The Linux Integrity Measurement Architecture and TPM-Based Network Endpoint Assessment

Andreas Steffen
ITA Institute for Internet Technologies and Applications
HSR University of Applied Sciences Rapperswil
CH-8640 Rapperswil, Switzerland
Email: andreas.steffen@hsr.ch

***Abstract*—The strongSwan VPN software fully supports Network Endpoint Assessment (NEA) and is able to collect evidence from the Integrity Measurement Architecture (IMA) on a Linux client and to transfer measurement data on more than 1000 system files via the Trusted Network Connect (TNC) protocols PA-TNC, PB-TNC, and PT-EAP over IKEv2 EAP-TTLS to a strongSwan TNC server. A quote signature by the Trusted Platform Module (TPM) on the TNC client establishes the trustworthiness of the IMA measurements. The overall decision process of either giving a client full network access or to relegate it to an isolation network takes less than 20 seconds.**

I. Introduction

The last two years saw the emergence of several Trusted Network Connect standards jointly developed and published by the IETF NEA working group [34] and the Trusted Computing Group (TCG) [35]. This raises the hope for a convergence of the multitude of proprietary Network Access Control (NAC) frameworks created by Cisco, HP, Juniper, Microsoft and others to a single set of standards over the next three to five years.

The strongSwan open source project [32] is one of the first organizations which has implemented the new NEA protocols and the first part of this paper is going to give a short overview on how the three layers of the NEA reference model work and how the TCG TNC specifications fit into it.

Next we will treat the concept of Platform Trust Services (PTS) and how in that context Linux IMA can be used to establish trust into a host platform. The recently published TCG PTS protocol [17] comes in handy to transfer BIOS and IMA measurements as standardized PTS PA-TNC attributes to the TNC server. On the TNC server a reference database specific to each particular Linux distribution is built using the strongSwan *attest* tool which allows the comparison of the actual IMA file measurements with values taken on a reference system known to be clean and uncorrupted. For client-specific BIOS measurements, the reference values are taken during a preliminary learning phase and are stored in the database linked to the fingerprint of the client TPM's particular Attestation Identity Key (AIK).

Finally we are going to discuss two issues that we encountered while implementing and testing our remote attestation solution. One problem are multiple versions of dynamic libraries that have differing IMA measurements values but the same relative filename. The second challenge consists of transferring 100-200 kB of measurement data over a transport protocol that does not support fragmentation.

II. Network Access Control

In a typical Network Access Control setup as shown in Fig. 1, a Policy Enforcement Point (PEP) which usually is either an IEEE 802.1X-enabled layer 2 switch or WLAN access point, or a layer 3 IPsec or layer 4 SSL VPN gateway, controls the access to a campus network. A NAC client seeking access to the protected network must undergo a series of health measurements first. Depending on the measurement results which are controlled and evaluated by a NAC Server and assisted by a Policy Manager which together form a Policy Decision Point (PDP), the computer under test is either *allowed* into the campus network, *blocked* from access or is *isolated* by relegating it to a special ”Isolation” or ”Remediation” network where the NAC client can update and patch its software or in case of an infection by malware, can be sanitized and returned to a healthy status.

When the first NAC solutions hit the market a couple of years ago, each vendor introduced his own set of proprietary NAC protocols which made interoperability next to impossible. But thanks to the *Trusted Network Connect* framework defined by the TCG, a whole set of specifications [10]–[16] has been made available, some of which have recently been adopted as Internet Standards [2]–[5] by the IETF.

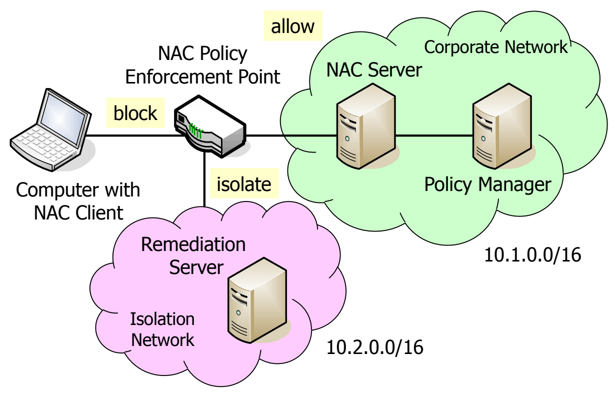

Fig. 1. Network Access Control *(source: FHH)*

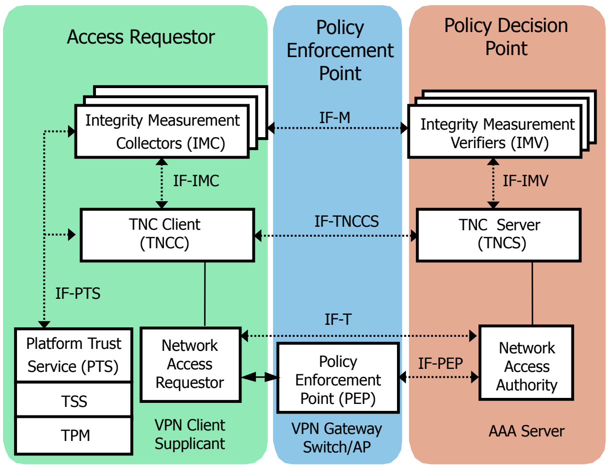

Fig. 2. TCG TNC framework *(source: TCG)*

A. *Trusted Network Connect (TNC)*

The TCG TNC framework is shown in Fig. 2, omitting the Meta Access Point (MAP) components and corresponding IF-MAP [22] interfaces which are not relevant in the context of this paper, although a strongSwan PEP can be equipped with an IF-MAP client interface.

On the left-hand side we have a ”Network Access Requestor” which is usually either a layer 2 supplicant or a layer 3 IPsec or layer 4 SSL VPN client, respectively. Integrated into the Access Requestor is a *TNC Client* which controls the local integrity measurements and communicates via the IF-TNCCS 2.0 Client/Server protocol [13] with a remote *TNC Server*. Attached to the TNC Client is at least one ”Integrity Measurement Collector” (IMC), a dynamic library (**.so* under Linux/Unix or **.dll* under Windows) loaded during runtime and which communicates with the TNC Client via the IF-IMC interface [11] consisting of a C language API which defines a set of functions callable either by the TNC Client or the IMC.

On the right-hand side the TNC Server is often co-located with an AAA Server which usually has an EAP-RADIUS interface [6]. Since layer 2 supplicants don’t get an IP address before the client authentication process hasn’t completed successfully and thus can reach their switch or access point only via layer 2 EAPOL (EAP-over-LAN) and similarly layer 3 IPsec clients can communicate with their VPN gateway only via IKEv2 EAP [9] during the IPsec tunnel setup phase, a half-duplex EAP-based IF-T transport protocol [14] is needed which can forward messages from the TNC Client to the TNC Server via the Policy Enforcement Point (PEP) and its RADIUS IF-PEP [16] connection with the Policy Decision Point (PDP). Because this IF-T protocol called EAP-TNC is vulnerable to certain man-in-the-middle attacks it must be protected by EAP-TTLS [7], EAP-FAST [8] or another EAP tunnel protocol. For layer 4 SSL VPN access or after layer 2 and layer 3 clients have gained access to the campus network, a broadband full-duplex TLS-based IF-T protocol [15] is preferred.

IETF NEA	TCG TNC
Posture Collector	Integrity Measurement Client
Posture Validator	Integrity Measurement Verifier
Posture Broker Client	TNC Client
Posture Broker Server	TNC Server
PA-TNC	IF-M 1.0
PB-TNC	IF-TNCCS 2.0
PT-EAP	IF-T for Tunneled EAP Methods 1.1
PT-TLS	IF-T for TLS 2.0

TABLE I
Mapping between IETF NEA and TCG TNC Terminology and Protocols

Attached to the TNC Server is at least one ”Integrity Measurement Verifier” (IMV), a dynamic library communicating with the TNC Server via an IF-IMV interface [12] which in its function set is nearly identical to its IF-IMC counterpart on the client side. IMCs and IMVs exchange messages with each other over the IF-M measurement protocol [10].

