CHAPTER 3

Getting It to Work: Provisioning Intel® TXT

Now that we have an understanding of what it means for a platform to implement Intel® Trusted Execution Technology, the next step is to put that technology into operation. This chapter describes the general steps that a platform owner (typically an IT organization) must take to enable Intel® TXT, and then identifies the steps to condition the platform to realize its values.

Note Intel provides a list of platforms that have demonstrated compliance with Intel Trusted Execution Technology. This list can be found at `www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-server-platforms-matrix.html`, or just go to `Intel.com` and search for Intel Trusted Execution Technology.

Provisioning a New Platform

Unfortunately, the default configuration for new platforms requires the owner to “opt in” to using Intel Trusted Execution Technology. Thus, the owner must explicitly enable Intel TXT, which includes a series of steps that must be performed to take advantage of the technology. These steps are necessary to protect owners that are not yet ready to take advantage of Intel TXT, such as protection from malicious software that could otherwise hijack the TPM and Intel TXT resources. The steps to provision the platform are as follows:

1. Enable and activate the TPM.
2. Enable supporting technology.
3. Enable Intel TXT.
4. Provision the TPM.
   a. Establish TPM ownership.
   b. Create a platform owner policy index.
5. Create a platform owner launch control policy.
6. Install a trusted operating system.

BIOS Setup

The first steps are accomplished via the BIOS Setup menu. We wish we could provide explicit instructions, but each platform manufacturer has their own style for BIOS menus, so the actual steps that need to be performed will vary from manufacturer to manufacturer, and possibly even between platform types. But let's walk through the process.

Enable and Activate the Trusted Platform Module (TPM)

This step is typically accomplished via the *Security* tab of the BIOS setup menu. Most BIOS implementations require setting the BIOS's *administrator password* before the BIOS will perform TPM operations. In some cases, the menu might allow you to select the "TPM enable" operation, but it does not perform that operation if the password has not been set. So it is best to set BIOS passwords before attempting any TPM operations. Besides, it is always good practice to set the BIOS password to help protect against unauthorized changes.

Unfortunately, the BIOS will not honor TPM commands at the same time that the password is set. So the actual steps will be to set the password, save and exit, and then reenter the BIOS setup to enable the TPM.

The BIOS menu might show *TPM status*. TPM status could take any of the four combinations of *Enabled/Disabled* and *Activated/Deactivated*. Most BIOS menus provide a "TPM command" that allows selection of the following:

- No operation
- Clear (or Clear Ownership)
- Enable/Turn On/Activate TPM
- Disable/Turn Off /Inactivate TPM

The terms will vary, but the effect is the same. For example, on a Dell PowerEdge R410, you will find this control on the *Security* tab listed as *TPM Status … Disabled/Deactivated*. You will need to select *Enable/Activate TPM*. For this example, you also need to set *TPM Security* to *On with Pre-boot Measurements*. Next select *save and exit*, which causes the platform to reset. The platform has to reset for the TPM state to change—this is a security feature of the TPM. If the BIOS does have separate *Enable* and *Activate* controls, then you will need to perform the following steps: Enable TPM, save and exit, Activate TPM, save and exit.

All of these platform resets can be time consuming, but if they are skipped, it is possible that the action is ignored. If performed properly, the TPM status should indicate that the TPM is enabled and activated. There is an effort underway to streamline/automate enabling the TPM, referred to as the *physical presence interface* that allows system software to request these operations and have the BIOS prompt the operator to "accept or reject."

Enable Supporting Technology

Intel TXT requires that Intel® Virtualization Technology (Intel® VT) is enabled. The steps to enabling Intel VT also vary by platform manufacturer. Some platforms might have a single control under the *Processor* tab to enable virtualization technology (this is the case with the Dell PowerEdge 410), or the platform might provide a control to enable VMX (virtualization instructions) under the *Processor* tab and another control to enable Intel Virtualization Technology for Directed I/O (Intel VT-d) on a separate chipset or I/O tab. In this case, both controls must be enabled. Various virtualization options (such as enabling SR-IOV) can be set as desired.

Intel TXT requires that VMX is enabled so that the system software can use it for isolation and protection of software processes (preventing them from accessing private, sensitive, or protected data) and that Intel VT-d is enabled so that the system software can use it to prevent hardware devices from accessing private, sensitive, or protected data.

Enabling Intel® TXT

In many cases, this step must be done after enabling the TPM and enabling Intel Virtualization Technology. Otherwise, the option to enable Intel TXT might be hidden or disabled.

In most cases, this control is found on the *Processor* tab or on the *Security* tab. Not everyone titles that control "Intel TXT" or has the same requirement. Regardless of what it is called, set it to enable *Intel TXT measured boot*, and then save and exit.

For our example, on a Dell PowerEdge R410, you will find this control on the *Security* tab listed as *Intel TXT* (see Figure 3-1). However, before you can enable it, you need to set *TPM Security* to *On with Pre-boot Measurements*. You can find the requirements by highlighting the *Intel TXT* control and pressing F1.

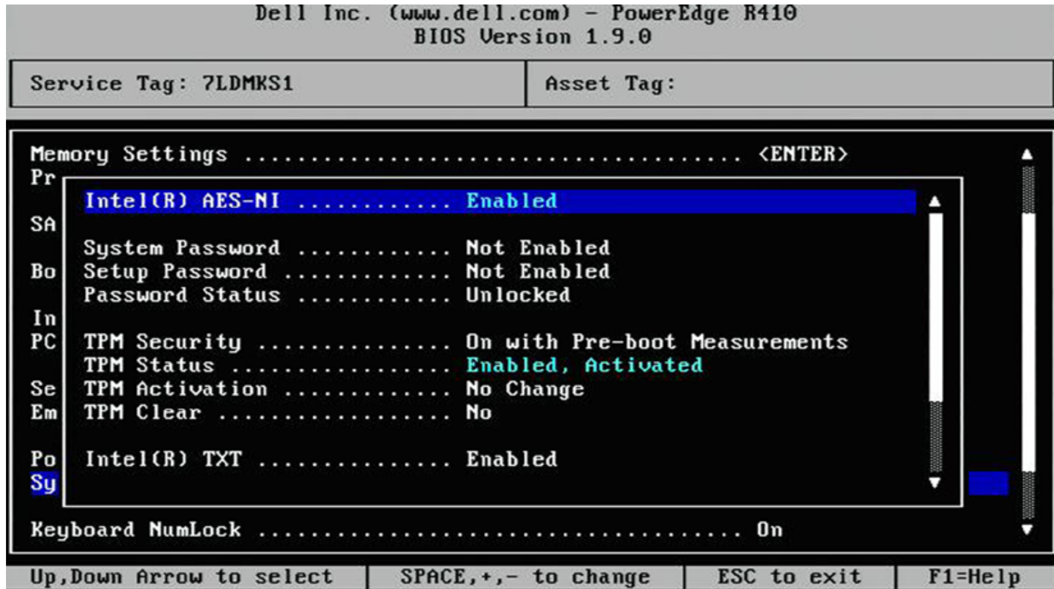

Figure 3-1. *BIOS security screen*

Summary of BIOS Setup

The following summarizes the BIOS setup:

1. Set the BIOS password.
2. Save and exit.
3. Enable/activate TPM.
4. Save and exit.
5. Enable virtualization (VMX and Intel VT-d).
6. May need to save and exit.
7. Enable Intel TXT.
8. Save and exit.

Automating BIOS Provisioning

As mentioned earlier, there is an effort underway to automate enabling of the TPM, called the *physical presence interface*. Intel is publishing an extension to that capability that allows system software to also request the enabling of Intel Virtualization Technology and Intel Trusted Execution Technology at the same time. With the physical presence interface, the operator would see a single prompt to accept or reject the request to enable everything needed to enable Intel TXT. I bring this up because the manual steps to enable Intel TXT are a real pain, not to mention very time consuming (multiple resets, and because more powerful servers take longer to boot, more time). But as the technology matures, it will become more user-friendly. It's all part of the *crawl-walk-run* paradigm.

The other good news is that there is at least one third-party company producing "*How To"* automation guides for a number of the more popular platforms, and another is working on providing automated provisioning scripts as part of its platform management suite.

