CHAPTER 4

Foundation for Control: Establishing Launch Control Policy

Now it is time to discuss which launch control policies are right for your datacenter. That decision rests on many factors and the purpose of this chapter is to help you understand those tradeoffs. We will also discuss how to manage policies, because setting the policy for one machine is not hard, but managing a datacenter with hundreds of servers can be challenging. Managing tens of thousands of servers requires quite a bit of discipline.

Quick Review of Launch Control Policy

In the previous chapter, we established that the launch control policy consisted of three independent policies, as illustrated in Figure 4-1.

- *SINIT policy* prevents a measured launch using an older SINIT Authenticated Code Module (ACM). This is done by specifying the minimum ACM version that is allowed to perform the measured launch. The minimum SINIT version is specified in the NV policy data stored in the TPM NVRAM and may also be specified in an MLE policy element (in the policy data structure).
- *Platform Configuration (PCONF) policy* prevents a measured launch unless the platform configuration matches a known good configuration, as measured by the values in a set of PCRs selected by you. These known good configurations are listed in a *PCONF policy element* that is part of the policy data structure stored in the boot directory.
- *MLE policy* prevents a measured launch unless the OS/VMM measurement matches a known good value. These known good measurements are listed in an MLE policy element that is part of the policy data structure.

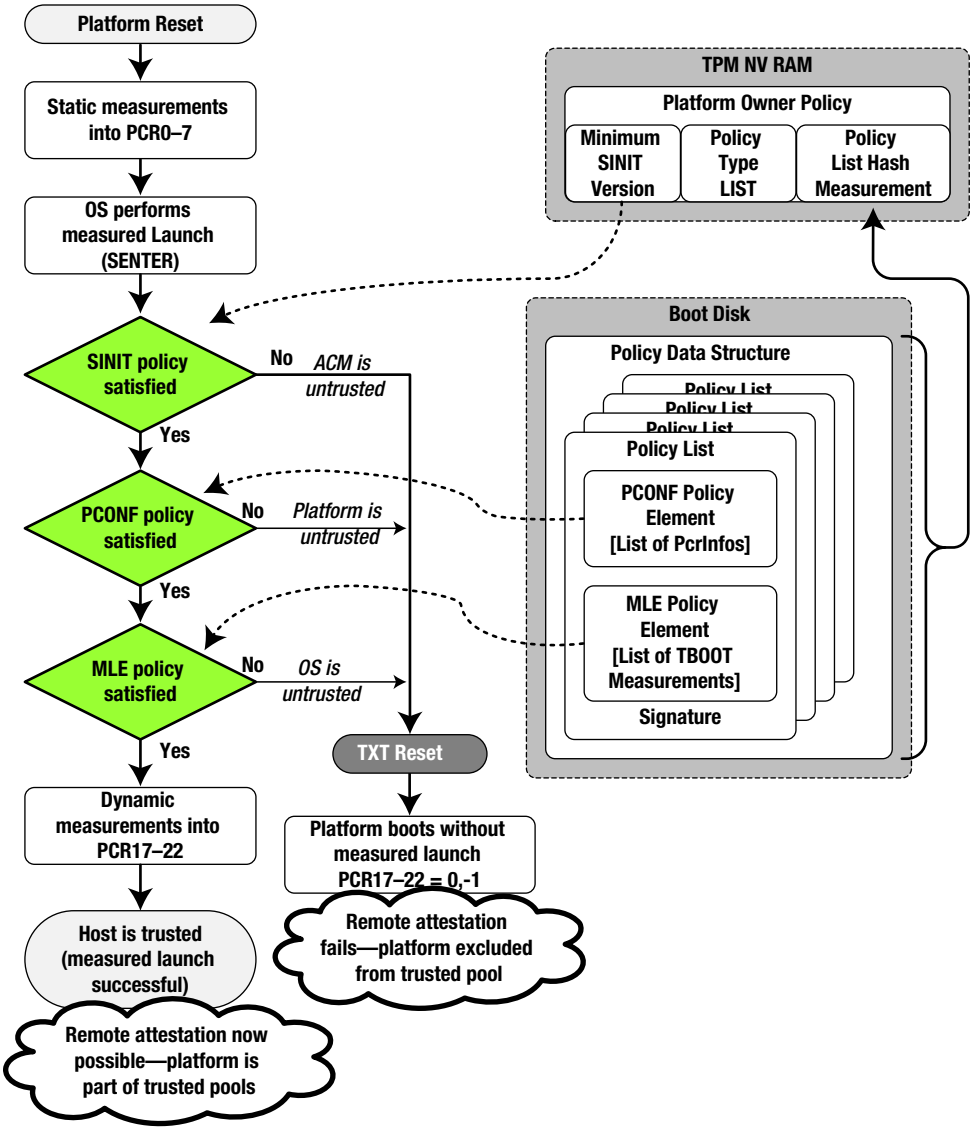

Figure 4-1. *Overview of launch control policy*

You may select a policy type of ANY or LIST. With a policy type of ANY, you can still enforce SINIT policy while allowing any OS/VMM to perform a measured launch using any platform configuration.

With a policy type of LIST, you can specifically establish a platform configuration policy, MLE policy, or both by specifying lists of known good values.

The lists of known good values reside on disk in a policy data file, and to prevent tampering, the policy type and the measurement of the policy file are stored in (and protected by) the TPM's NVRAM.

When Is Launch Control Policy Needed?

In the previous chapter, we raised the following questions:

- Is remote attestation needed if there is a strong launch control policy?
- Is a launch control policy needed if there is remote attestation?

These answers are very subjective, will vary from datacenter to datacenter, and most likely will vary over time. To discuss those questions, we need a better understanding of remote attestation.

Remote Attestation

In a nutshell, *remote attestation* makes policy decisions based on the server's integrity as measured by its PCR values (illustrated in Figure 4-2). But wait, isn't this what the launch control policy does?

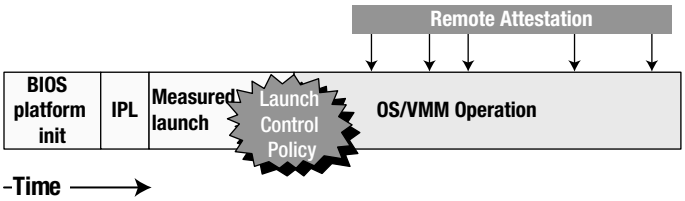

Figure 4-2. *Policy decisions*