B. *Network Endpoint Assessment (NEA)*

In 2008 the IETF NEA working group defined a generic reference model for Network Endpoint Access [1] which is shown in Fig. 3.

1) Posture Attribute Protocol (PA): The NEA reference model defines a ”Posture Attribute” protocol (PA) that carries one or more attributes between ”Posture Collectors” and their associated ”Posture Validator”. The PA protocol is a message-oriented lightweight wrapper around a set of attributes being exchanged.

PA-TNC [2] which is identical to TCG TNC IF-M 1.0 [10] defines an instance of a PA protocol complying with the NEA reference model. The PA-TNC message header just consists of a version field and a message identifier. The actual information is carried by any number of PA-TNC attributes which are identified by their attribute type and the attribute vendor ID which determines the namespace the attribute is defined in.

2) Posture Broker Protocol (PB): The NEA reference model defines a ”Posture Broker” protocol (PB) that carries

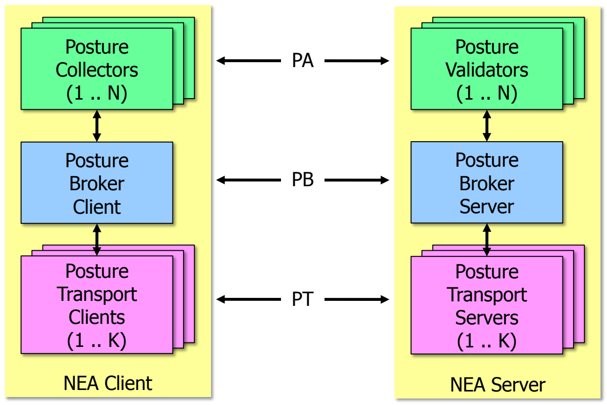
Posture Collectors (1 .. N)
PA
Posture Validators (1 .. N)
Posture Broker Client
PB
Posture Broker Server
Posture Transport Clients (1 .. K)
PT
Posture Transport Servers (1 .. K)
NEA Client
NEA Server

Fig. 3. IETF NEA reference model [1]

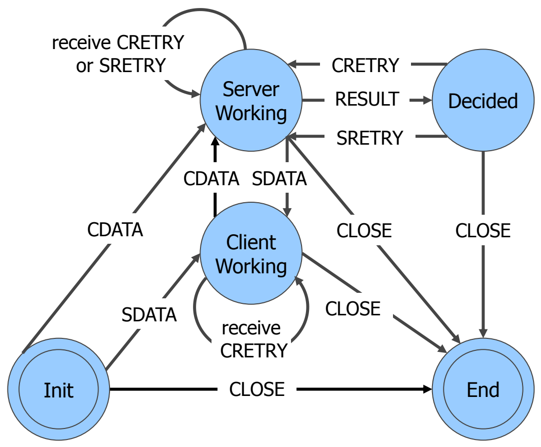

Fig. 4. PB-TNC finite state machine [3], [13]

aggregate attribute messages between the Posture Collectors on the NEA Client and the corresponding Posture Validators on the NEA Server involved in a particular assessment. The PB protocol provides a session allowing for message dialogs for every assessment. The PB protocol may also carry the global assessment decision in the Result Attribute from the ”Posture Broker Server” to the ”Posture Broker Client”.

PB-TNC [3] which is identical to TCG TNC IF-TNCCS 2.0 [13] defines an instance of a PB protocol complying with the NEA reference model. PB-TNC is a stateful session protocol governed by the finite state machine depicted in Fig. 4. Starting from an *Init* state the Posture Broker Client and Posture Broker Server exchange CDATA and SDATA batches, respectively, alternatively switching between the *Server Working* and the *Client Working* states. If the Posture Broker Server arrives at a final assessment it sends a RESULT batch, driving the state machine into the *Decided* state. With an empty CLOSE batch the PB-TNC session is terminated and goes to the final *End* state.

Both client and server can terminate the session at any time by sending an empty CLOSE batch or one containing a single PB-Error message if a fatal error occurred. If either client or server comes to the conclusion that an integrity measurement should be repeated, they can send a CRETRY or SRETRY batch, respectively, even if the session has already arrived at the *Decided* state.

The PB-TNC protocol defines a number of PB-TNC message types that can be transported in PB-TNC batches, the most important being the PB-PA message which encapsulates a PA-TNC message received from the higher PA layer.

3) Posture Transport Protocol (PT): The NEA reference model defines a ”Posture Transport” protocol (PT) between the NEA Client and the NEA Server responsible for carrying the messages generated by the PB protocol. Depending on the underlying network protocols there might be restrictions concerning the maximum payload size, the number of roundtrips or the inability of servers to initiate messages.

PT-EAP [4] which is largely based on EAP-TNC [14] defines an instance of a PT protocol complying with the NEA reference model. As with all EAP methods, the maximum message size including the EAP header is 65’535 octets. Contrary to EAP-TNC, a fragmentation mechanism isn’t foreseen in the current PT-EAP Internet Draft, so that the higher PB layer has to restrict the maximum batch size to 65’529 octets. This is going to become an important issue with Linux IMA remote attestation since a lot of bulk measurement data has to be transferred. Due to security reasons PT-EAP must always be encapsulated in an EAP tunnel method such as EAP-TTLS [7] or EAP-FAST [8].

PT-TLS [5] which is identical to TCG TNC IF-T for TLS 2.0 [15] defines another instance of a PT protocol. Based on TCP and secured by Transport Layer Security (TLS), it is a broadband protocol which allows to transport an unlimited amount of data because the underlying TLS record protocol takes care of any fragmentation issues.

III. Platform Trust Services (PTS)

A. The Lying Endpoint Problem

Network endpoint assessment only works if we can trust the measurements made by the Posture Broker Client. If the remote host platform is subverted by dangerous malware such as a root kit which hides itself well right in the operating system or even in the BIOS then a compromised Posture Collector or Broker Client will send the expected correct measurement value e.g. of a system command in order to soothe the Posture Validator but the actual command loaded and executed on the host will be a modified one serving a malicious purpose.

One way out of this lying endpoint problem is the installation of a trusted incorruptible entity on the host which acts as a kind of local deputy on behalf of the remote validator as shown in Fig. 5. A suitable tamper proof device is the ”Trusted Platform Module” (TPM) [18], a hardware chip present on most PC platforms. The TPM can accumulate and store measurement hashes in up to 24 incorruptible ”Platform Configuration Registers” (PCRs) and can sign these measurements with a protected RSA private key.

Since the TPM API is quite low-level, the Trusted Computing Group has defined a layered ”TCG Software Stack” (TSS)

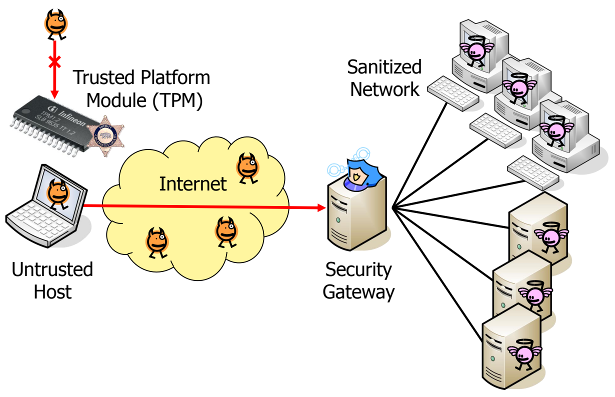

Fig. 5. Establishing trust by means of a TPM

PCR	SHA-1 Measurement Hash		Comment
0	4d894eef0ae7cb124740df4f6c5c35aa0fe7dae8	08	[S-CRTM Version]
0	f2c846e7f335f7b9e9dd0a44f48c48e1986750c7	01	[POST CODE]
	...		
7	9069ca78e7450a285173431b3e52c5c25299e473	04	[]
4	c1e25c3f6b0dc78d57296aa2870ca6f782ccf80f	05	[Calling INT 19h]
4	67a0a98bc4d6321142895a4d938b342f6959c1a9	05	[Booting BCV Device 80h, - Hitachi HTS723216L9A360]
4	06d60b3a0dee9bb9beb2f0b04aff2e75bd1d2860	0d	[IPL]
5	1b87003b6c7d90483713c90100cca3e62392b9bc	0e	[IPL Partition Data]