Establish TPM Ownership

This step has significant dependencies on the particular host OS/VMM that is to be installed. In some cases, the OS/VMM installation must perform this step. Having the OS/VMM installation automatically perform this step definitely makes it easier, but does restrict the platform to a single host operating system. In the past, this might have been a problem because it could prevent repurposing[1] of the platform. However, virtualization technology (virtual machines) provides a better solution for managing time-varying workloads, so this should not be an issue.

What Is TPM Ownership? Why Is This Important?

TPM ownership means that an authority (the OS or the datacenter) has set a password that must be used to allocate TPM resources. An entity that knows that password is considered the TPM owner and is able to specify access rights for TPM resources. In addition, certain TPM commands can only be issued by the TPM owner.

When a new platform arrives, the TPM is in an unowned state. Before any TPM objects and resources can be allocated, a TPM owner must be established. This is accomplished via a *Take Ownership* command issued to the TPM, which provides the TPM with a secret value that is only known by the TPM and the TPM owner. In essence, this is a password. It is a strong password or, more likely, the hash of a password/phrase (since the "secret" is 20 bytes—the size of a SHA-1 hash digest). Once ownership has been established, the TPM's *Take Ownership* operation is no longer available.

Thus, the first entity that successfully performs the *Take Ownership* operation becomes the TPM owner. This is one of the reasons that the TPM is initially in a disabled/inactive state—to prevent malicious software from taking ownership. Thus, you should only enable the TPM immediately before you are ready to establish ownership.

Anyone that knows the secret value (or can generate it by hashing the password/passphrase) is considered the TPM owner, and thus can perform owner authorized operations on the TPM. The role of the TPM owner will become more apparent as we discuss tools to set launch control policy, allocate TPM NVRAM, create keys, and so on.

How to Establish TPM Ownership

Some OS/VMMs require that they perform the take ownership operation (typically during the installation process or the first boot) and others expect the datacenter to establish ownership. Why is this? Well, the TPM and Intel TXT were architected with flexibility in mind. Thus, various OSV/VMM vendors make use of the TPM in different ways. To better understand why, let's look at three TPM management models. The following models are all valid and illustrate various considerations for establishing TPM ownership and managing the TPM.

---

[1]Repurposing of a system is where the platform reboots for a different purpose. One example is where a server used during normal working hours as a file server would be repurposed at night to serve as a backup server.

Pass-Through TPM Model

Figure 3-2 illustrates a simple management model in which a TPM management utility resides as an application on the server. The host OS/VMM simply provides TPM access to the application, and the management utility establishes a session with the TPM. In this scenario (for performing owner authorized operations), the utility prompts the user for the TPM password and uses it to establish the session with the TPM. It is this same TPM utility that initially establishes TPM ownership and later performs TPM operations that require owner authorization (such as setting owner policy). Many Linux operating systems support this model.

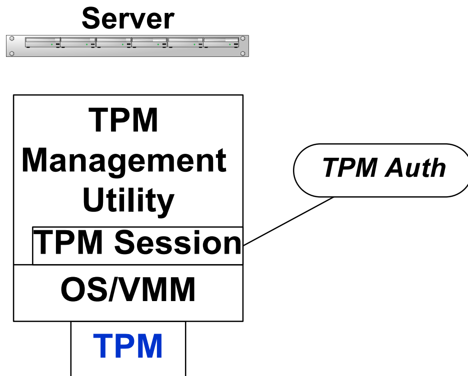

Figure 3-2. *Local TPM management*

The TPM management application can be supplied by the OS/VMM vendor or by a third party. If the OS/VMM or other entity needs to perform authorized actions, then they will need to prompt the user for the TPM password.

Remote Pass-Through TPM Model

Figure 3-3 illustrates an adaptation of the locally managed model where the TPM management utility is executed on a remote management console. Because the TPM command/response is communicated across a network, the management utility would create a secure transport session with the TPM if it needs to protect the data in the command/response while in flight. Other than the use of a secure channel, this model functions the same as the local management model.

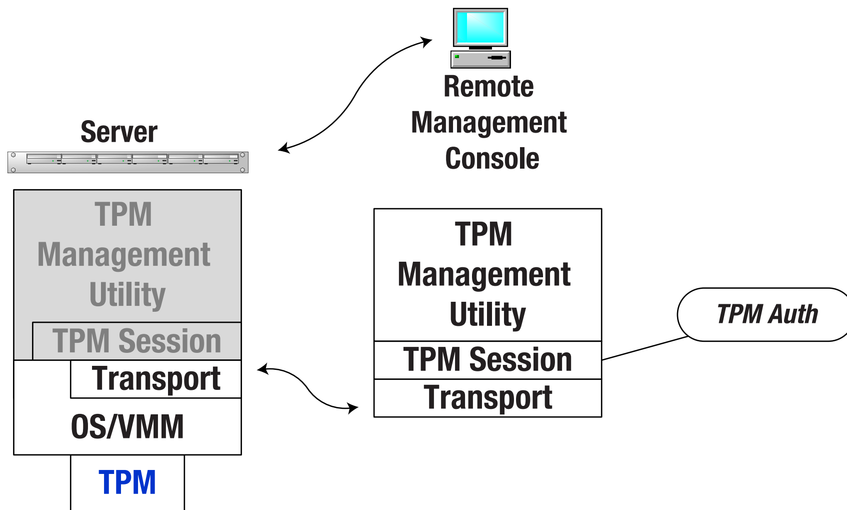

Figure 3-3. *Remote TPM management*

This model does not preclude executing the TPM management utility locally, and any operating system that supports local pass-through should also be able to support this model.

Management Server Model

Figure 3-4 illustrates a model where TPM management is integrated with other server management utilities in a central management server. Unlike the previous models, the management server would be the TPM owner—that is, perform *TPM Take Ownership* and therefore track the TPM authorization value for each of the compute servers. This model supports each TPM having a unique TPM owner authorization value or all servers in the managed pool might have the same owner authorization value. Either way, (other than the TPMs themselves) only the central management server knows the authorization values. VMware ESXi supports this style of TPM management.

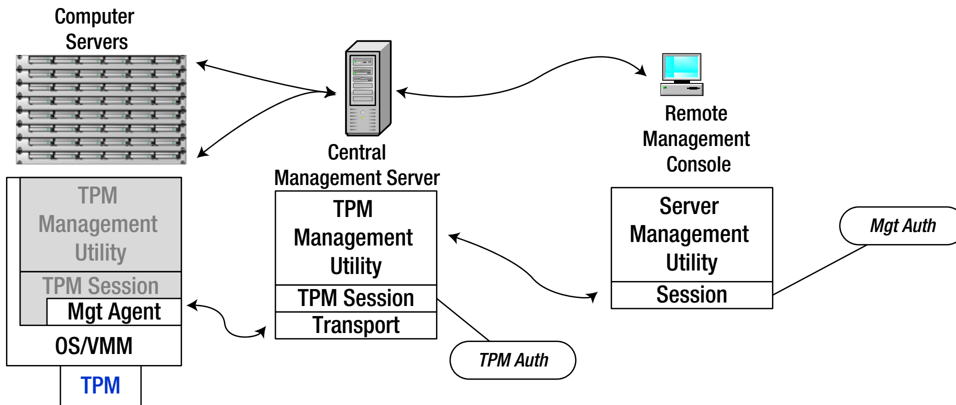

Figure 3-4. *Central management server*

A central management server is able to enforce more granular control over which TPM operations are authorized by a particular client.

For example, the datacenter manager might have complete access to all TPM commands, whereas other management support personnel are limited to a subset that fits their particular jobs. Clients that use the management service to manage their applications have little or no TPM management capability.

One main benefit of the central management server is that each client of the management server only needs to know his or her own password to log into the management server and does not have to track TPM passwords.

Protecting Authorization Values

It should be noted that the TPM secret authorization value is never transmitted as part of the command data. Rather, the initiator of a command that requires authentication proves it knows the authentication value by computing the hash of the TPM command data, plus a nonce provided by the TPM, plus the TPM authorization value. The initiator sends that hash result with the TPM command, as illustrated in Figure 3-5. As we learned in the previous chapter, this is the Hash Method of Authentication (HMAC). The TPM authenticates that the initiator knows the correct authorization value by performing a hash of the command data, expected nonce, and the TPM's copy of the Owner Authorization value; this result must match the hash sent with the command or else the TPM rejects the command. Thus it is not possible to derive the TPM authorization value by monitoring the communication.