The answer is yes and no. Yes, the LCP makes a policy decision based on the PCR measurements. But LCP controls the ability of the platform to do a measured launch and prevents untrusted platforms from performing the measured launch. Remote attestation cannot control the measured launch because it uses the PCR values derived from the measured launch. On the other hand, remote attestation can impact many other decision processes and can govern the operation of the platform. For example, it could do the following:

- Determine if a platform is trusted and cause an untrusted platform to be placed into quarantine, isolating it from the hive, and denying it access to storage and the production network.
- Establish various levels of trust based on both the platform and the OS/VMM version.
- Determine which applications (if any) are allowed to run on which servers. In later chapters, we will look at how trust-based migration policies control whether an application is allowed to migrate to another platform that has a different trust level.
- Assure compliance with government and corporate regulations.
- Provide an audit trail that can prove compliance to both customer and government requirements.

The implementation and usage of remote attestation is very flexible. Thus, there can be very few or many agents involved in remote attestation, such as agents that pull measurements from the servers, an agent that evaluates those measurements against known good values (white lists) to establish a trust level, and various management agents that each enforce policies based on that determination. Essentially, there are very few limits on what can be done with remote attestation.

What Does Launch Control Policy Deliver?

It's useful to think of launch control policy as the precursor to remote attestation. That is, LCP is the first level of defense that occurs at boot-time, and remote attestation is the second level that occurs as often as needed. Here are some capabilities of launch control policy that remote attestation does not provide:

- *Enhanced protection against reset attacks*. When the BIOS uses autopromotion, the defense relies on the LCP to determine if the BIOS is trusted in the first place. With a policy of ANY, the platform is protected against alteration/corruption of BIOS after the last boot (and before the reset), but cannot protect against long-term corruption (that is, before the last boot). Using a PCONF policy detects any BIOS corruption. If the BIOS uses a signed SBIOS policy, then this is not an issue because the OEM provides the list of known good SBIOS measurements. Remember that SBIOS is the trusted portion of BIOS responsible for mitigating the reset attack. Thus, at the time of a reset attack, the platform detects a corrupted SBIOS regardless of when the corruption occurred. Although signed BIOS policy is ideal for protection against reset attacks, it does not replace the PCONF policy in the LCP. That is because the signed BIOS policy only comes into play after the reset attack and the PCONF policy is evaluated at every launch (typically as part of loading the OS/VMM).