TABLE II
/sys/kernel/security/tpm0/ascii_bios_measurements: SRTM BIOS measurements

PCR	SHA-1 Template Hash		SHA-1 File Data Hash	Filename
10	d0bb59e83c371ba6f3adad491619524786124f9a	ima	365a7adf8fa89608d381d9775ec2f29563c2d0b8	boot_aggregate
10	76188748450a5c456124c908c36bf9e398c08d11	ima	f39e77957b909f3f81f891c478333160ef3ac2ca	/bin/sleep
10	df27e645963911df0d5b43400ad71cc28f7f898e	ima	78a85b50138c481679fe4100ef2b3a0e6e53ba50	ld-2.15.so
	...		...	
10	30fa7707af01a670fc353386fcc95440e011b08b	ima	72ebd589aa9555910ff3764c27dbdda4296575fe	parport.ko
	...		...	

TABLE III
/sys/kernel/security/ima/ascii_runtime_measurements: IMA runtime measurements

[19] which in itself is a 742 page document. A popular open source implementation is TrouSerS [26] created and released by IBM.

B. PTS Architecture

In order for runtime measurements to be collected, a process within a Trusted Platform must be defined to do so. The TGC ”Integrity Measurement Model” (IMM) [20] has defined ”Platform Trust Services” (PTS) as a standard IMM component which provides all the necessary services which an ”Integrity Measurement Collector” (IMC) can interface with to measure components, parse Reference Manifests, and generate Integrity Reports. As shown in the TCG TNC overview of Fig. 2, PTS is the component responsible for interfacing with the TPM via TSS and with the PTS-IMC via IF-PTS [21].

Reference Manifests, Integrity Reports and the IF-PTS interface rely on an XML-based data representation. This very flexible but also rather verbose approach of using XML encoding has been chosen e.g. by the OpenPTS project [31]. Since the PA-TNC and PB-TNC standards are both based on efficient TLV (Type-Length-Value) encoding schemes, we wanted to adhere to this lean design concept and for our strongSwan TNC implementation [33] decided to use only the subset of PTS PA-TNC attributes defined by the ”PTS Protocol Binding for IF-M” [17] that employ a TLV encoding of PTS ”Attestation Evidence” data (for more details see section III-D). As a consequence our strongSwan *Attestation* IMC described in this paper does not use the IF-PTS interface at all but collects integrity measurement data directly and accesses the TPM via the TSS API. Thus in our implementation PTS is rather an object of the *Attestation* IMC than an independent entity.

C. Linux Integrity Measurement Architecture (IMA)

The Linux Integrity Measurement Architecture (IMA) [23], [27] was introduced with the Linux 2.6.30 kernel in 2009. If a TPM is present on a host’s motherboard then during boot time, SRTM (Static Root of Trust for Measurement) BIOS measurements are made and the SHA-1 measurement hash values are stored in the *binary_bios_measurements* file located in the */sys/kernel/security/tpm0/* directory. At the same time these values are extended into specific PCRs of the TPM thus securing the BIOS content against malicious change. A shortened example of a human-readable ASCII version of the SRTM BIOS measurement list which, depending on the BIOS version, typically has between 20 to 130 entries is shown in Table II.

If IMA is enabled in the kernel then a */sys/kernel/security/ima/* directory is created and runtime measurements are made that are stored as a list in the *binary_runtime_measurements* file. A typical example of the the ASCII version is shown in Table III. The first measurement list entry is the *boot_aggregate* which consists of a SHA-1 hash of the contents of the PCRs 0. . . 7. This means with IMA on, it is sufficient to check the *boot_aggregate* value in order to quickly verify whether the BIOS measurements still have the original values.

Table III lists two IMA measurement values for each entry. The right-hand value is the SHA-1 hash of the file data itself and the left-hand value is the so called ”Template Hash” which

```
# PROC_SUPER_MAGIC
dont_measure fsmagic=0x9fa0
# SYSFS_MAGIC
dont_measure fsmagic=0x62656572
# DEBUGFS_MAGIC
dont_measure fsmagic=0x64626720
# TMPFS_MAGIC
dont_measure fsmagic=0x01021994
# SECURITYFS_MAGIC
dont_measure fsmagic=0x73636673
measure func=BPRM_CHECK
measure func=FILE_MMAP mask=MAY_EXEC
# SE Linux
measure func=PATH_CHECK mask=MAY_READ \
        obj_type=modules_object_t
```

Fig. 6. */etc/sysconfig/ima-policy*: Custom IMA Measurement Policy is the SHA-1 hash of the 20 octet file data hash concatenated with a fixed text buffer of 256 octets containing the filename truncated to 255 characters and padded up to the buffer length with NUL characters. This template hash gets extended into PCR 10 dedicated to IMA measurements.

Using a dracut [28] *initramfs* it is possible to replace the default IMA measurement policy by a custom one defined in the */etc/sysconfig/ima-policy* configuration file as listed in Fig. 6.

Besides executable files and loaded dynamic libraries, also dynamically loaded kernel modules are measured which requires a file system labeled by SELinux [29], though. Under an *Ubuntu 12.04 LTS i686* OS this results in about 1200 IMA measurements among them about 700 dynamic libraries and 60 kernel modules up to the moment when the strongSwan TNC client is started right after the Ubuntu login.

D. PTS Protocol

TNC@FHH [30] was one of the first TNC applications to implement TPM-based remote attestation. Their *attestation* IMC/IMV pair is using a proprietary encoding to embed "Attestation Identity Key" (AIK) certificates, measurement requests and PCR contents into XML-encoded TCG TNC IF-TNCCS 1.1 batches.

OpenPTS [31] uses PA-TNC attributes transported over a raw *ssh* connection with attribute types somehow similar to those of the TCG PTS protocol but defined in the OpenPTS namespace.

strongSwan [32] currently seems to be the first and only open source software which uses PTS PA-TNC attributes from the TCG namespace defined by the "PTS Protocol Binding to TNC IF-M" [17] officially released in August 2011. As already mentioned in section III-B, only the subset of attributes listed in Table IV which are needed to efficiently transfer TLV-based attestation evidence as well as Linux/Unix-style file measurements are currently supported.

PTS IF-M Attribute Name	Sender
Request PTS Protocol Capabilities	IMV
PTS Protocol Capabilities	IMC
D-H Nonce Parameters Request	IMV
D-H Nonce Parameters Response	IMC
D-H Nonce Finish	IMV
PTS Measurement Algorithm Request	IMC
PTS Measurement Algorithm Selection	IMV
Get TPM Version Information	IMV
TPM Version Information	IMC
Get Attestation Identity Key	IMV
Attestation Identity Key	IMC
Request Functional Component Evidence	IMV
Generate Attestation Evidence	IMV
Simple Component Evidence	IMC
Simple Evidence Final	IMC
Request File Metadata	IMV
Unix-Style File Metadata	IMC
Request File Measurement	IMV
File Measurement	IMC

TABLE IV
TLV/Unix subset of PTS IF-M Attribute Types [17]

IV. The strongSwan TNC Solution

The strongSwan open source VPN project [32] was founded in 2004 as one of the successors of the then discontinued FreeS/WAN IPsec project. In the meantime the whole code base has been rewritten from scratch, although still in the C programming language but with a modern, modular, object-oriented and fully multi-threaded archictecture. Currently strongSwan is the most complete open source implementation of the IKEv2 [9] Internet Key Exchange protocol and support for the deprecated but still widely used IKEv1 protocol has been added recently.