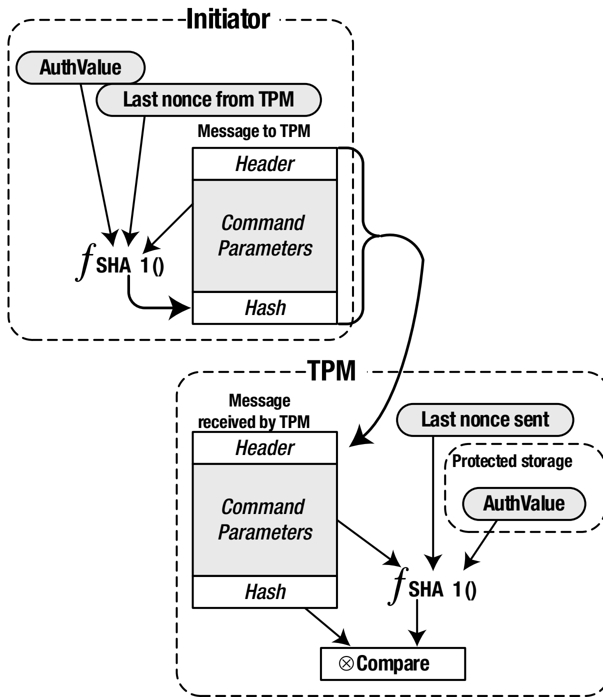

Figure 3-5. *TPM command authentication process*

The nonce prevents replay attacks because the nonce randomizes the HMAC hash value. Each response from the TPM provides a new nonce that the initiator must use in the next command. Thus the exact same command has a different expected HMAC value each time it is sent. The TPM response is also protected by the use of a different nonce (one that the initiator includes in the command data), which the TPM then uses to generate the response HMAC value. In summary, the HMAC in the command proves to the TPM that the initiator is authorized and that the message had not been altered. The HMAC in the response proves to the initiator that the response came from the TPM and had not been altered.

Perhaps you were wondering how the *Take Ownership* command works without sending the owner authorization value in the clear since the TPM does not yet know the TPM secret password. To protect the secret value from being snooped, the TPM provides the initiator with the public portion of an asymmetric *endorsement key* and the prospective TPM owner uses it to encrypt the secret value when it sends the *Take Ownership* command. The endorsement key is a unique RSA encryption key pair created by the TPM vendor for the TPM at the time the TPM was manufactured. The user can read the public encryption key, but only the TPM knows the private decryption key. Thus only the TPM can decrypt the encrypted value. The TPM then uses that decrypted value to authenticate the *Take Ownership* command and all subsequent commands that require owner authorization.

A couple of points of interest: the TPM owner can gracefully change the owner authorization value using a TPM *Change Owner Auth* command. Additionally, in case the owner disposes of the platform or forgets the TPM owner authorization value, the BIOS does have the means to clear the TPM's owner authorization value (and storage root key) so that the *Take Ownership* process is again enabled. However, this action destroys all objects (such as TPM NVRAM indexes) that require owner authorization. Likewise, any keys or external entities that were sealed to the TPM are rendered useless. So the *BIOS Clear Ownership* action essentially returns the TPM to its initial state and cannot be used to recover a lost TPM password. Thus, if you lose the TPM password, the only resolution is to clear ownership and start from scratch. The TPM owner also has a command that is equivalent to the *BIOS Clear Ownership*, except it does not require resetting the platform and entering the BIOS setup menu. Either one is an excellent way for a datacenter manager to revoke all keys and objects protected by the TPM before selling, transferring, or otherwise disposing of a server platform.

Install a Trusted Host Operating System

There is no change in how the OS/VMM is installed, but you will need to know the TPM ownership requirement of the particular OS/VMM being installed. Many OS/VMMs support “zero-touch” automatic or scripted installs, including Intel TXT settings.

Depending on the OS/VMM, you might have to modify the startup code to instruct the OS to perform a measured launch (see the following “Linux Example”). This assures that the first OS code loaded is the Trusted Boot (called TBOOT) code that sets up the platform for Intel TXT and initiates the measured launch before loading the OS kernel. This was true for most Linux operating systems when I started writing this book, but I have been informed that many of the major Linux OSVs are now making it simple by providing the modified grub files. So check with your favorite OSV for the requirements to enable Intel TXT.

VMware ESXi Example

VMware ESXi requires that the TPM be in an unowned state or (for the case of reinstalling) ownership previously established by a VMware central management server). The install program automatically installs its Intel TXT components. It is the runtime operation that checks if the TPM is present and if the platform supports Intel TXT.

Each time a VMware ESXi server boots, it checks if the TPM is enabled but not owned. If so, it performs the take ownership operation to establish itself as the TPM owner. After that, it checks if Intel TXT is enabled, and if it is, the VMM performs a measured launch and enters secure mode. Note that this allows you to install ESXi before enabling the TPM and/or enabling Intel TXT—but I still recommend that you follow the prescribed order.

Thus a new server is ready for loading ESXi after performing the BIOS setup steps (see “BIOS Setup” at the beginning of this chapter). If you are reprovisioning a server (that is, installing ESXi after installing a different OS/VMM), then you need to execute the Clear TPM Ownership action from the BIOS menu and re-enable the TPM before installing ESXi.

Linux Example (Ubuntu)

After installing the OS, follow these steps to enable the Intel TXT–measured launch. As noted earlier, these steps might no longer be required, depending on the OS version.

1. Install TBOOT.
   a. Install modules.

      ```
      $ apt-get install tboot
      ```

   b. Change to the /boot directory.
   c. Verify that tboot.gz is there.
2. Copy SINIT ACM to the /boot directory.
   - For 5600 series platforms, you will need to download SINIT ACM (X5600_SINIT_16.BIN) from http://software.intel.com/en-us/articles/intel-trusted-execution-technology/ (scroll down to “Server Platforms”).
   - For newer platforms, the BIOS image contains the SINIT ACM; however, you may download the latest SINIT ACM from that same web site and the OS will use the most recent.

3. Install and verify the TCG software stack.

   a. Install.

   ```
   $ apt-get install trousers
   $ apt-get install trousers-dbg
   $ apt-get update
   ```

   b. Verify them by running TCSD daemon.

   ```
   $ tcsd
   ```

4. Edit the GRUB menu. It is best to copy a menu item and then alter it. To modify the following grub menu item:

   ```
   title    Ubuntu 11.10, kernel 3.0.0-12-server
   kernel   /boot/vmlinuz-3.0.0-12-server root=/dev/sda1 ro quiet splash
   initrd   /boot/initrd.img-3.0.0-server
   quiet
   ```

   a. Modify the *title* to indicate Trusted Boot.

   b. Change *kernel* and *initrd* lines to be modules.

   c. Add "`kernel    /boot/tboot.gz logging=memory`" before those module definitions.

   d. Add the SINIT module for the platform after those module definitions; for example:

   ```
   Module    /boot/X5600_SINIT_16.BIN
   ```

   e. The modified grub menu item should look like this:

   ```
   title    Trusted Boot - Ubuntu 11.10, kernel 3.0.0-12-server
   kernel   /boot/tboot.gz logging=memory
   module   /boot/vmlinuz-3.0.0-12-server root=/dev/sda1 ro quiet splash
   module   /boot/initrd.img-3.0.0-server
   module   /boot/X5600_SINIT_16.BIN
   ```

5. Verify that the platform performs the measured launch.

   a. Reboot the server.

   ```
   $ reboot
   ```

   b. Select the new menu item from the grub menu.

   c. Run TXT-stat; it should show TXT Measured Launch = TRUE.

   ```
   $ tcsd
   $ txt-stat
   ```

   d. Query the device file to read the PCRs (PCRs 17, 18, and 19 will be populated. If not, those PCRs' values will contain the default value of FF FF FF … FF).

   ```
   $ cat /sys/class/misc/tpm0/device/pcrs
   ```

If these steps were successful, the platfrom performed the measured launch and PCRs 17–19 contain the SINIT and OS measurements.

Create Platform Owner's Launch Control Policy

The launch control policy (LCP) is the first owner-controlled use of attestation. That is, the first chance that the owner has to make a trust decision based on attestation measurements made by Intel TXT. The launch control policy allows the platform owner to control which platform configurations and which operating systems are considered trusted.