- *Restrict TPM access*. Any OS/VMM has access to TPM locality 0, however, only an OS/VMM that passes the launch control policy has access to TPM locality 2. Locality 2 is what gives the OS/VMM the ability to measure its components into the dynamic PCRs (19–22) and thus seal/unseal data to those values. Likewise, keys, TPM NVRAM, and other TPM resources can be tied to locality 2, preventing anyone except the trusted OS/VMM from using them. By using a stronger launch control policy (specifically an MLE policy), you further prevent rogue software from impersonating a trusted OS/VMM, and thus further restrict what software can even attempt to use those protected resources.

- *Self-contained*. Although remote attestation is more flexible, it does bring additional complexity that is not always needed. For example, consider a private cloud scenario where the service clients implicitly trust their IT department and have no need for third-party, trust-based compliance and auditing. For this case, the local attestation provided by the launch control policy is all that is needed.

The bottom line is that launch control policy is applied before the OS/VMM is permitted to do a measured launch and remote attestation comes sometime later.

It is possible that over time, remote attestation capability will grow to the point where the value of launch control policy diminishes. For example, consider a device that evaluates the platform's measurements immediately after the launch and either quarantines the platform or powers it down. This could be just as effective as failing the launch control policy, especially if this is done before any applications are allowed to execute. For this case, the autopromotion requirement would be satisfied because the protection from the early remote attestation would be as effective as the TXT reset caused by LCP.

Let's review one more thought before we get down to the details of selecting the policy components. That is, full-blown remote attestation takes the platform's PCR measurements and compares them to known good values, and then uses that result to determine a trust level. The final role is for a management application to make a policy-based decision using that trust level. So, if the platforms provide an adequate launch control policy, it might be sufficient for the management application just to detect that the platform was able to perform a measured launch. Now much of the complexity of remote attestation goes away, because management applications only need to detect whether PCR18 contains the default value—any other value indicates that the platform passed the launch control policy.

Platform Configuration (PCONF) Policy

Before you can decide if you want to specify a PCONF policy as part of your launch control policy, you need a better understanding of what it means to specify a PCONF policy and how you can customize the PCONF policy.

The biggest problem with having a PCONF policy is that there is no single correct policy, and most likely the best-suited policy for you this year will not be the best in the years to come.

One other thought to keep in mind is that, as mentioned in Chapter 1, Intel® Trusted Execution Technology is part of a defense-in-depth strategy. BIOS developers have incorporated many security concepts (such as signed BIOS updates and password-protected configuration) that protect against malicious BIOS corruption and unauthorized platform configuration change. PCONF policy is yet another layer of protection. The other side of the coin is that there are additional advantages that can be derived from a PCONF policy. The point is that as you evaluate what PCONF policy to use, also consider protections offered by other technologies.

Specifying Trusted Platform Configurations

The first step in creating the PCONF portion of the launch control policy is to determine the matrix of PCRs (that is, which PCRs need to be examined). In the previous chapter, we identified that PCR0–PCR7 were candidates and that part of the PCONF policy is selecting which PCRs to include, as illustrated in Figure 4-3.