In 2010 the author became aware of the emerging IETF and TCG TNC standards and decided to add support for the following protocols to the strongSwan software:

- PA-TNC [2]
- PB-TNC [3]
- PT-EAP [4]
- IF-IMC [11]
- IF-IMV [12]
- IF-PEP [16]
- PTS [17]

The strongSwan distribution now includes *Test*, [port] *Scanner*, and *Attestation* IMC/IMV pairs that can be used with any third party TNC client/server equipped with an IF-IMC or IF-IMV interface, respectively.

Since currently no other TNC product besides strongSwan is known to support the PB-TNC standard, we have no alternative but to test the PB and underlying PT layer with the TNC client/server pair integrated into the IKEv2-based strongSwan VPN client and server, respectively.

A. Internet Key Exchange (IKEv2)

In remote access VPN scenarios usually a password-based EAP method such as EAP-MD5 or EAP-MSCHAPv2 is used for client authentication and the user credentials are managed and verified by an AAA Server as shown in Fig. 7.

If TNC support is added then an end-to-end EAP-TTLS tunnel is created between the VPN/TNC client and the AAA server, the EAP-TTLS packets being transported between the IPsec client and IPsec gateway over IKEv2-EAP and between the VPN gateway and AAA server via EAP-RADIUS. By loading the *tnc-pdp* plugin strongSwan can act as a standalone Policy Decision Point (PDP) which combines a TNC server with a simple AAA server possessing a RADIUS interface. strongSwan can also be configured as a combined Policy Enforcement Point (PEP) and PDP where the VPN gateway and the TNC server are co-located. This is the setup we are going to work with in the following paragraphs.

The *ipsec.conf* files of a typical strongSwan VPN/TNC client and matching VPN/TNC server configuration are shown in Fig. 8 and Fig. 9, respectively. The VPN network topology is equivalent to the NAC scenario shown in Fig. 1 with an *allow* and an *isolate* subnet.

With a hierarchical structure the *strongswan.conf* file in Fig. 10 defines options for the combined IKEv2/TNC client in the *charon* section and the *attestation* IMC in the *libimcv* section. Each *charon* plugin and IMC or IMV can have configuration parameters of its own. The corresponding file for the server side listed in Fig. 11 contains definitions for the combined IKEv2/TNC *charon* daemon, the *attestation* IMV

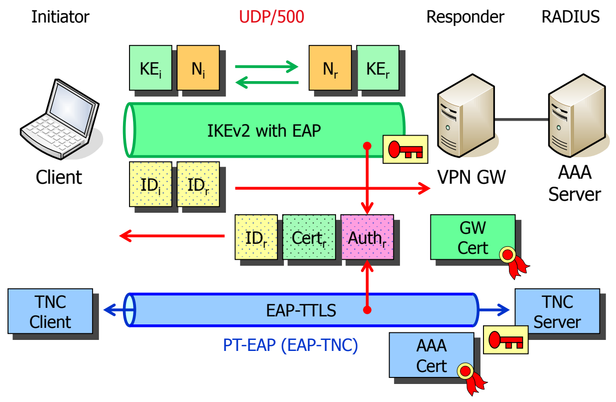

Fig. 7. PT-EAP transport via IKEv2 EAP-TTLS

```
config setup
     charondebug="tnc 2, imc 3, pts 3"

conn home
     left=%any
     leftid=carol@strongswan.org
     leftauth=eap
     right=192.168.0.1
     rightsubnet=10.1.0.0/16
     rightid=moon.strongswan.org
     rightauth=any
     auto=start
```

Fig. 8. *ipsec.conf*: configuration of strongSwan VPN client

```
config setup
     charondebug="tnc 2, imv 3, pts 3"

conn rw-allow
     rightgroups=allow
     leftsubnet=10.1.0.0/16
     also=rw-eap
     auto=add

conn rw-isolate
     rightgroups=isolate
     leftsubnet=10.2.0.0/16
     also=rw-eap
     auto=add

conn rw-eap
     left=192.168.0.1
     leftcert=moonCert.pem
     leftid=moon.strongswan.org
     leftauth=eap-ttls
     rightauth=eap-ttls
     right=%any
```

Fig. 9. *ipsec.conf*: configuration strongSwan VPN server

```
charon {
  plugins {
    eap-ttls {
      max_message_count = 0
      fragment_size = 1024
    }
    eap-tnc {
      protocol = tnccs-2.0
      max_message_count = 20
    }
    tnccs-20 {
      max_batch_size = 32754
      max_message_size = 32722
    }
    tnc-imc {
      preferred_language = en, pl, de
    }
  }
}

libimcv {
  plugins {
    imc-attestation {
      pcr_info = no
      use_quote2 = yes
      aik_cert = /home/carol/privacyca/AIK_Cert.der
      aik_blob = /home/carol/privacyca/AIK_Blob.bin
    }
  }
}
```

Fig. 10. *strongswan.conf*: options for VPN/TNC client & IMC

```
charon {
  plugins {
    eap-ttls {
      phase2_method = md5
      phase2_piggyback = yes
      phase2_tnc = yes
      max_message_count = 0
      fragment_size = 1024
    }
    eap-tnc {
      protocol = tnccs-2.0
      max_message_count = 20
    }
  }
}

libimcv {
  plugins {
    imv-attestation {
      database = sqlite:///etc/pts/config.db
      cadir = /etc/pts/cacerts
      hash_algorithm = sha1
    }
  }
}

attest {
  database = sqlite:///etc/pts/config.db
}
```

Fig. 11. *strongswan.conf*: options for VPN/TNC server & IMV

and the *attest* command line tool useful for managing the attestation database.

According to the IF-IMC [11] and IF-IMV [12] standards, the library paths of the IMCs and IMVs to be loaded must be defined in the file */etc/tnc_config* with entries of the form

```
IMC Scanner     /usr/lib/ipsec/imcvs/imc_scanner.so
IMC Attestation /usr/lib/ipsec/imcvs/imc_attestation.so
```

on the client side and with corresponding entries

```
IMV Scanner     /usr/lib/ipsec/imcvs/imv_scanner.so
IMV Attestation /usr/lib/ipsec/imcvs/imv_attestation.so
```

on the server side, respectively. These integrity measurement libraries are dynamically loaded and initialized during the startup of the TNC *charon* daemon.

B. PTS Attestation Database

The strongSwan distribution comes with an SQLite database schema to be found in the file *src/libpts/plugins/imv_attestation/tables.sql* which defines all tables needed to store attestation reference measurement data. Additionally some example file measurement data for various Linux distributions is included in the *data.sql* file located in the same directory.

The PTS attestation database is created on the TNC server with the following command:

```
cat tables.sql data.sql | sqlite3 /etc/pts/config.db
```

Alternatively strongSwan offers a plugin for MySQL and in principle any relational database could be attached by writing a corresponding SQL driver plugin.

The *attest* command line tool can be used to manage the database. E.g. the query shown in Fig. 12 returns all Linux distributions for which some demo data is available.

The strongSwan *Attestation* IMC retrieves the Linux distribution version the client is running on either from */etc/lsb-release* if this file exists or by applying some heuristics. It also determines the hardware architecture using the *uname* system call. A string is formed from this gathered platform information and sent via a standard IETF "Product Information" PA-TNC attribute to the *Attestation* IMV. The IMV can then retrieve reference measurement information using this product string. The command line query shown in Fig 13 can be used to check which file measurements are available for a given software product.

In order to verify the IMA measurements in the format depicted in Table III, the SHA-1 template hashes of typically about 10'000 files per Linux distribution version must be stored in the attestation database. This process can be fully automated by running a *build-database* shell script which currently uses about 200 *attest* calls, a small excerpt of which is shown in Fig. 15. Together with the normal SHA-1 file data hash for each entry which is computed with the `--sha1-ima` option as well and which equals the output of the *sha1sum* command and thus facilitates the debugging of version problems, the SQLite database attains a size of about 2 MB. Executable files must be stored with absolute path names whereas dynamic libraries and kernel modules are stored with both the directory path and the basename by adding the `--rel` option.