The next chapter provides a guide to help you in choosing a launch control policy that best suits the datacenter. In this chapter, we concentrate on what launch control policy does, why it is important, and what impact it has.

Before we get down to the details, let me inject some insight. There are some subjective questions that one must answer and your opinion outweighs mine. Those questions are

- Is remote attestation needed if there is a strong launch control policy?
- Is a launch control policy needed if there is a remote attestation?
- Are appropriate tools available to create and manage the policies?

The simplest policy is "ANY" because it defers the policy decision to a later time and allows any platform configuration and any OS. I predict that in the early stages, datacenter managers will find it prudent to select a launch control policy of ANY because it is less complicated and the tools for managing a more complex policy are either not available or not very intuitive. In addition, managing a complex policy can be problematic. I expect this to change as the technology matures and datacenters have better tools and more experience. The more paranoid you are, the faster you will want to transition to a stronger policy. We will discuss these topics in the next chapter, but for now, let's take an objective look at launch control policy.

How It Works

Let's take a closer look at the measured launch process illustrated in Figure 3-6. In particular, the TBOOT module will setup the platform for a measured launch and then invoke the SENTER command.

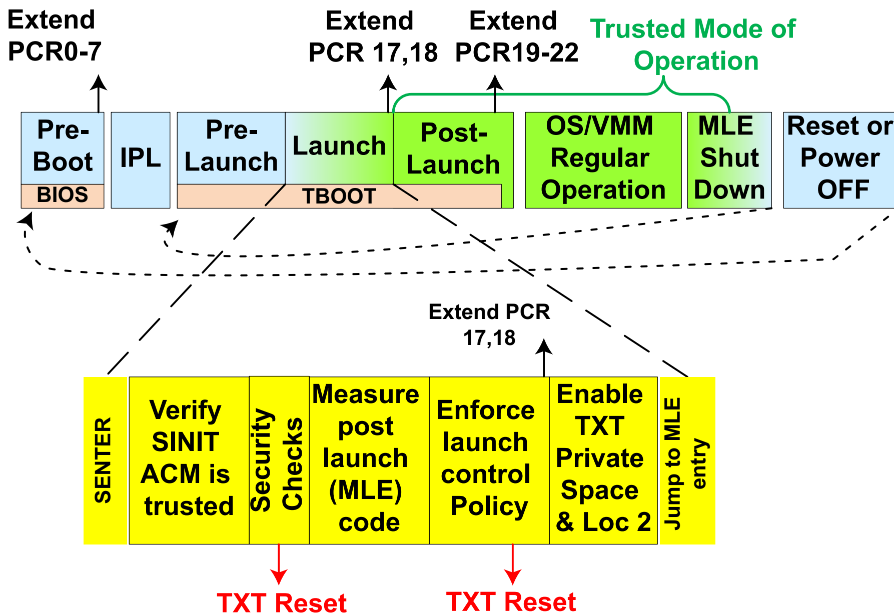

Figure 3-6. *Measured launch timeline*

For the OS/VMM to do a measured launch, it places its trusted post-launch code (referred to as *measured launch environment*, or MLE code) in contiguous memory, fills in a table that indicates where the code resides, and where the platform owner's LCP data structure resides. It then invokes the GetSec SENTER processor instruction.

The processor microcode loads the SINIT ACM (the authenticated code module provided by and signed by Intel) into special protected memory inside the processor, where it validates the ACM and then executes it. Part of the ACM contains the Launch Control Policy Engine that processes the launch control policy.

A launch control policy actually consists of two physical parts, as illustrated in Figure 3-7. Those parts are as follows:

- *NV Policy Data*: A small piece stored in a well-known TPM NVRAM location that uses TPM protections to prevent unauthorized alteration.
- *Policy Data Structure*: Contains a variable amount of policy information (lists of known good measurements). The policy data structure is protected from unauthorized alteration by storing the hash measurement of that structure in the TPM NV policy data.

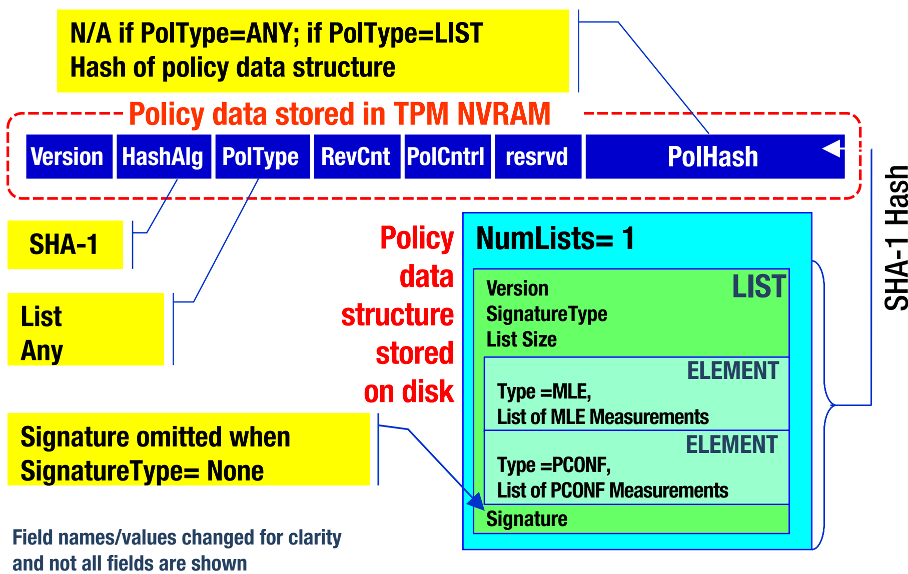

Figure 3-7. *Launch control policy*

The ACM validates the integrity of the policy data structure by measuring it and comparing its measurement to the one stored in the NV policy data. Any attempt by malicious software to modify or replace the platform owner's policy will be detected by the ACM, which results in a *TXT reset*. A TXT reset means that the platform has detected a threat and disables further attempts to perform a measured launch (at least until the platform is power-cycled). Only if the policy data structure is valid will the ACM continue to process the launch control policy.

What LCP Does

Essentially, the LCP is a go/no go decision that determines if the OS/VMM is permitted to do a measured launch (as illustrated in Figure 3-8). This policy is evaluated at the time the OS/VMM initiates a measured launch. There are several parts to the policy decision:

- Specifying which platform configurations are trusted (PCONF policy)
- Specifying which OS/VMMs are trusted (MLE policy)
- Specifying what ACM versions are trusted (SINIT policy)
- Overriding the platform's default policy set by the platform supplier