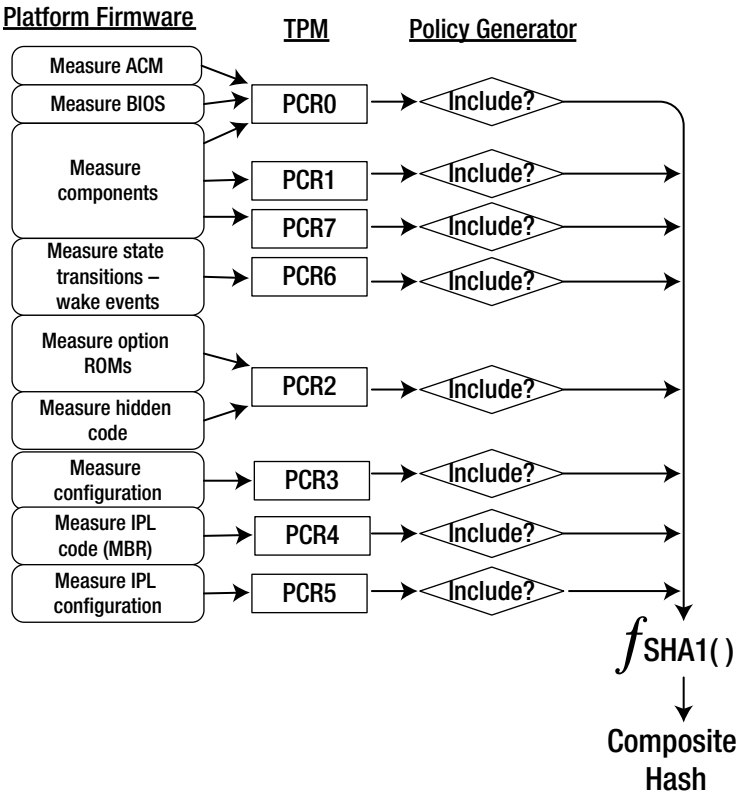

Figure 4-3. *PCONF PCR selection*

So let's take a closer look.

PCR0: CRTM, BIOS, and Host Platform Extensions

You will always want to include PCR0 because it contains the core-root-of-trust (CRTM), and thus it attests to the validity of the other PCRs. In addition, this PCR value represents the BIOS integrity. If the BIOS has been corrupted or maliciously altered, it will show up in this measurement. Here is what the TCG requires[1] being measured into PCR0:

- The CRTM's version identifier.
- All firmware physically bound to the motherboard.
- Manufacturer-controlled embedded option ROMs. These are embedded option ROMs whose release and updates are controlled by the platform manufacturer.
- Embedded System Management Module (SMM) code and the code that sets it up.
- ACPI flash data prior to any modifications.
- Boot Integrity Service (BIS) code (excluding the BIS certificate).

The TCG specification also requires that "If the measurement of the CRTM, POST BIOS and Embedded Option ROMs cannot be made, the CRTM MUST extend the value 01h to each PCR in the range 0–7." This prevents rogue code from extending those PCRs with what would appear to be known good values.

The TCG allows the BIOS to measure "any other code or information that is relevant to the CRTM, POST BIOS or Platform Extensions" to PCR0. Typically, all BIOS code gets measured to PCR0 because OEMs want the maximum coverage and don't want to risk their reputation by excluding a piece of code that could be the target of an attack.

> *Unauthorized modification of BIOS firmware by malicious software constitutes a significant threat because of the BIOS's unique and privileged position within the PC architecture. A malicious BIOS modification could be part of a sophisticated, targeted attack on an organization—either a permanent denial of service (if the BIOS is corrupted) or a persistent malware presence (if the BIOS is implanted with malware).*
>
> —NIST: National Institute of Standards and Technology, US Department of Commerce, Special Publication 800-147 *BIOS Protection Guidelines*

It is easy to see why including PCR0 should be fundamental. If the BIOS code is altered, either accidentally or maliciously, the platform could experience loss of confidentiality, integrity, and availability—resulting in system instability, system failure, and/or information leakage.

PCR1: Host Platform Configuration

This is the second most important PCR because it can be used to detect unauthorized changes to BIOS configuration. Unauthorized changes to platform configuration can potentially be just as disruptive and threatening as changes to BIOS code because it also could cause system instability, system failure, and/or information leakage.

The TCG requires the following components be measured into PCR1:

- CPU microcode update
- Platform configuration, including the state of any *disable flags* affecting the measurement of entities into this PCR

[1]From the *TCG PC Specific Implementation Specification*, Version 1.1, August 18, 2003, Copyright© 2003 Trusted Computing Group, Incorporated.