With IMA we primarily want to verify the integrity of system files. Unfortunately some application programs like *firefox* or *thunderbird* bundle their own versions of system libraries, e.g. *libfreebl3.so* or *libsoftokn3.so*. Thus in order to avoid matching errors we have to store these measurement values, too, because the lookup happens on the basename of the dynamic libraries. By the way, in the case of *firefox* or *thunderbird* it is worthwile to keep tabs on the integrity of these important applications as well.

The reference database must be built on a Linux system known to be clean. Since most Linux distributions issue security patches and bug fixes every other day, the reference database must be updated accordingly. Currently the attestation database is generated from scratch every time an update is published so that clients which haven't applied the latest patches will be strictly relegated to the remediation network in order to update their system files. In the future the hashes of older versions might be kept in the database if using these deprecated files does not pose any immediate threat.

C. PTS Functional Component Evidence

The TCG PTS protocol [17] organizes the gathering of PTS evidence around a hierarchically structured sequence of so-called *Functional Component* measurements. Some standard functional components have already been defined in the TCG namespace but unfortunately they can not be easily applied to the kind of measurements that are done by the Linux Integrity Subsystem. Therefore we created three functional components of our own defined in the ITA-HSR namespace as shown in Fig. 14. In the context of this paper we restrict ourselves to the functionality of the ITA-HSR "Linux-IMA" component which can be further subdivided by using a *Qualifier*. We have defined a "Trusted Platform" qualifier value designating the SRTM BIOS measurements and an "Operating System" qualifier for the IMA runtime measurements.

SRTM BIOS measurements are rather specific to the client

```
ipsec attest --products
  3: CentOS release 5.6 (Final) x86_64
  6: Gentoo Base System release 1.12.11.1 i686
  5: Ubuntu 10.10 i686
  4: Ubuntu 10.10 x86_64
  1: Ubuntu 11.04 i686
  2: Ubuntu 11.04 x86_64
  7: Ubuntu 11.10 i686
  8: Ubuntu 12.04 LTS i686
8 products found
```

Fig. 12. *attest* query returns all registered Linux distributions

```
ipsec attest --hashes --sha1 --product 'Ubuntu 10.10 x86_64'
  3: /lib/libdl.so.2
43:50:f0:82:51:1c:74:2c:c0:50:50:d1:8a:23:d1:da:9f:b0:93:40
  6: /lib/libxtables.so.2
92:e6:6a:e2:82:94:7f:66:54:46:82:03:9a:33:fd:1d:bd:40:22:44
  4: /sbin/iptables
86:c4:46:32:93:85:98:74:24:3d:83:74:f7:f3:ef:60:f4:4f:93:09
  9: /lib/xtables/libxt_tcp.so
d2:bf:35:56:a0:b3:8c:fb:a2:96:2d:05:8f:a8:ea:77:73:97:e8:2d
  8: /lib/xtables/libxt_udp.so
20:0e:ab:67:37:7b:f3:d5:a2:53:72:83:8c:38:84:16:58:a7:18:e4
5 SHA1 values found for product 'Ubuntu 10.10 x86_64'
```

Fig. 13. *attest* query returns all file hashes for a given product

```
ipsec attest --components
  1: ITA-HSR/Trusted GRUB Boot Loader [K.] Trusted Platform
  2: ITA-HSR/Trusted Boot [K.] Trusted Platform
  3: ITA-HSR/Linux IMA [K.] Trusted Platform
  4: ITA-HSR/Linux IMA K.] Operating System
4 components found
```

Fig. 14. *attest* query returns all registered functional components

```
#!/bin/sh
# executable files with absolute filenames
ipsec attest --add --product "Ubuntu 12.04 LTS i686" --sha1-ima --dir /sbin
ipsec attest --add --product "Ubuntu 12.04 LTS i686" --sha1-ima --dir /usr/sbin
ipsec attest --add --product "Ubuntu 12.04 LTS i686" --sha1-ima --dir /bin
ipsec attest --add --product "Ubuntu 12.04 LTS i686" --sha1-ima --dir /usr/bin
      ...
# dynamic libraries with relative filenames
ipsec attest --add --product "Ubuntu 12.04 LTS i686" --sha1-ima --rel --dir /lib
ipsec attest --add --product "Ubuntu 12.04 LTS i686" --sha1-ima --rel --dir /usr/lib
ipsec attest --add --product "Ubuntu 12.04 LTS i686" --sha1-ima --rel --dir /lib/i386-linux-gnu
      ...
# applications using different versions of Linux system libraries
for file in /usr/lib/firefox/*.so
do
ipsec attest --add --product "Ubuntu 12.04 LTS i686" --sha1-ima --rel --file $file
done
     ...
# kernel modules with relative filenames
for file in `find /lib/modules/3.2.21ima/kernel -name *.ko`
do
ipsec attest --add --product "Ubuntu 12.04 LTS i686" --sha1-ima --rel --file $file
done
```

Fig. 15. Creating the IMA Measurement Reference Database

platform so that reference values have to be stored in the attestation database under a unique platform identifier. We decided to use for this purpose the fingerprint of an AIK certificate bound to the TPM's endorsement key. The AIK certificate must be registered in the database by the system administrator with the command shown in Fig. 16 before the automatic reference measurement enrollment process can started. This prevents a user or a root kit from changing the platform configuration and then requesting a fresh AIK certificate from the Privacy CA so that the modified measurements could be registered anew. As a matter of fact, since TNC clients are not supposed to be anonymous, it would easier if the system administrator set up and manage an AIK CA so that any AIK certificate issued to a TPM and associated user could be registered automatically.

```
ipsec attest --add --owner carol --aik AIK_Cert.der
key 'b772a6730776b9f028e5adfccd40b55c320a13b6' inserted
```

Fig. 16. *attest* registering an AIK public key in the attestion database

Lines 01...08 of Fig. 17 show that the TNC client *carol* receives a TCG "Functional Component Evidence" PA-TNC attribute from the TNC server *moon* which specifies that first the ITA-HSR "Linux IMA - Trusted Platform" functional component is to be measured, followed by the ITA-HSR "Linux IMA - Operation System" component. A TCG "Generate Attestation Evidence" PA-TNC attribute starts the actual evidence gathering. A sensible approach would be to collect the SRTM BIOS evidence (126 measurements on our test system) the first time a client is measured but to omit this functional component as long as the *boot_aggregate* measurement returns the correct hash value over PCRs 0...7.

Lines 09...12 show that the first SRTM BIOS measurement listed in Table II is received by the TNC server *moon* embedded in a TCG "Simple Component Evidence" PA-TNC attribute.

Lines 13...17 show that the last of the 126 SRTM BIOS measurements is transferred in a TCG "Simple Component Evidence" PA-TNC attribute for the functional component ITA-HSR "Linux IMA - Trusted Platform".

Lines 18...23 show that the first IMA runtime measurement listed in Table III which is the *boot_aggregate* value is transferred in a TCG "Simple Component Evidence" PA-TNC attribute as well.

Lines 24...38 show the IMA runtime measurement transfers of an executable file, a dynamic library and a kernel module, respectively, the first attribute including an absolute file name and the latter two basenames only.

Lines 39...47 show the synthetic construction of the PCR Composite hash and the TPM Quote Info structure using software-based shadow PCR registers 0..7 and 10 continuously extended by the Attestation IMV with the measurements received from the Attestation IMC. When the TCG "Simple Evidence Final" PA-TNC attribute arrives containing the TPM Quote signature, the signature can be verified with the help of the synthesized TPM Quote Info structure. If the signature is valid then full trust in all functional component evidence is established.

Line 60 shows that from a total of 1247 IMA measurements (excluding *boot_aggregate*) 1177 matched with Ubuntu 12.04 LTS reference values stored in the attestation database and 70 non-system filenames have not been found in the reference database and thus are ignored.