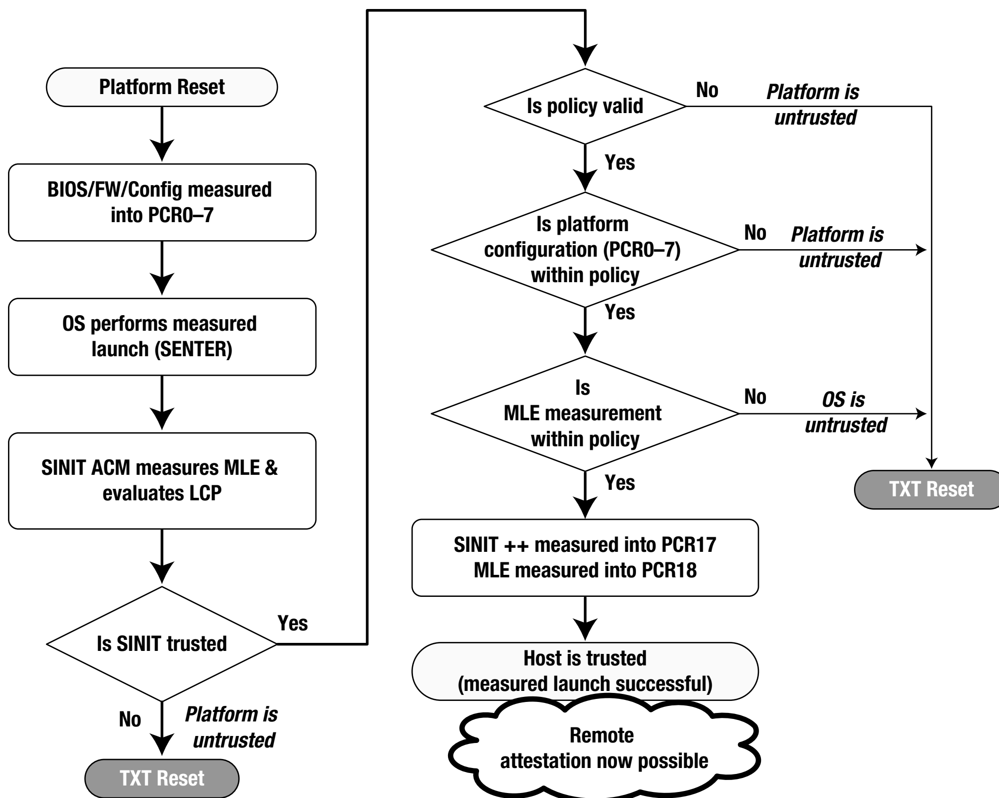

Figure 3-8. *Launch control policy flow*

A tool for creating the policy is illustrated in Figure 3-9. The user has the option to set the SINIT policy by specifying the minimum SINIT version. Under the CONTROL options, the user can also select whether to include SINIT capabilities in the PCR17 measurement. This will impact the PCR17 value for remote attestation, but has no impact on passing the launch control policy. The user selects a *policy type* of ANY or LIST. Selecting LIST displays the LIST information. Since a policy may contain up to eight lists, the tool allows the user to create a list or select an existing list to view, modify, or delete. Selecting LIST also means that the tool will create a policy data structure, calculate the hash measurement of the policy data structure, and use that list hash when it generates the NV policy data. Currently, the only hash algorithm supported is SHA1. We can expect additional algorithm choices in the future to support countries like China that require a different hash algorithm.

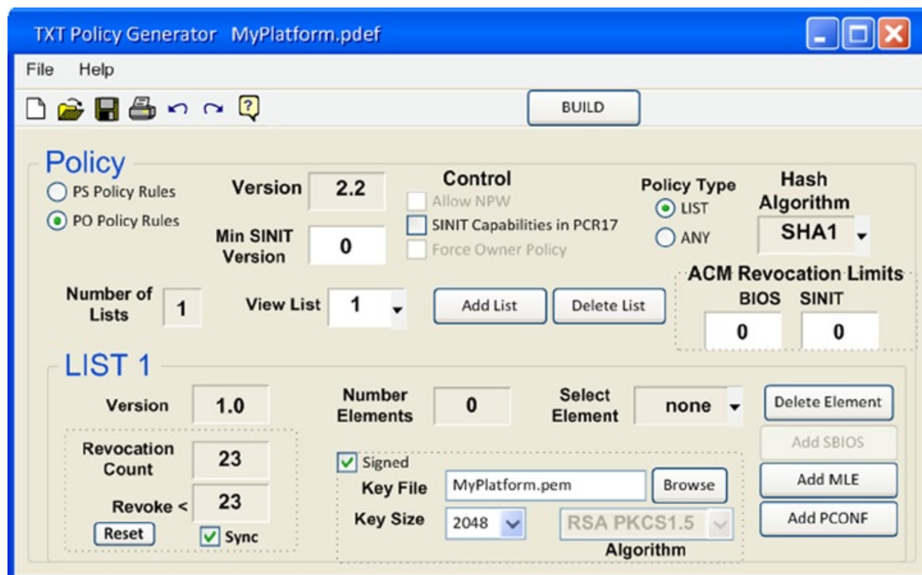

Figure 3-9. *Policy generator*

Each list may be signed or unsigned. Signed lists allow the manager to update the policy data structure without changing the NV policy data (because the hash measurement of a signed list only covers the public key used to verify that the list is authentic). This makes it easier to update policy by pushing new policy "down the wire" or use a file server for policy administration. Any update to an unsigned list requires the NV policy data to be updated with the new hash measurement of the policy data structure. And of course this requires a TPM NVRAM operation. In our opinion, signing a list is much simpler than updating TPM NVRAM—especially considering that you would need to perform the TPM update on every platform to which the policy change applies.

A signed list has a *revocation count* that is incremented each time the list is modified. The user may select *Sync* to synchronize the *Revoke* value with the *Revocation Count* value or manually set it. The *Revoke* value is part of the NV policy data and means that the corresponding list with a *Revocation* count less than the *Revoke* value will not be allowed. Since the *Revoke* value is held in the NV policy data, it is not applied until the manager updates the TPM NV policy data. Updating the *Revoke* value in the NV policy data typically is not necessary unless the manager is removing a PCONF or MLE measurement and wants to prevent that configuration or OS/VMM from performing a measured launch.

I should probably explain this better. Take for example that you have a signed list allowing OS versions 1.1, 1.2, and 2.1. Let's assume its *Revocation Count* is 2 and the *Revoke* value in the NV policy data is 0. You then update the policy data structure (PDS) with a new signed list that only allows versions 2.1 and 2.2 because 1.x has a security flaw. The revocation counter is automatically incremented to 3 in the list; however, the *Revoke* value in the NV policy data does not change. A potential attack would be for an attacker to replace the new *PDS* with a copy of the old PDS and then cause the platform to boot an older version of the OS so it can exploit the vulnerability. The way to prevent this is to update the *Revoke* value in the NV policy data. When you update the *Revoke* value in the NV policy data to be the same as the new list (3), then the SINIT LCP engine will reject the older signed list (because its *Revocation Count* is less than the allowed value). The *Revocation Count* is protected by the signature, so it cannot be maliciously altered. If you feel confident that the older versions don't pose a threat (either they don't have a vulnerability or there are other protections that prevent OS rollback), then there is no need to update the TPM NVRAM when generating a new signed list.

OK, so let's get back to discussing what is in a policy list. Each list may contain zero or one PCONF element and zero or one MLE element. Note that an SBIOS element is only valid in a platform supplier policy and is not used for the launch control policy.

Specifying Platform Configuration: The PCONF Element

The launch control policy allows the platform owner to specify what is considered an acceptable platform configuration. In the policy generator, the manager would select *Add PCONF* and the tool would display PCONF information, as illustrated in Figure 3-10.

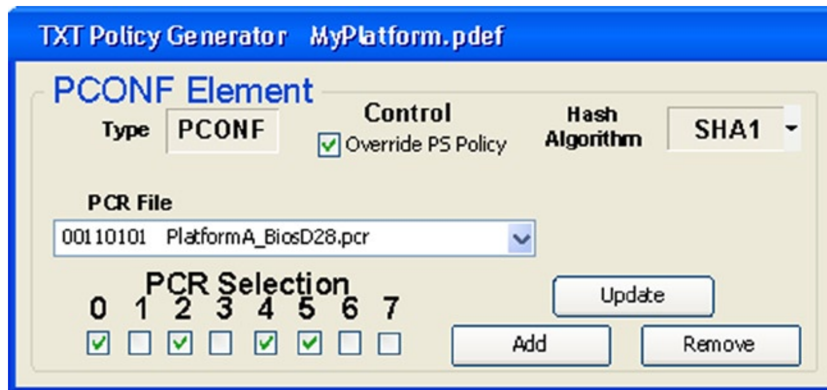

Figure 3-10. *PCONF policy generator*

Up to this point, we have revealed that a set of measurements has been made and stored in protected *Platform Configuration Registers* (PCRs). This starts with the "static root of trust" measurement, which is stored in PCR 0 and the platform firmware makes additional measurements into PCRs 0–7.

- PCR0 - CRTM, BIOS, and Host Platform Extensions
- PCR1 - Host Platform Configuration
- PCR2 - Option ROM Code
- PCR3 - Option ROM Configuration and Data
- PCR4 - IPL Code (usually the MBR)
- PCR5 - IPL Code Configuration and Data (for use by the IPL Code)
- PCR6 - State Transition and Wake Events
- PCR7 - Host Platform Manufacturer Control

These first eight PCRs can be thought of as measuring the platform configuration. However, you might not want to include all of these in the matrix that determines if the platform configuration is trusted. For example, if you were concerned only in assuring the BIOS has not been corrupted and that the Master Boot Record has not been altered, then you would only need to specify that PCR0 and PCR4 contain expected values.

Intel TXT allows the platform owner to specify a set of *PCRInfo structures*, where each PCRInfo structure describes an acceptable platform configuration and the content of a PCRInfo specifies which PCRs are to be considered (and the resulting hash of those PCRs). We refer to this set of PCRInfos as the platform configuration (PCONF) policy.