- BIS certificate (measurement can be disabled)
- POST BIOS-based ROM strings (measurement can be disabled)

The TCG allows measurement of the following (excluding privacy information):

- ESCD, CMOS, and other NVRAM data
- SMBIOS structures
- Passwords

BIOS does not measure values that are dynamic, such as counters, system time/date, or anything else that automatically changes. This means that the PCR1 measurement should be the same from boot to boot if there is no configuration change. It also excludes system-unique information, such as asset numbers and serial numbers. Thus, identical platforms configured the same should produce the same PCR0 and PCR1 values.

It was discovered that not all manufactures initially realized this requirement, so some platforms out there will not have the same PCR0 and PCR1 values, even when they are configured identically. If you have any of these platforms, it is not a security concern, but it will require additional effort to include them in a global PCONF policy. This has been corrected, and as of 2012, all manufactures were conforming to this requirement.

PCR2, 3: Option ROM Code and Configuration Data

Since option ROMs provide the means for add-in cards to provide firmware extensions (especially for adding boot capabilities), this PCR provides additional protection for platforms that contain plug-in devices that impact the boot flow. This can include option ROMs on the main board that are not controlled by the BIOS update.

It's prudent to be a little skeptical about PCR3. We know we can trust the BIOS to measure the visible portion of the option ROM code into PCR2; however, the visible option ROM code is responsible for measuring any hidden portions of the code (and extending into PCR2), as well as measuring device configuration information and extending that measurement into PCR3. The authors are not aware of any program that actually confirms that I/O device vendors adhere to those requirements. Therefore, it would not be surprising if PCR3 never gets extended by the I/O device. Check with the device manufacturer to see whether the device complies with TCG requirements.

Even if this is the case, it doesn't mean you should not specify PCR3; rather it means that the effectiveness of specifying PCR3 may be reduced, so you will only want to purchase add-on devices that comply with TCG requirements.

The importance of including PCR2 (and PCR3) in the PCONF policy depends on a number of factors:

- If there are never any add-in cards, then this becomes a "don't care." Adding PCR2 to PCONF policy would be a way to detect if an I/O card is added (or removed). On the other hand, do you want such an event to prevent measured launch (until a card is removed/replaced or PCONF policy is updated)? Note that add-in devices that don't have option ROMs have no impact and are not a concern because they don't impact the boot flow and they don't show up in the PCR2 and PCR3 measurements.
- If any of the add-in cards can impact the boot choice, and the MLE policy is ANY (meaning no MLE element), then including these PCRs can provide additional protection against attacks seeking to load a different OS/VMM.
- Once the OS boots, it typically installs its own (signed) I/O drivers, and thus does not depend on option ROM code or configuration. If this is the case, and the MLE policy only allows authenticated OS/VMM, then the value of detecting option ROM modifications is reduced.

- Many option ROMs are implemented using flash memory so that they can be updated. Option ROM updates don't get the same level of scrutiny as BIOS updates. Including PCR2 or PCR3 in PCONF policy means that anytime there is an option ROM update, the PCONF policy must be updated. Since the OS handles driver updates, you might not be aware that the update takes place until the measured launch fails.
- Don't forget that any option ROM is a potential attack point. That is, if malicious software can control the flash image, it can configure the option ROM to insert itself into the boot process—regardless of the functionality of the original code.

PCR4, 5: IPL Code and Configuration Data

These two PCRs can be used to assure that you boot from the same source. PCR4 contains the measurement of the initial program loader (IPL) code. Typically, this comes from the master boot record (MBR) when booting via a storage device or the bootstrap code from the network. By definition, it is the code that performs the handoff from the BIOS to the OS. What makes PCR4 potentially difficult to use is that each time the BIOS selects a boot device, it measures the IPL code, and if the code returns to the BIOS to find another device, the BIOS measures the IPL code of the next device. Since measurements are extended and not replaced, the final value of PCR4 depends on the order in which boot attempts are made. Thus you will always want to place your primary boot source first in the boot priority. Also remember that booting from a CD/DVD or from a USB will produce different digests, and you will want to make sure to include those if you need to boot a "trusted OS" by multiple methods.

