TPM Protection Against Lying Endpoints

This document describes how the Trusted Platform Module (TPM) can be used with PT-EAP to detect and protect against lying endpoints. The reader is assumed to have already read and understood the document titled “Protecting NEA Against Asokan Attacks with Cryptographic Bindings”.

Background on the TPM

The Trusted Platform Module (TPM) is a hardware security function or chip whose capabilities and interfaces are defined in specifications [2] from the Trusted Computing Group (TCG). Multiple manufacturers build TPMs and they are included in virtually all business laptops. The TPM has several capabilities such as secure key storage. However, the capability most essential to the Asokan attack is the ability to collect and generate a verifiable signed report on the integrity of an endpoint’s security-critical software (the Trusted Computing Base). This countermeasure can be used to determine the endpoint’s posture with much greater certainty than a software-only approach. However, if the NEA PT protocol does not include protection against Asokan attacks (described in the other document), the TPM’s protections can be defeated by mounting an Asokan attack.

Normal TPM-Based Integrity Check

Before looking at the Asokan attack in any detail, let’s look at how a TCG Trusted Platform [3] can provide integrity information about an endpoint. As the core of the Trusted Platform is the TPM which provides a hardware isolated environment (frequently inside a chip) where sensitive information like private keys can be stored. The TPM also provides operations allowing the Trusted Computing Base (TCB) to “extend” hashes of information about the local system including programs about to execute, configurations, firmware and alike into a set of TPM housed registers. These extended hashes are stored inside the TPM, so are protected from modification from software (except subsequent extends by TCB). The extend operation hashes the current value in a TPM register with the value provided so the new register value factors in the previous and new value into the result. This has the effect that each TPM register contains an aggregate hash of all the values extended into the register since the boot.

As a Trusted Platform boots, a sequence of trusted software loaders are started and each is responsible to measuring (hashing) any programs the loader starts and storing the result into the TPM. As a result the TPM’s registers will contain a set of aggregated hashes of all the software on the endpoint during the boot and subsequent use. An equivalent integrity log is kept that includes details about the software run and its individual hash. Optionally, the TCB loaders could have local policy identifying hashes of software that are allowed to run on the system and prevent processes with other hashes from being started.

Now the endpoint’s TPM contains registers reflecting the aggregate hashes of the operational state of the system and a set of public/private key pairs in protected storage. When an authorized remote system like Server1 decides to assess the trustworthiness of the endpoint, it can use an “attestation” protocol to fetch verifiable information about the endpoint. The endpoint’s TCB includes a service that receives attestation requests (must like NEA’s PA-TNC messages) and responds with TPM register values signed using a TPM’s private key (for more information on TPM keys see the TCG Credentials specification [5]). The remote challenger can obtain the TPM’s X.509 certificate associated with the TPM’s signing private key, a signed structure containing the requested register values and a portion of the systems integrity log with the requested details about the endpoint. At that point the challenger can verify the TPM signed structure was created by a trustworthy TPM (using its protected private key) and then compare the results against policy. Note that malware on the endpoint will be unable to access the TPM’s private signing key so can not forget the TPM signed report nor can the malware change the TPM registers to a desired value (that might be used to hide its existence) since only the extend operation is available.
The attestation protocol can be used during a NEA assessment to determine the trustworthiness of the NEA architecture components and their dependencies on the underlying system. As a result the remote challenger can determine whether the responses provided by the NEA Client are valid and not under the influence of malware. Similarly, the challenger can determine whether the endpoint is subject to lying or sending incorrect information during the assessment. However, even with the described strong security protections, it is possible that an intermediary using the Asokan attack might be able to replay information from another Trusted Platform inside the EAP tunnel so an additional protection is required to bind the inner method information to the authenticated outer tunnel.

Example Asokan Attack When TCG Technology in Use

This section describes an example of the Asokan, Niemi and Nyberg attack against a network admission where the use of TPM and the TCG Trusted Platform is required to join the network. Trusted Platform includes an enabled TPM and a TCB that measures all software as it is loaded so Server1 can assess what software is running on the system and detect malware.

In this example, the endpoint is called Laptop1 and the NEA Server is referred to as Server1 (see figure 1.) Laptop1 contains an operating Trusted Platform including a TPM and is wishing to gain access to the company intranet via an Access Point (AP1). Server1 requires user authentication using PEAP, and verification of the NEA Client software and its dependencies using the Trusted Platform’s TPM. Therefore, if Laptop1 gets compromised, it cannot obtain network access since the TPM signed (set of registers) report will not match the Server1 expected values.