The policy generator allows the manager to add a new PCRInfo to the PCONF policy or select an existing one to delete or modify. To be able to create a PCRInfo, the tool needs the PCR values. The PCR Dump tool performs this task and is run on the platform that is being added to the policy. The PCR Dump tool captures all of the PCR measurements so that the manager can specify any combination of PCRs. The manager simply selects the PCR dump

file and specifies which PCRs. In Figure 3-10, we see that the user selected PCRs 0, 2, 4, and 5 from a PCR dump file named PlatformA_BiosD28.pcr. When the user builds the policy, the generator will create a PCRInfo by generating a hash of the PCR0, 2, 4, and 5 values from the specified PCR file. The user specifies as many PCRInfos as desired, each specifying a PCR file name and set of PCRs. However, what is placed in the policy data structure is only the list of PCRInfo (that is, the PCR selections and respective composite hashes.

At the time of the measured launch, the policy engine evaluates each PCRInfo in the PCONF policy until it finds a match. It evaluates a PCRInfo by reading the current values of the specified PCRs, creating a composite hash from them, and comparing that result to the composite hash specified in the PCRInfo. A match means that the specified PCRs contain the exact same measurement values that were used in calculating the PCRInfo. Thus, the platform has a known acceptable configuration.

If there is no PCONF policy element (this is not the same as no PCRInfos) then the PCONF policy equates TRUE. If one of the PCRInfos evaluates TRUE, then the PCONF policy evaluates TRUE. Otherwise (no PCRInfos match), the PCONF policy evaluates FALSE, which means the platform configuration is not in policy, which results in a *TXT reset*.

One last point, there may be multiple lists in the policy data structure and each list may contain a PCONF element. For the PCONF policy to pass, it only requires a match in any of the elements.

Specifying Trusted Operating Systems: The MLE Element

The launch control policy allows the platform owner to specify which system software (operating systems) are allowed to perform a measured launch and also prevent an allowed OS/VMM from performing a measured launch if its trusted code has been altered. Remember that the code that is measured and executed first after the SENTER is referred to as the *measured launch environment* (MLE) measurement.

There is an expectation that the TBOOT MLE code will first enable and enforce all of the protections mechanisms, after which it will then load, measure, and authenticate additional modules (such as kernel code and drivers). An OS can use the fact that MLE code passing LCP means that it has not been altered, and thus data objects within that code can be trusted. An example of this is when the OSV/VMV signs the kernel and other modules and includes the public signing key(s) in the MLE code. The MLE code uses the keys to verify that the kernel and other modules are authentic, and thus will not execute those modules if they have been modified or signed by the wrong key. Note that the public keys can be trusted if, and only if, the MLE code has not been modified (that is, has passed LCP).

Therefore, *whether the operating system will be trusted is determined by the measurement of its MLE code*. Each list in the policy data structure may contain an MLE element, which contains a list of MLE measurements that meet the datacenter's policy.

To include an MLE element, in the policy generator tool, select *Add MLE* and the tool will display MLE information, as illustrated in Figure 3-11. Next select the hash files to include. Hash files are provided by the OSV/VMV. The policy generator extracts the MLE hash from the file such that the policy data structure only contains the hash values (not file names). The MLE policy also allows you to specify the minimum SINIT version that can be used to launch those OS/VMMs specified in that list.

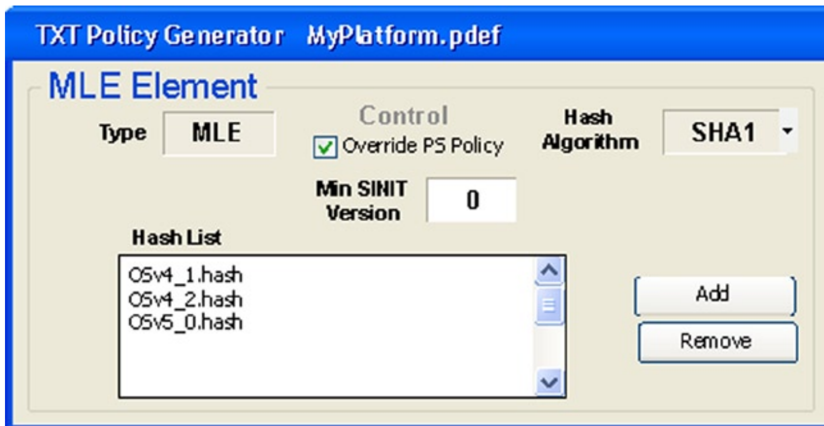

Figure 3-11. *MLE policy generator*

At the time of the measured launch, the policy engine evaluates the MLE policy, comparing that actual MLE measurement to those in the MLE elements until it finds a match. If there is no MLE policy (no MLE element in the policy data structure), then the MLE policy equates to TRUE. If the MLE measurement matches one of the measurements in the MLE Element, then the MLE policy evaluates to TRUE. Otherwise (no match), the MLE policy evaluates to FALSE, which results in a TXT reset.

Specifying Trusted ACMs

Regardless of how much an ACM is tested, there is still a possibility of security vulnerabilities. Even more important, ACMs can be updated with enhanced security checking and other features. As with any software, ACMs mature over time, and thus the launch control policy provides for the platform owner to specify the minimum version of the SINIT ACM that will be allowed to perform a measured launch. This is done in order to prevent malicious software from replacing the ACM with an older ACM to take advantage of a known flaw or shortcoming. It should be noted that Intel has its own means to revoke both BIOS ACMs and SINIT ACMs if the need arises, and that means does not involve the LCP (it uses NVRAM controlled by the ACMs to store the revocation values). Thus, revoking ACMs because of security flaws are handled regardless of the LCP.

There are actually two ways to set the minimum SINIT version. Both the MLE element in the policy data structure and the NV policy data contain a "minimum SINIT version," and whichever is larger determines the minimum value for the platform. The value in the NV policy data allows the platform owner to establish the minimum for all policies. The value in the MLE element applies to the measurements listed in that element, and thus the platform owner can specify different minimums for different OS/VMMs. Although I am not sure how practical this is, the capability is there. The real value is that using the min SINIT in a signed list avoids having to update the TPM NVRAM.

Specifying a Policy of "ANY"

The platform owner can specify a launch control policy of ANY. This means that the policy does not provide PCONF or MLE measurements, and thus any OS/VMM on any platform configuration is permitted to perform a measured launch. This is an acceptable policy, especially when there is remote attestation. However, remote attestation applies its policy after the measured launch.

Since there are no PCONF and MLE elements, there is no need for the policy data structure. Thus, the NV policy data contains all of the information needed for the ACM to evaluate the launch control policy.

Revoking Platform Default Policy

The platform supplier also provides an NV policy data and (optionally) a policy data structure that affect the measured launch. Unlike the platform owner policy (PO policy), the platform supplier policy (PS policy—sometimes referred to as *platform default* or *PD policy*) may contain an SBIOS element that is used by a different policy engine to protect against reset attacks. The owner policy has no impact with respect to the SBIOS policy. Typically, the platform supplier policy is set to ANY (that is, does not contain PCONF or MLE elements) unless there is one of the following:

- *A fallback BIOS*. The platform supplier has the capability to switch back to the original BIOS image (in case the current BIOS/VMM becomes corrupted). Thus the PS policy contains an SBIOS element that specifies the measurement of the original BIOS startup code.
- *A signed BIOS policy.* The platform supplier has the capability to gracefully switch to another BIOS image (in case of BIOS update or falling back to a previous version). The PS policy contains an SBIOS element specifying measurements of authorized BIOS startup code, signed by the vendor, so it can be updated as part of the BIOS update.
- *A preinstalled OS/VMM:* The platform ships with an OS/VMM installed. The supplier may optionally provide PCONF and MLE elements.

The presence of the SBIOS element does not influence the launch control policy for the measured launch. If the platform does not come with a preinstalled OS/VMM, then typically there are no PCONF or MLE elements in the PS policy, and as far as the launch control policy engine is concerned, the PS policy is the same as ANY. Even when the platform does come with a preinstalled OS/VMM, the platform supplier is not required to include PCONF and MLE elements. But let's consider what happens when the PS policy does contain either PCONF or MLE elements.

The PCONF element and MLE element in the PO policy each have a flag that can be set to override the corresponding PS policy element. If the override flag is not set, then that policy evaluates to TRUE if either the PO policy element or the PS policy element evaluates to TRUE. When the override flag is set, the policy engine does not evaluate the corresponding element in the PS policy. Table 3-1 shows how the SINT ACM interprets the policies.