The authors are not aware of any OS/VMM installation/reinstallation that requires a secure launch. So this is not likely a consideration—if you run into such a problem you will need to change the policy. Consider setting LCP=ANY during the installation and then resuming with PCONF policy.

PCR 5 holds the measurement of any configuration data that the IPL code might need (such as disk geometry). In many cases, there is no additional data needed (such as booting from a single partition disk). Also, it is the IPL code, not the BIOS, that extends the configuration data into PCR5. Thus, if the IPL code is not TCG compliant or there is no additional configuration data, this PCR may not be very useful.

PCR6: State Transition and Wake Events

This PCR gets extended by both the BIOS and the OS. The BIOS extends this register on wake events and the OS extends it on power state change. Here are the expected actions (per ACPI power state):

- *S0*: Normal operation, platform fully operational.
- *S1*: Standby w/ Low latency wakeup. This is typical of the platform reducing the processor clock rate to save power. TPM remains powered so PCR values are preserved. This can happen automatically without OS or BIOS knowledge. All contexts are maintained. The chain of trust is not broken, and thus nothing is extended to PCR6 upon entering or exiting this state.
- *S2*: Standby with CPU context lost. This is typical of an OS-involved, reduced-power state taking one or more cores offline during idle periods. BIOS is not involved and the TPM remains powered so PCR values are preserved. All other contexts are maintained. The chain of trust is not broken, and thus nothing is extended to PCR6 upon entering or exiting this state.
- *S3*: Suspend to RAM. This is where the OS calls the BIOS to put the platform in a deep sleep state, preserving RAM contents but powering down other components, including the TPM. The TPM is instructed to save its state, which includes the PCR values. When the platform wakes, the BIOS ACM detects that the platform is waking from S3 and restarts the TPM, causing it to reload the original PCR values. The chain of trust is interrupted but not broken as long as the resume is successful. The OS (optionally) extends PCR6 with the TPM_SaveState event immediately before entering S3, and when successfully resuming from S3, the BIOS ACM extends PCR6 with the TPM_ResumeState.

- *S4*: Suspend to disk. This is where the OS saves its context to disk and powers down the platform. When the platform wakes from S4, it goes through a normal startup and loads the OS, which recovers its previous state from disk and resumes where it left off. BIOS and TPM don't do anything special when waking from S4. Thus the chain of trust terminates on entering S4, and upon waking from S4, all PCRs are reset to their initial values and the chain of trust is rebuilt. There is no need to extend anything to PCR6 on entering or exiting S4, because the PCR will be reset.
- *S5*: Power OFF. This is where the OS simply powers down the platform (or power is lost, or the platform is reset). When the platform wakes from S5, it goes through a normal startup. BIOS and TPM don't do anything special when waking from S5. Thus the chain of trust terminates on entering S5, and upon waking from S5, all PCRs are reset to their initial values and the chain of trust is rebuilt. There is no need to extend anything to PCR6 on entering or exiting S5, because the PCR will be reset.

Including PCR6 in the PCONF policy is not very useful for servers. Server platforms don't support S3, and the PCR6 value does not change for the other power states. Workstations do support S3; however, workstations are outside the authors' area of expertise and we cannot offer any insight on the value of PCR6 for a workstation's PCONF policy.

PCR7: Host Platform Manufacturer Control

The content of this PCR is not currently defined and reserved for the platform manufacturer. Unless you know what is being measured into this PCR, you should not include it in the PCONF Policy.

Tools Needed for Creating a PCONF Policy

The primary tool will be the LCP policy generator, as described in the previous chapter. You will also need a tool that can gather the needed PCR values for input to the LPC policy generator, such as PcrDump. PcrDump is a tool that executes on the target platform, reads the values in all PCRs, and saves them to a file. You may want to consider naming the file after the platform and BIOS version (example: DellR410v18.pcr). You only need to do this once for each group of platforms that will have identical PCR values. That is, all identical platforms with the same BIOS version are considered a group. Copy the PCR dump file to the working directory for the policy generator. When you run the policy generator tool, specify which PCRs you want to include, plus the PCR dump file name(s), and the tool generates the PCONF policy element by creating a PCRInfo structure for each PCR file specified, as illustrated in Figure 4-4.