D. PA-TNC and PB-TNC Bulk Data Flow Control

As mentioned in section II-B3, the largest PB-TNC batch that fits into a PT-EAP payload can have a size of 65'529 octets. This is an inefficient choice though since before TLS protection is applied, a TLS Application Data record is fragmented into TLS Plaintext records of 16'384 octets maximum size by the TLS record layer, so that with the 8 octet AVP header added by the EAP-TTLS encapsulation, a trailing runt fragment of just 7 octets results. Therefore we recommend an optimum PB-TNC batch size of 65'522 octets which fits

```
01 carol charon: 16[TNC] processing PA-TNC message with ID 0x184fd6d0
21 carol charon: 16[TNC] processing PA-TNC attribute type 'TCG/Request Functional Component Evidence' 0x005597/0x00100000
03 carol charon: 16[TNC] processing PA-TNC attribute type 'TCG/Generate Attestation Evidence' 0x005597/0x00200000
04 carol charon: 16[IMC] evidence requested for 2 functional components
05 carol charon: 16[PTS] * ITA-HSR functional component 'Linux IMA' [K.] 'Trusted Platform'
06 carol charon: 16[PTS] loaded bios measurements '/sys/kernel/security/tpm0/binary_bios_measurements' (126 entries)
07 carol charon: 16[PTS] * ITA-HSR functional component 'Linux IMA' [K.] 'Operating System'
08 carol charon: 16[PTS] loaded ima measurements '/sys/kernel/security/ima/binary_runtime_measurements' (1248 entries)

09 moon  charon: 09[TNC] processing PA-TNC attribute type 'TCG/Simple Component Evidence' 0x005597/0x00300000
10 moon  charon: 09[PTS] ITA-HSR functional component 'Linux IMA' [K.] 'Trusted Platform'
11 moon  charon: 09[PTS] measurement time: Jul 30 19:28:11 2012
12 moon  charon: 09[PTS] PCR  0 extended with: 4d:89:4e:ef:0a:e7:cb:12:47:40:df:4f:6c:5c:35:aa:0f:e7:da:e8
        ...
13 moon  charon: 09[TNC] processing PA-TNC attribute type 'TCG/Simple Component Evidence' 0x005597/0x00300000
14 moon  charon: 09[PTS] ITA-HSR functional component 'Linux IMA' [K.] 'Trusted Platform'
15 moon  charon: 09[PTS] measurement time: Jul 30 19:28:11 2012
16 moon  charon: 09[PTS] PCR  5 extended with: 1b:87:00:3b:6c:7d:90:48:37:13:c9:01:00:cc:a3:e6:23:92:b9:bc
17 moon  charon: 09[PTS] checking 126 ITA-HSR 'Linux IMA' BIOS evidence measurements

18 moon  charon: 09[TNC] processing PA-TNC attribute type 'TCG/Simple Component Evidence' 0x005597/0x00300000
19 moon  charon: 09[PTS] ITA-HSR functional component 'Linux IMA' [K.] 'Operating System'
20 moon  charon: 09[PTS] measurement time: Jul 30 19:28:13 2012
21 moon  charon: 09[PTS] PCR 10 extended with: d0:bb:59:e8:3c:37:1b:a6:f3:ad:ad:49:16:19:52:47:86:12:4f:9a
22 moon  charon: 09[PTS] 'boot_aggregate'
23 moon  charon: 09[PTS] checking ITA-HSR 'Linux IMA' boot aggregate evidence measurement

24 moon  charon: 09[TNC] processing PA-TNC attribute type 'TCG/Simple Component Evidence' 0x005597/0x00300000
25 moon  charon: 09[PTS] ITA-HSR functional component 'Linux IMA' [K.] 'Operating System'
26 moon  charon: 09[PTS] measurement time: Jul 30 19:28:13 2012
27 moon  charon: 09[PTS] PCR 10 extended with: 76:18:87:48:45:0a:5c:45:61:24:c9:08:c3:6b:f9:e3:98:c0:8d:11
28 moon  charon: 09[PTS] '/bin/sleep'
29 moon  charon: 09[TNC] processing PA-TNC attribute type 'TCG/Simple Component Evidence' 0x005597/0x00300000
30 moon  charon: 09[PTS] ITA-HSR functional component 'Linux IMA' [K.] 'Operating System'
31 moon  charon: 09[PTS] measurement time: Jul 30 19:28:13 2012
32 moon  charon: 09[PTS] PCR 10 extended with: df:27:e6:45:96:39:11:df:0d:5b:43:40:0a:d7:1c:c2:8f:7f:89:8e
33 moon  charon: 09[PTS] 'ld-2.15.so'
         ...
34 moon  charon: 09[TNC] processing PA-TNC attribute type 'TCG/Simple Component Evidence' 0x005597/0x00300000
35 moon  charon: 09[PTS] ITA-HSR functional component 'Linux IMA' [K.] 'Operating System'
36 moon  charon: 09[PTS] measurement time: Jul 30 19:28:13 2012
37 moon  charon: 09[PTS] PCR 10 extended with: 30:fa:77:07:af:01:a6:70:fc:35:33:86:fc:c9:54:40:e0:11:b0:8b
38 moon  charon: 09[PTS] 'parport.ko'
         ...
39 moon  charon: 04[TNC] processing PA-TNC attribute type 'TCG/Simple Evidence Final' 0x005597/0x00400000
40 moon  charon: 04[PTS] constructed PCR Composite hash: df:07:94:6e:04:72:ee:fe:4d:b0:c3:6e:92:1b:83:dc:e6:49:28:df
41 moon  charon: 04[PTS] constructed TPM Quote Info: => 52 bytes @ 0x8287e2c
42 moon  charon: 04[PTS]    0: 00 36 51 55 54 32 B2 CC 00 38 9D 23 E7 3B 43 D2  .6QUT2...8.#.;C.
43 moon  charon: 04[PTS]   16: 91 88 CE D1 A1 0E 48 F2 B5 54 00 03 FF 04 00 01  ......H..T......
44 moon  charon: 04[PTS]   32: DF 07 94 6E 04 72 EE FE 4D B0 C3 6E 92 1B 83 DC  ...n.r..M..n....
45 moon  charon: 04[PTS]   48: E6 49 28 DF                                      .I(.
46 moon  charon: 04[IMV] received PCR Composite matches constructed one
47 moon  charon: 04[IMV] TPM Quote Info signature verification successful

48 moon  charon: 04[PTS] processed 1247 ITA-HSR 'Linux IMA' file evidence measurements: 1177 ok, 70 unknown, 0 differ, 0 failed
```

Fig. 17. Verification of Simple Component Evidence attributes via database lookups and TPM Quote2 signature

into exactly four TLS Plaintext records of maximum size. If we deduct the PB-TNC batch header (8 octets), the PB-TNC message header (12 octets) and PB-BA message header (12 octets) then the maximum PA-TNC message that can be transported in a single batch has a size of 65'490 octets.

The *max_batch_size* and *max_message_size* parameters of the *tnccs-20* plugin can be used to configure the maximum PB-TNC batch and PA-TNC message sizes as shown in Fig. 10, the default values being 65'522 and 65'490 octets, respectively. With the latest version 1.3 of the IF-IMC interface [11], an IMC can query the maximum PA-TNC message size that is going to be accepted by the TNC client it is attached to via a *GetAttribute* function call.

One argument which speaks against using very large PB-TNC batches is the 4 second default retransmission time of the UDP-based IKEv2 protocol as implemented by strongSwan. About 700 TCG "Simple Component Evidence" PA-TNC attributes fit into the largest possible PA-TNC message. Looking up 700 file measurement values in the attestation database takes longer that 4 seconds so that the IKEv2 client retransmits the last IKE_AUTH request which in turn has to be handled by the IKEv2 server already busy processing the Component Evidence data. Performance tests have shown that 32'754 and 32'722 octets for the PB-TNC batch and PA-TNC message size, respectively, which fit into two maximum TLS Plaintext payloads, are optimum in the sense that no retransmits are triggered because the processing time for each PA-TNC message just stays below the 4 second limit. The resulting log with increased debugging levels is shown in Fig. 18.