Table 3-1. *Policy Significance*

PO Policy	PS Policy	Result
None	ANY	Any PCONF & any MLE
	Only PCONF	PS.PCONF & any MLE
	PCONF&MLE	PS.PCONF & PS.MLE
	Only MLE	Any PCONF & PS.MLE
ANY	<don't care>	Any PCONF & any MLE
Only PCONF	ANY	PO.PCONF & any MLE
	Only PCONF	(PO.PCONF+PS.PCONF) & any MLE
	PCONF&MLE	
	Only MLE	PO.PCONF & any MLE
PCONF+MLE	ANY	PO.PCONF & PO.MLE
	Only PCONF	(PO.PCONF+PS.PCONF) & PO.MLE
	PCONF&MLE	(PO.PCONF+PS.PCONF) & (PO.MLE+PS.MLE)
	Only MLE	PO.PCONF & (PO.MLE+PS.MLE)

(*continued*)

Table 3-1. (*continued*)

PO Policy	PS Policy	Result
Only MLE	ANY	Any PCONF & PO.MLE
	Only PCONF	
	PCONF&MLE	Any PCONF & (PO.MLE+PS.MLE)
	Only MLE	
PCONF (override)	<don’t care>	PO.PCONF & any MLE
PCONF (override) & MLE	ANY	PO.PCONF & PO.MLE
	Only PCONF	
	PCONF&MLE	PO.PCONF & (PO.MLE+PS.MLE)
	Only MLE	
PCONF & MLE (override)	ANY	PO.PCONF & PO.MLE
	Only PCONF	(PO.PCONF+PS.PCONF) & PO.MLE
	PCONF&MLE	
	Only MLE	PO.PCONF & PO.MLE
PCONF (override) & MLE (override)	<don’t care>	PO.PCONF & PO.MLE
MLE (override)	<don’t care>	Any PCONF & PO.MLE

■ **Note** A “+” means if either equates true, “&” means both must equate true, “none” means either the policy does not exist or the policy is not ANY and there are neither PCONF nor MLE elements.

Why Is PO Policy Important?

Most likely when you install a new platform, measured launch works without having to create a platform owner policy. So let’s look at some reasons why it is beneficial for you to do so.

■ **Note** Before publishing this book, we asked a few leading datacenter managers to review its content. According to feedback from a senior datacenter technologist, this entire section deserves to be highlighted. So we thought we would convey his emphasis to you. Also, we would like to thank those who did review, and thus helped make this a better book.

Prevent Interference by the Platform Supplier Policy

The platform vendor providing a preinstalled OS/VMM means well by providing a PS policy with the OS/VMM measurement. This was done as an aid in providing the initial policy with the expectation that the datacenter will supplement it with the PO policy.

As you can see from Table 3-1 and Figure 3-8, if there is no PO policy and there is a PS policy for a preinstalled OS/VMM, then the PS policy can prevent a measured launch after the datacenter either updates the BIOS or updates the OS/VMM. The solution to this problem is to create a platform owner policy, even if that policy is ANY.

Establishing Trusted Pools

The owner policy is the place where the datacenter asserts its policy on what is considered "trusted." The fact that a platform has performed a secure launch attests to the platform complying with prescribed procedures for protecting the OS/VMM and its data. For instance, when an OS boots, all of the processors are started in *real* mode, and the OS has to transition them to *virtual* mode and then to *protected* mode. During this time, there are vulnerabilities. To negate those vulnerabilities, the measured launch places all the processors in a special sleep state and wakes them up in protected mode—but only after the initiating processor has set up all of the protections.

Thus performing the measured launch is sufficient qualification for a server platform to be considered part of a trusted pool of servers. However, the datacenter might want to impose additional requirements or otherwise qualify what is considered "trusted." That can be enforced via the platform owner policy, as illustrated in Figure 3-12.

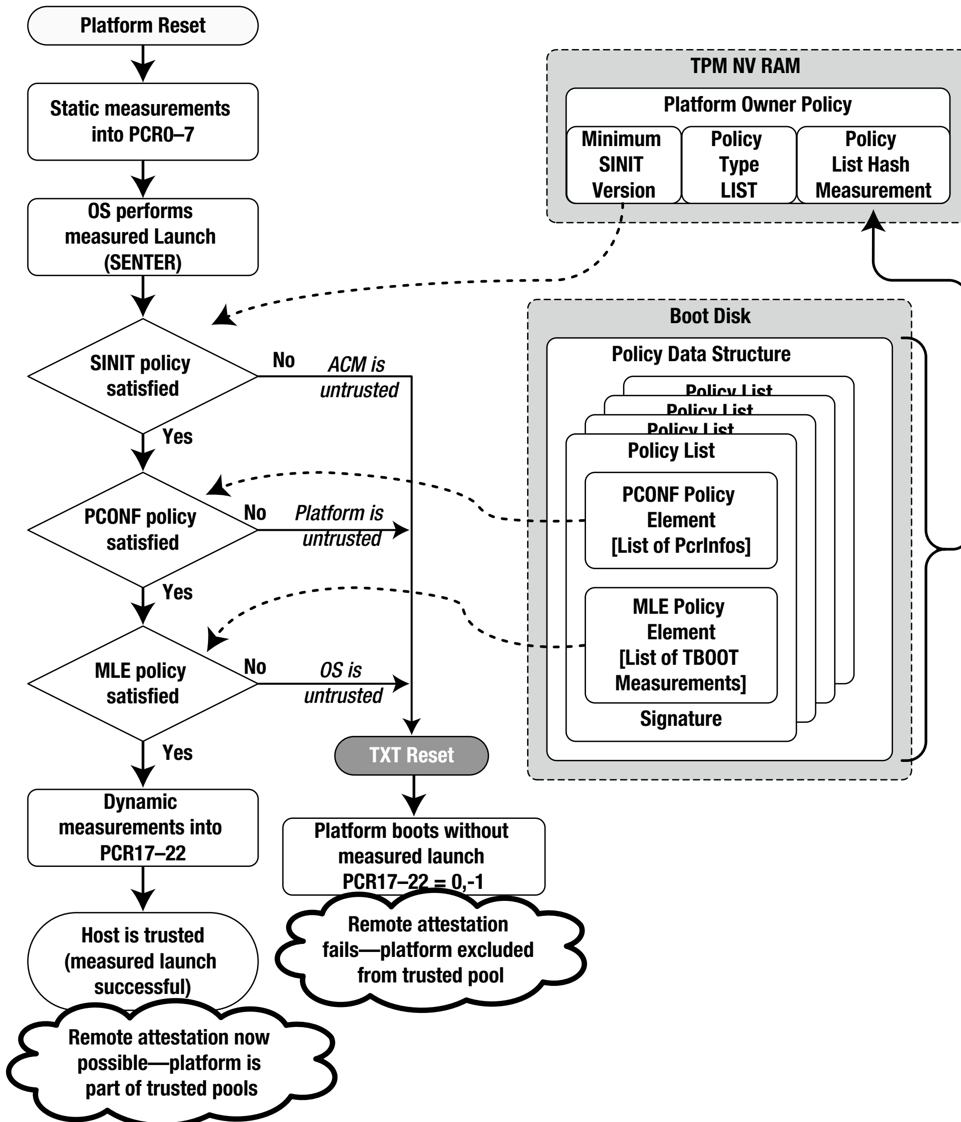

Figure 3-12. *Launch control policy flow*

Let’s take a look at an example. Referring to the MLE policy in Figure 3-13, assume an older version of the OS (version 4.x in our example) has a known vulnerability, and even though that version is capable of performing a measured launch, you want to exclude it from the pool of trusted servers.