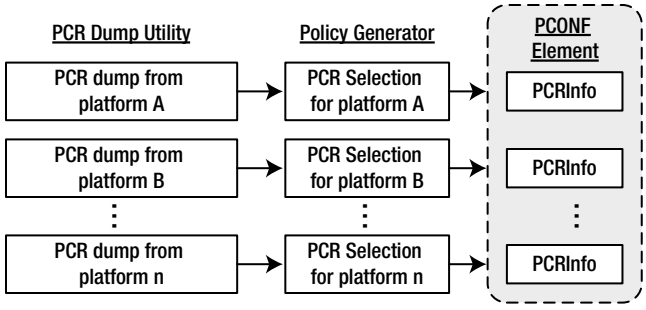

Figure 4-4. *Building the PCONF element*

The PCRInfo is simply a structure that identifies which PCRs were selected, and contains a composite hash digest created by hashing the selected PCR values. When the policy engine executes, it reads each PCRInfo to determine which PCRs to evaluate, performs a hash of those selected PCRs (using the current value from the TPM), and compares that digest to the composite hash digest in the PCRInfo. The digest will only match if all of the selected PCRs' values in the TPM are the same as the corresponding values in the PcrDump file used to create the PCRInfo.

The LPC policy generator builds a policy list that includes that PCONF policy element and provides the means for you to sign that list (you specify the key file that contains the public and private signing keys). The final step is for the LPC generator to generate the policy data structure and the NV policy data (for the portion of the policy that is stored in the TPM NVRAM, which contains the measurement of the policy data structure).

You will need a different tool for updating the TPM NVRAM (if necessary; see the discussion on using signed policy lists later in this chapter). That capability is typically provided by the OS/VMM (if the OS/VMM is the TPM owner) or by the management software you used to establish TPM ownership.

Difficulties with Using PCONF Policy

Hopefully by now you have a good idea of which PCRs you want to include in your ideal PCONF policy. PCR0 should be one of them (and possibly the only one). But there are challenges with specifying a PCONF policy.

As we stated in the previous chapter, one of the biggest challenges in maintaining a policy that includes PCR0 is that each BIOS update changes the value in PCR0. Thus, the PCONF policy must be updated to include the new "known good" value.

Here's the catch: if you update the BIOS before you update the policy, then the platform will fail its measured launch. To update the policy before the BIOS update, you must already know the new PCR value. Some solutions follow.

One solution is to update the machine offline (where it is not vulnerable) with a policy of ANY (or don't perform a measured launch), capture the new PCR values, update the policy, and push the policy to all platforms that will get that BIOS update. You will need to make sure you keep the old platform configuration in the PCONF policy to permit platforms to continue to launch before the BIOS update and to allow fallback in case the BIOS update fails.

Another solution is for the OS/VMM to perform a sanity check before it does a measured launch. The sanity check consists of testing if the policy is satisfied. If not, it notifies the user and prompts the user to accept the configuration change. If the user selects "accept," the OS prompts for authorization to accept the change (example: the TPM password or system administrator's password). This sanity check works for any platform configuration change. The tool simply looks at the existing PCRInfo to determine which PCRs the platform owner had selected, and uses that set of PCRs for both the sanity check and the PCONF policy update. For this case, each BIOS update (or any change to the platform configuration) results in a simple authenticated acknowledgement.

Unfortunately, we don't know of any OS/VMM that currently implements that capability. Perhaps you might want to talk to your favorite OS/VMM vendor about the value of providing such a capability.

Another potential solution (when only PCR0 integrity is needed) comes from relying on the PS policy (the one that the platform manufacturer provides). Many OEMs are planning on using signed BIOS policy, which allows them to provide an SBIOS element containing the list of known good SBIOS measurements. It would be fairly easy for them to include a PCONF policy element that includes only PCR0 measurement(s). The great thing about this solution is that it is automatic. That is, the BIOS update includes the new PS policy data structure, which authenticates the BIOS update. All that you would need to do is include a PCONF policy element in your Platform Owner policy data structure. It doesn't even have to contain any PCRInfo because the presence of the PCONF policy element tells the policy engine that PCONF must be satisfied, and allows the PCONF policy element in the PS policy to satisfy that requirement. However, not all OEMs support signed BIOS policy, and those that do might not support this signed SBIOS+PCONF concept.

