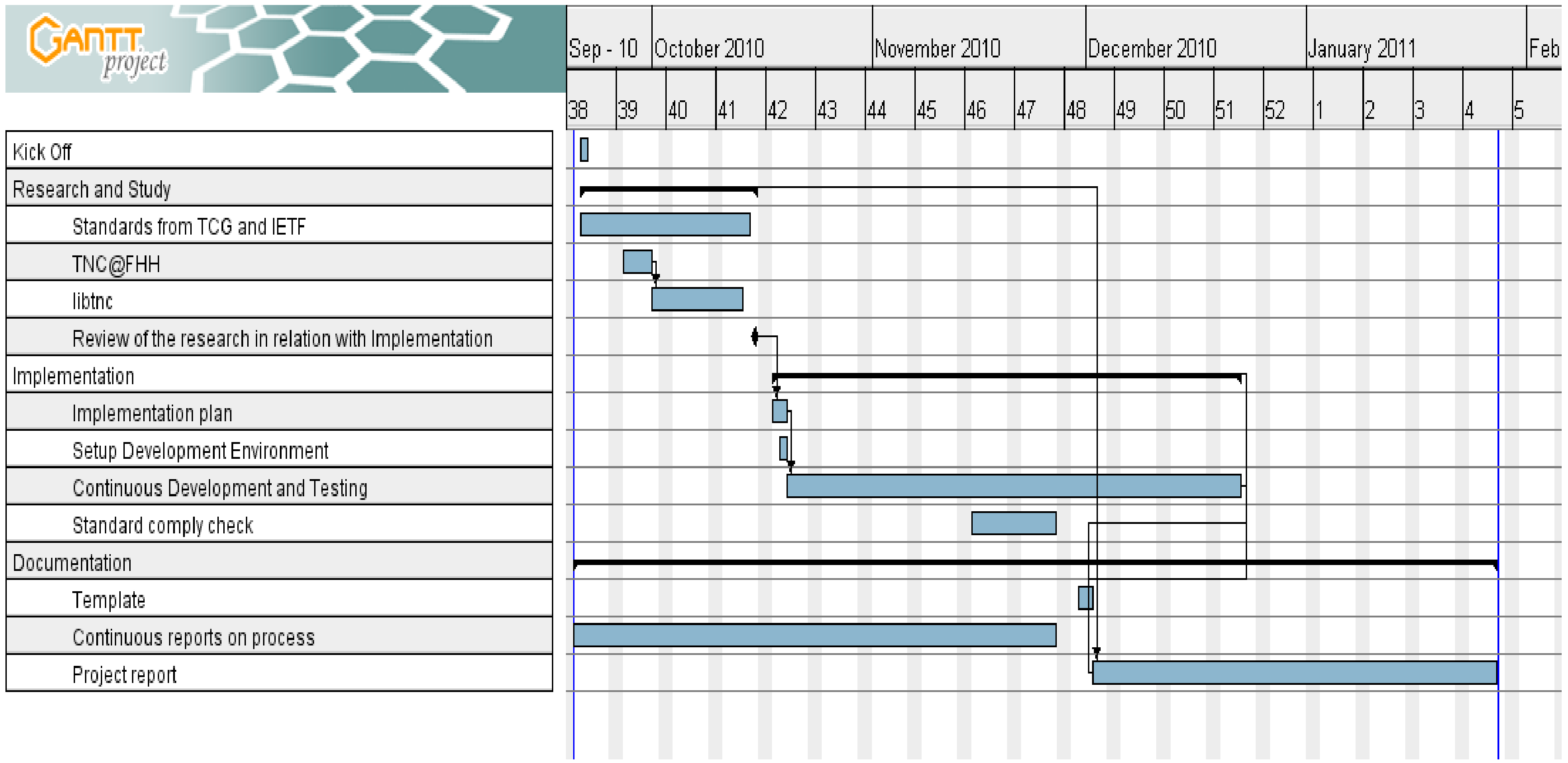
[32] strongSwan Wiki, Trusted Network Connect HowTo, strongSwan Developer Documentation

[33] strongSwan Test results, Test ikev2/rw-eap-tnc-20, strongSwan.org

[34] strongSwan Test results, Test ikev2/rw-eap-tnc-20-tls, strongSwan.org

[35] libtnc at SourceForge, http://sourceforge.net/projects/libtnc/, version 1.24

Project plan



Kick Off
Research and Study
Standards from TCG and IETF
TNC@FHH
libtnc
Review of the research in relation with Implementation
Implementation
Implementation plan
Setup Development Environment
Continuous Development and Testing
Standard comply check
Documentation
Template
Continuous reports on process
Project report