Lines 01…17 of Fig. 18 show that a total of 1374 TCG

```
01 carol charon: 16[TNC] creating PA-TNC message with ID 0xf3bc541f
02 carol charon: 16[TNC] creating PA-TNC attribute type 'TCG/Simple Component Evidence' 0x005597/0x00300000
03       charon: last message repeated 380 times
04 carol charon: 16[TNC] creating PB-PA message type 'TCG/PTS' 0x005597/0x00000001

05 carol charon: 16[TNC] creating PA-TNC message with ID 0xa22d16f2
06 carol charon: 16[TNC] creating PA-TNC attribute type 'TCG/Simple Component Evidence' 0x005597/0x00300000
07       charon: last message repeated 346 times
08 carol charon: 16[TNC] creating PB-PA message type 'TCG/PTS' 0x005597/0x00000001

09 carol charon: 16[TNC] creating PA-TNC message with ID 0x0600eabb
10 carol charon: 16[TNC] creating PA-TNC attribute type 'TCG/Simple Component Evidence' 0x005597/0x00300000
11       charon: last message repeated 337 times
12 carol charon: 16[TNC] creating PB-PA message type 'TCG/PTS' 0x005597/0x00000001

13 carol charon: 16[TNC] creating PA-TNC message with ID 0x512bd6ea
14 carol charon: 16[TNC] creating PA-TNC attribute type 'TCG/Simple Component Evidence' 0x005597/0x00300000
15       charon: last message repeated 307 times
16 carol charon: 16[TNC] creating PA-TNC attribute type 'TCG/Simple Evidence Final' 0x005597/0x00400000
17 carol charon: 16[TNC] creating PB-PA message type 'TCG/PTS' 0x005597/0x00000001

18 carol charon: 16[TNC] PB-TNC state transition from 'Client Working' to 'Server Working'
19 carol charon: 16[TNC] creating PB-TNC CDATA batch
20 carol charon: 16[TNC] adding PB-PA message
21 carol charon: 16[TNC] sending PB-TNC CDATA batch (32678 bytes) for Connection ID 1
22 carol charon: 16[TNC] queued 3 PB-TNC messages for next CDATA batch

23 carol charon: 16[IKE] sending tunneled EAP-TTLS AVP [EAP/RES/TNC]
24 carol charon: 16[ENC] generating IKE_AUTH request 16 [ EAP/RES/TTLS ]
25 carol charon: 16[NET] sending packet: from 192.168.0.254[4500] to 192.168.0.1[4500]
      ...
26 carol charon: 03[NET] received packet: from 192.168.0.1[4500] to 192.168.0.254[4500]
27 carol charon: 03[ENC] parsed IKE_AUTH response 47 [ EAP/REQ/TTLS ]
28 carol charon: 03[ENC] generating IKE_AUTH request 48 [ EAP/RES/TTLS ]
29 carol charon: 03[NET] sending packet: from 192.168.0.254[4500] to 192.168.0.1[4500]

30 moon  charon: 09[TNC] no recommendation available yet, sending empty PB-TNC SDATA batch
31 moon  charon: 09[TNC] PB-TNC state transition from 'Server Working' to 'Client Working'
32 moon  charon: 09[TNC] creating PB-TNC SDATA batch
33 moon  charon: 09[TNC] sending PB-TNC SDATA batch (8 bytes) for Connection ID 1
34 moon  charon: 09[IKE] sending tunneled EAP-TTLS AVP [EAP/REQ/TNC]
35 moon  charon: 09[ENC] generating IKE_AUTH response 48 [ EAP/REQ/TTLS ]
36 moon  charon: 09[NET] sending packet: from 192.168.0.1[4500] to 192.168.0.254[4500]

37 carol charon: 02[NET] received packet: from 192.168.0.1[4500] to 192.168.0.254[4500]
38 carol charon: 02[ENC] parsed IKE_AUTH response 48 [ EAP/REQ/TTLS ]
39 carol charon: 02[IKE] received tunneled EAP-TTLS AVP [EAP/REQ/TNC]
40 carol charon: 02[TNC] received TNCCS batch (8 bytes) for Connection ID 1
41 carol charon: 02[TNC] PB-TNC state transition from 'Server Working' to 'Client Working'
42 carol charon: 02[TNC] processing PB-TNC SDATA batch
43 carol charon: 02[TNC] PB-TNC state transition from 'Client Working' to 'Server Working'
44 carol charon: 02[TNC] creating PB-TNC CDATA batch
45 carol charon: 02[TNC] adding PB-PA message
46 carol charon: 02[TNC] sending PB-TNC CDATA batch (32695 bytes) for Connection ID 1
47 carol charon: 02[TNC] queued 2 PB-TNC messages for next CDATA batch
48 carol charon: 02[IKE] sending tunneled EAP-TTLS AVP [EAP/RES/TNC]
49 carol charon: 02[ENC] generating IKE_AUTH request 49 [ EAP/RES/TTLS ]
50 carol charon: 02[NET] sending packet: from 192.168.0.254[4500] to 192.168.0.1[4500]
      ...
51 moon  charon: 04[TNC] IMV 1 provides recommendation 'allow' and evaluation 'compliant'
52 moon  charon: 04[TNC] PB-TNC state transition from 'Server Working' to 'Decided'
53 moon  charon: 04[TNC] creating PB-TNC RESULT batch
54 moon  charon: 04[TNC] adding PB-Assessment-Result message
55 moon  charon: 04[TNC] adding PB-Access-Recommendation message
56 moon  charon: 04[TNC] sending PB-TNC RESULT batch (40 bytes) for Connection ID 1
57 moon  charon: 04[IKE] sending tunneled EAP-TTLS AVP [EAP/REQ/TNC]
58 moon  charon: 04[ENC] generating IKE_AUTH response 143 [ EAP/REQ/TTLS ]
59 moon  charon: 04[NET] sending packet: from 192.168.0.1[4500] to 192.168.0.254[4500]
```

Fig. 18. Bulk data flow control on PA-TNC and PB-TNC layers, fragmentation by EAP-TTLS

”Simple Component Evidence” attributes and one TCG ”Simple Evidence Final” PA-TNC attribute are distributed by the Attestion IMC over four PA-TNC messages the sizes of which will never exceed 32’722 octets.

Lines 18…22 show that a PB-TNC batch with a size of 32’678 bytes carrying the first PA-TNC message encapsulated in a PB-PA message is formed by the TNC client *carol* and sent to the TNC server *moon* whereas the three remaining PB-PA messages are queued for later transmission.

Lines 23…25 show that the 32k EAP-TTLS AVP payload is fragmented by the EAP-TTLS protocol into small fragments of 1024 octets each so that they certainly fit into the UDP datagrams the IKE_AUTH messages are transported in.

Lines 26…29 show that 32 additional IKE_AUTH exchanges are needed to transfer all EAP-TTLS fragments.

Lines 30…36 show the situation on the TNC server *moon* which does not have any PB-TNC messages to send and therefore checks if all IMVs have already provided their recommendation via the IF-IMV [12] interface so that the TNC server could form its final recommendation based on its policy. The PA-TNC finite state machine in Fig. 4 could then advance to the *Decided* state by sending a RESULT batch. This is not the case because the Attestation IMV is still withholding its recommendation since it hasn't received the TCG "Simple Evidence Final" PA-TNC attribute from the Attestation IMC yet. If no final recommendation is possible then the TNC server sends an empty SDATA batch which according to the state machine will allow the TNC client to send the next CDATA batch. In order to prevent endless loops the TNC server will only send an empty SDATA batch if the last received CDATA batch was non-empty. Otherwise a recommendation will be solicited from all pending IMVs and a state change to *Decided* will be enforced.

Lines 37…50 show that the TNC client upon reception of the empty SDATA batch sends the second PB-TNC batch. The dispatch of the two remaining queued PB-PA messages via two additional SDATA/CDATA batch exchanges is omitted in Fig. 18

Lines 51…59 show the recommendation provided by the Attestation IMV after reception of the TCG "Simple Evidence Final" attribute, allowing the verification of the complete functional component evidence measurements via the TPM quote signature. The TNC server then shifts into the *Decided* state by sending a RESULT batch containing a PB-Assessment-Result and a PB-Access-Recommendation message.