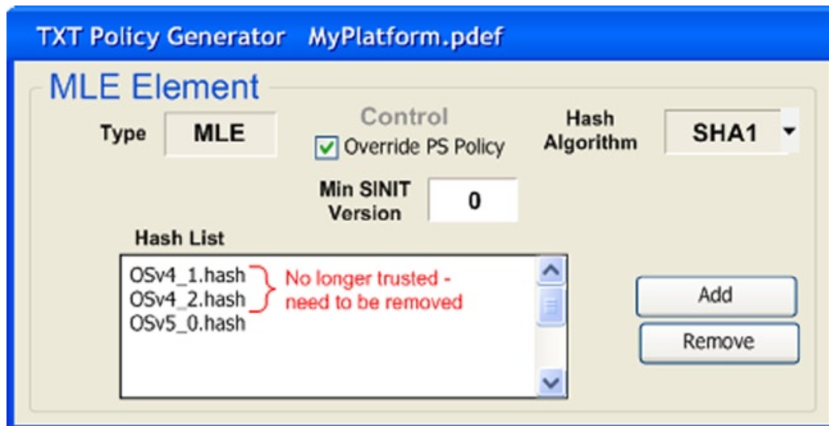

Figure 3-13. *MLE policy generator*

The solution is to create a policy that includes the measurements of only the trusted versions. So we simply remove the measurements for version 4.1 and 4.2. Any platforms with the older OS/VMM versions are now prevented from performing a measured launch, and thus prevented from joining the pool of trusted servers (at least until their OS is updated)—this is because attestation will fail since PCR17 and 18 will contain all zeros.

As you update the OS/VMM on the untrusted servers, those servers will now pass the measured launch control policy, and thus once again join the pool of trusted servers. Once all servers have been updated, you might be tempted to change the policy back to ANY, but that would not protect against an attack where malicious software was able to roll back the OS/VMM version to take advantage of its vulnerability.

Reduce the Need for Remote Attestation

To determine if a platform has passed the launch control policy, all that needs to be done is to verify that PCR18 contains a nonzero value. For those who trust the datacenter to establish a launch control policy that meets their needs, this should be sufficient. Otherwise, remote attestation will need to compare PCR values with lists of known good values.

At this point, we should discuss the difference between private clouds and public clouds. One of the biggest drivers for remote attestation is the ability to satisfy the service client about the integrity of the platform that is hosting the client's applications. This is a significant factor for public clouds (where the service clients are not part of the organization providing the computing services).

But what about private clouds (where the service provider and service clients are part of the same organization)? In this case, the datacenter would be trusted with maintaining the integrity of the services without the need for remote attestation by a third party. Thus the establishment and use of trusted pools might very well be sufficient for the service clients, and therefore the satisfaction of the platform owner policy provides the level of attestation required by the clients.

One could also extend this argument to include a public cloud provider providing services for clients within its own organization. I can imagine that the billing department for `Amazon.com` would trust its own organization to provide secure services, while the billing department for an external client would want an audit trail confirmed by a third party via remote attestation. There would be no need to burden a client within the service provider's organization with overhead for remote attestation. On the other hand, if remote attestation is there, why not make use of it?

Reset Attack Protection

One of the benefits of Intel TXT is protection against reset attacks. A reset attack is where an attacker causes a platform reset before the OS/VMM can do an orderly shutdown. Thus there can be secrets such as encryption keys, passwords, and personal information left unprotected in memory.

To negate reset attacks, Intel TXT maintains a *Secrets* flag, which the OS/VMM sets after it does a measured launch, but before placing any secrets in memory. The OS/VMM clears the *Secrets* flag when it does a graceful shutdown (after removing all secrets and sensitive information from memory). If the platform resets before the *Secrets* flag is cleared, the memory must be scrubbed. For client platforms, the memory architecture is simple, so the ACM performs the memory scrubbing and does not enable memory controllers until after it has scrubbed the memory. Since the memory architecture for servers is more complex, the ACM has to depend on the BIOS to scrub the memory. When the *Secrets* flag is set when the platform initializes, the memory controllers are disabled until the BIOS ACM validates that the BIOS is trusted to scrub memory. This is where the SBIOS policy engine comes into play.

There are two types of SBIOS policy. The platform vendor may choose either autopromotion (the most common at this time) or signed BIOS policy (which is receiving a lot of interest from manufacturers):

- *Autopromotion.* Each time the platform resets, the BIOS ACM measures the BIOS startup (SBIOS) code and places that measurement in a TPM NVRAM location to remember it for the next boot. If the platform performs a measured launch, it means that the PCONF policy was satisfied, and thus the BIOS was trusted. If the platform resets with secrets in memory (that is, the *Secrets* flag set), then the SBIOS policy engine measures the BIOS startup code and makes sure it had not been altered. It does this by comparing its measurement against the saved measurement.
- *Signed BIOS policy.* Instead of using the remembered SBIOS measurement, when the platform resets with the *Secrets* bit set, the SBIOS policy engine compares the measurement of the BIOS startup code to a list of measurements signed by the platform vendor. Because the list is signed, each BIOS update can provide a new signed list of SBIOS measurements for the current code.

Signed BIOS policy is independent of launch control policy. That is, for the BIOS to be trusted, its measurement must match the known good value provided by the factory. On the other hand, autopromotion depends on the LCP to determine if the BIOS is trusted.

Therefore, after a reset attack, the malicious software finds nothing in memory. And an attempt to alter the BIOS to bypass the scrubbing will result in no memory. There is a weakness with autopromotion that can be mitigated with the PO policy. If LCP allows any platform configuration, then any BIOS corruption that occurs before the last platform reset would not be detected. This can easily be avoided by setting a PO PCONF policy that uses the PCR0 measurement. Now, PCR0 must contain a known good value and because the PCR0 value includes the SBIOS measurement, any corruption to the BIOS prevents the measured launch.

Considerations

There are a number of issues to consider when establishing the launch control policy.

- The launch control policy consists of setting MLE policy, PCONF policy, and SINIT policy.
- Changing SINIT policy can be done by changing the TPM NV policy data or changing the MLE element in the policy data structure. Changing the NV policy data requires performing a privileged TPM operation on each platform.
- Changing MLE policy or PCONF policy can be done without changing the NV policy data if the platform owner uses signed policies. Signed policies allow the datacenter to push policy updates “down the wire,” making it easier to administer launch control policies. See “Policy Management” in the next chapter.

- MLE policy uses the measurement of the TBOOT MLE code to verify that the code has not been altered by comparing the measurement to a list of known good values. Each OS/VMM update potentially adds another known good value. An OS/VMM version/revision number should be part of the TBOOT MLE code to force a new measurement for each major update. Without this, the LCP will not be able to distinguish between different versions of the software (if there are no updates to the TBOOT MLE code). The platform owner needs to know if the TBOOT MLE measurement implicitly changes with each update or just major updates.
- The downside of including a revision value in the TBOOT MLE measurement is that every revision requires updating the PO policy. See “MLE Updates” under “Policy Management” in the next chapter.
- PCONF is the set of measurements that defines the platform configuration. PCR0 contains the static root of trust for the other static PCRs (PCR 1–7). PCR0 should always be included because PCR0 attests to the integrity of the other PCR measurements.
- PCR0 contains multiple measurements, but, in essence, it represents the BIOS code. Including PCR0 in the PCONF policy verifies that the code was provided by the factory (at the time the platform left the factory or the BIOS update as sent from the factory). Any change to any part of the BIOS trusted code results in a change to the PCR0 measurement. Thus, including PCR0 detects any unauthorized BIOS changes.
- The downside of including PCR0 in the PCONF policy is that every BIOS update requires updating the PO policy. See “BIOS Updates” under “Policy Management” in the next chapter.
- There is an inherent complexity in using multiple PCRs in the PCONF policy. That is, the combinations of PCRInfos that need to be specified multiply with the number of variations. For example, let’s say the PCONF policy uses PCR0, PCR1, and PCR4, and there are three acceptable values for PCR0, four for PCR1, and two for PCR4. This would require $3 \times 4 \times 2 = 24$ PCRInfos to allow all combinations. A BIOS update that adds another PCR0 value changes that to $4 \times 4 \times 2 = 32$—meaning eight more PCRInfos must be added to the policy. On the other hand, if the new PCR0 value replaces one of the others, then eight of the original PCRInfos would have to be removed. This requires sophisticated policy management tools to simplify policy tracking.

Summary

At this point, you should have a good understanding of what needs to be done to enable Intel TXT from BIOS setup, establishing TPM ownership, and installing system software. These are very objective steps, and although they do vary from vendor to vendor, the learning curve is simple and you should be able to get specific information directly from the platform manufacturer as well as the OS/VMM vendor.

On the other hand, launch control policy is very subjective and depends more on the datacenter and the availability of tools for creating, setting, and maintaining policies. The next chapter will walk you through the process for selecting a launch control policy that meets your needs.