Next, let’s assume that Laptop1 does get compromised and can be controlled remotely by the attacker (maybe over the laptop’s EV-DO card independent of the WLAN NIC used in this example.) So now sometime after the Trusted Platform has performed its early platform measurements the attacker’s malware is loaded and can communicate with the attacker. Now the attacker sets up his own equipment, Laptop2, AP2, and Server2 (see figure 2 below) on a stub network to aid his attack. Laptop2 is configured to match a “good” configuration that would be accepted by Server1 and even includes an enabled TPM and Trusted Platform. However, Server2 contains malware to aid the attacker obtain the TPM signed report from Laptop2.
Next, the honest (unaware of the malware) user of Laptop1 tries to connect to the network, see figure 3 for flow diagram. Eventually, PEAP is started and the user is authenticated (steps 1-2). At the same time, the attacker uses Laptop2 and starts PEAP with his own Server2, and does user authentication on the stub network (steps 3-4).
Next, the NEA protocols start within the authenticated PEAP tunnel. At this point Server2 begins to interact with the compromised Laptop1. Server1 sends a NEA PT EAP request (either using EAP-TNC or EAP-NEA-TLV) request to Laptop1 (step 5), which being compromised, forwards it to Server2 and eventually Laptop2 (steps 6 and 7). Laptop2 creates a response including a TPM signed report including hashes of the NEA Client and its dependencies on Laptop2. The TPM signed report is sent to Server2 that forwards it to Laptop1 who presents it to Server1 as a description of the contents of Laptop1 (steps 8-10). Steps 5-10 can be repeated as often as required by Server1.

Eventually the exchanges will succeed because Server1 is unable to determine that the TPM signed report describing Laptop2 received in step 10 does not describe the same system that participated in the outer authenticated PEAP session (Laptop1), thus compromised Laptop1 gets access to the network despite the requirement for TPM based integrity reporting. Therefore the key to the countermeasure is to cryptographically verify that the system performing the authenticated outer EAP method is the same system providing the assessment attributes and TPM signed report.

**Using Diffie-Hellman Pre-Negotiation as a Countermeasures**

One approach to ensuring that the active MiTM is not able to undetectably forward the TPM signed report from another system is to include a shared secret in the report. The Diffie-Hellman Pre-Negotiation (D-H PN) is an exchange that occurs at the start of the inner EAP method with the goal of establishing a fresh shared secret between the two active parties in the assessment that is included in the TPM signed report.
The D-H PN [4] exchanges D-H public values and nonces to assure freshness of the session (for replay detection.) The resulting secret information derived from the exchanged nonces and D-H public values are known only to the two parties performing the D-H PN. These secrets are used to derive a shared secret that is included in the TPM signed report and another value (also derived based upon the contents of the NEA attribute exchange) that is exported to the outer EAP method for subsequent mixing with the outer methods session keys. This has the effect of cryptographically binding the TPM signed report and the NEA attribute exchanges with the outer EAP method authentication because the creator of the TPM signed report and the NEA attributes must know the D-H PN secrets so must be the same entity.

At the conclusion of the NEA EAP protocol including the new D-H pre-negotiation, a derivative (hash) of the established secret keys and the exchanged messages during the assessment is exported to the outer method that cryptographically mixes the keys into those used to protect the tunnel in order to address this attack. To ensure that a MiTM is not present, the EAP outer method must verify both parties had knowledge of all the keys, so would perform a final roundtrip exchange using the new mixed keys. The NEA Server must also check the TPM signed reports received from the endpoint to make sure they indicate that the secret derived from the D-H PN was included within the TPM signed report. If the endpoint side of the tunnel is able to complete the exchange (involving use of the mixed key) and the TPM signed report sent by the endpoint includes the use of the secret derived from the D-H PN, the endpoint has proven that it was not a MiTM and that it reported its local state. While an active MitM (Laptop1) could have figured out the D-H PN secrets, it would not be able to cause the secret values to be used in the TPM signed report produced by another clean system (Laptop2) because the clean system would negotiate its own D-H PN unique secrets with Server2, since both parties contribute nonces and D-H public keys Laptop2’s secret value would not be controllable by Server2 or Laptop1.

If an active MiTM (Laptop1) takes part in the D-H PN it establishes a shared values with Server1 that aren’t actually known by Laptop2 so can’t be included in Laptop2’s TPM signed report. The different value used by Laptop2 is detectable by Server1 since it’s included in the TPM signed report.

If a MiTM (Laptop1) just forwarded the D-H PN protocol over a tunnel to Server2 so that clean Laptop2 and Server1 were using the same D-H values and nonces, Laptop1 would be unable to determine the established resulting secret since it lacks knowledge of any D-H private values and would be unable to complete (export) the final keys to the outer EAP tunnel exchange so again would be detected by Server1.

References