V. Conclusion

We have shown that remote attestation based on TPM-certified Linux IMA measurements can be done very efficiently thanks to the new TLV-based PTS, PA-TNC and PB-TNC standard protocols. The problem that the PT-EAP protocol does not support fragmentation was solved by setting the maximum size of both the PB-TNC batches and PA-TNC messages and by optimally allocating the PA-TNC attributes to be transferred to consecutive PA-TNC messages each of which is embedded into a PB-TNC batch of its own.

Acknowledgment

The author would like to thank Sansar Choinyambuu for implementing the PB-TNC [24] and TCG PTS [25] protocols in the C programming languange and integrating them into the strongSwan open source software as part of her HSR master program. I'm also grateful to Gloria Serrao for making me aware of the TCG PTS protocol [17] in the first place as an emerging standard for transporting PTS evidence over an IF-M communication channel and for Carolin Latze and Paul Sangster for shedding light on some particular questions concerning the practical use of PTS functional components.

References

[1] P. Sangster, H. Khosravi, M. Mani, K. Narayan, and J. Tardo, *Network Endpoint Assessment (NEA): Overview and Requirements)*, RFC 5209, June 2008. [Online]. Available: http://tools.ietf.org/html/rfc5209

[2] P. Sangster and K. Narayan, *PA-TNC: A Posture Attribute (PA) Protocol Compatible with Trusted Network Connect (TNC)*, RFC 5792, March 2010. [Online]. Available :http://tools.ietf.org/html/rfc5792

[3] R. Sahita, S. Hanna, R. Hurst, and K. Narayan, *PB-TNC: A Posture Broker (PB) Protocol Compatible with Trusted Network Connect (TNC)*, RFC 5793, March 2010. [Online]. Available: http://tools.ietf.org/html/rfc5793

[4] N. Cam-Winget and P. Sangster, *PT-EAP: Posture Transport (PT) Protocol For EAP Tunnel Methods*, I-D draft-ietf-nea-pt-eap-03, July 2012. [Online]. Available: http://tools.ietf.org/html/draft-ietf-nea-pt-eap

[5] P. Sangster, N. Cam-Winget, and J. Salowey, *PT-TLS: A TCP-based Posture Transport (PT) Protocol*, I-D draft-ietf-nea-pt-tls-07, Aug. 2012. [Online]. Available: http://tools.ietf.org/html/draft-ietf-nea-pt-tls

[6] B. Aboba and P. Calhoun, *RADIUS Support for Extensible Authentication Protocol (EAP)*, RFC 3579, Sep. 2003. [Online]. Available: http://tools.ietf.org/html/rfc3579

[7] P. Funk and S. Blake-Wilson, *EAP Tunneled TLS Authenticated Protocol Version 0 (EAP-TTLSv0)*, RFC 5281, Aug. 2008. [Online]. Available: http://tools.ietf.org/html/rfc5281

[8] N. Cam-Winget, D. McGrew, J. Salowey, and H. Zhou, *The Flexible Authentication via Secure Tunneling EAP Method (EAP-FAST)*, RFC 4851, May 2007. [Online]. Available: http://tools.ietf.org/html/rfc4851

[9] C. Kaufmann, P. Hoffman, Y. Nir, and P. Eronen, *Internet Key Exchange Protocol Version 2 (IKEv2)*, RFC 5996, Sep. 2010. [Online]. Available: http://tools.ietf.org/html/rfc5996

[10] P. Sangster et. al., *TCG TNC IF-M: TLV Binding*, TCG Specification Version 1.0, Rev. 37 , March 10, 2010. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tnc_ifm_tlv_binding_specification

[11] D. Arroyo, S. Hanna, H. Rathor, and P. Sangster, *TCG TNC IF-IMC*, TCG Specification Version 1.3, Rev. 16 , July 10, 2012. (to be published)

[12] D. Arroyo, S. Hanna, H. Rathor, and P. Sangster, *TCG TNC IF-IMV*, TCG Specification Version 1.3, Rev. 11 , July 10, 2012. (to be published)

[13] R. Sahita, S. Hanna, R. Hurst, et. al., *TCG TNC IF-TNCCS: TLV Binding*, TCG Specification Version 2.0, Rev. 16 , Jan. 22, 2010. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tnc_iftnccs_specification

[14] P. Sangster, et. al., *TCG TNC IF-T: Protocol Bindings for Tunneled EAP Methods*, TCG Specification Version 1.1, Rev. 10, May 21, 2007. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tnc_ift_protocol_bindings_for_tunneled_eap_methods_specification

[15] P. Sangster, et. al., *TCG TNC IF-T: Binding to TLS*, TCG Specification Version 2.0, Rev. 4, July 10, 2012. (to be published)

[16] M. Sanchez et al., *TCG TNC IF-PEP: Protocol Bindings for RADIUS*, TCG Specification Version 1.1, Rev. 0.7, Feb. 5, 2007. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tnc_ifpep_protocol_bindings_for_radius_specification

[17] P. Sangster et. al., *TCG Attestation PTS Protocol: Binding to TNC IF-M*, TCG Specification Version 1.0, Rev. 28 , Aug. 24, 2011. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tcg_attestation_pts_protocol_binding_to_tnc_ifm

[18] *TPM Main Specification Level 2*, TCG Specification 1.2, Rev. 116, Mar. 1, 2011. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[19] *TCG Software Stack (TSS)*, TCG Specification 1.2, Jan. 6, 2006. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tcg_software_stack_tss_specification

[20] *Integrity Management Architecture*, TCG Specification 1.0, Rev. 1.0, Nov. 17, 2006 [Online]. Available: http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_architecture_part_ii__integrity_management_version_10

[21] *Platform Trust Services Interface*, TCG Specification 1.0, Rev. 1.0, Nov. 17, 2006 http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_platform_trust_services_interface_specification_ifpts_version_10

[22] S. Bailey, R. Chickering, L. Lorenzin, S. Venema, D. Vigier, et al., *TNC IF-MAP Binding for SOAP*, TCG Specification Version 2.1, Rev. 15, May 7, 2012. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tnc_ifmap_binding_for_soap_specification

[23] D. Safford, M. Zohar, and R. Sailer, *Using IMA for Integrity Measurement and Attestation*, Linux Plumbers Conf. 2009. [Online]. Available: http://linuxplumbersconf.org/2009/slides/David-Stafford-IMA_LPC.pdf

[24] S. Choinyambuu, *A Posture Broker Protocol Compatible with Trusted Network Connect*, HSR Master Project, Jan. 2011. [Online]. Available: http://security.hsr.ch/mse/projects/2011_TNC_Compatible_Posture_Broker_Protocol.pdf

[25] S. Choinyambuu, *TCG Attestation: PTS Protocol Binding to TNC IF-M*, HSR Master Thesis, Feb. 2012. [Online]. Available: http://security.hsr.ch/mse/theses/2012_tcg_attestation_pts_protocol.pdf

[26] TrouSerS download site [Online]. Available: http://sourceforge.net/projects/trousers/

[27] Linux Integrity Measurement Architecture wiki. [Online]. Available: http://sourceforge.net/apps/mediawiki/linux-ima/

[28] Dracut project wiki [Online]. Available: http://dracut.wiki.kernel.org/

[29] SELinux project wiki. [Online]. Available: http://selinuxproject.org/

[30] TNC@FHH project homepage. [Online]. Available: http://trust.inform.fh-hannover.de/joomla/index.php/projects/tncfhh

[31] Open Platform Trust Services (OpenPTS) project homepage. [Online]. Available: http://sourceforge.jp/projects/openpts/wiki/

[32] strongSwan project homepage. [Online]. Available: http://www.strongswan.org/

[33] strongSwan Trusted Network Connect wiki. [Online]. Available: http://www.strongswan.org/tnc/

[34] IETF Network Endpoint Assessment Working Group homepage. [Online]. Available: http://datatracker.ietf.org/wg/nea/

[35] Trusted Computing Group homepage. [Online]. Available: http://www.trustedcomputinggroup.org/