Specifying Trusted Host Operating Systems

The MLE policy is much more intuitive because it involves only a simple measurement. That is, the measurement of the OS/VMM's trusted boot (MLE) code as calculated by the SINIT ACM during the measured launch process. Even though the value measured by the ACM is extended into PCR18, it cannot be learned by reading PCR18 (remember that PCRs are extended, not written). However, it can be learned by evaluating the PCR log for PCR18. PCR logs identify what is measured into the PCRs. The first measurement extended to PCR18 is the MLE measurement. It is that first extended value that must match a value that is listed in the MLE policy element in the policy data structure.

Another option would be for the OS/VMM vendor to provide that value. This could be done a number of ways. My favorite is for the OSV/VMM to provide a default signed policy data structure that includes valid MLE measurements. Here is how that works (if supported by the OSV):

- OSV/VMV creates a policy data structure (OS PDS) that includes a single MLE policy element, which lists the valid MLE measurements—typically, only a single measurement is needed.
- When the OS/VMM is installed, the OS creates the PO policy in the TPM NVRAM specifying PolicyType = LIST and the list hash of that OS PDS.
- If you desire a policy that only includes MLE and not PCONF policy, then you do nothing; otherwise, you create the Owner PDS using the MLE policy element from the OS PDS and your own PCONF policy element. Both the OS PDS and the Owner PDS are located in the boot directory. You update the PO_POLICY data in the TPM NVRAM to specify the hash of your Owner PDS.
- When the OS does the measured launch, it uses the Owner PDS if it exists; otherwise, it uses the default PDS.
- Each OS/VMM update provides a new OS PDS containing the new list of valid MLE measurements.
  - If you have chosen to use the OS PDS, then no action is required (unless you want to revoke any previous versions).
  - Otherwise, you need to update your Owner PDS with the MLE values in the new OS PDS. If you had chosen not to use a signed list, then you also have to update the NV policy data with the new Owner PDS measurement.

Tools Needed for Creating MLE Policy

The primary tool is the LCP policy generator, as described in the previous chapter. The authors do not know of any specific tools for harvesting MLE measurements. You cannot use the PCR18 value, because the value needed for the MLE policy is the value that gets extended to PCR18, not the result after extending it. Typically, these measurements should come from the OS/VMM vendor, so check with your vendor on how they distribute that information.

The LPC policy generator builds a policy list that includes that MLE policy element and provides the means for you to sign that list (you specify the key file that contains the public and private signing keys), as illustrated in Figure 4-5. Note that if you choose to include a PCONF policy element, you may include it in the same list, or you may choose to use a different list (especially if the PCONF list is signed using a different signing key).

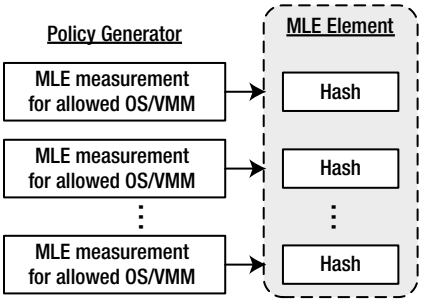

Figure 4-5. *Building the MLE element*

The final step is for the LPC generator to assemble the lists to generate the policy data structure and the NV policy data (for the portion of the policy that is stored in the TPM NVRAM, which contains the measurement of the policy data structure).

As mentioned previously, you will need a different tool for updating the TPM NVRAM, and that capability is typically provided by the OS/VMM (if the OS/VMM is the TPM owner) or by the management software that you used to establish TPM ownership.

Options and Tradeoffs

It's useful to think of launch control policy as having four parts. You are probably thinking PCONF policy, MLE policy, and SINIT policy—so what is the fourth part? Think of the PCONF policy as consisting of OEM-controlled BIOS values (PCR0) and owner-controlled configuration values (PCR1–6). The reason to do this is because BIOS updates are associated with the OEM. You might control when BIOS updates are applied, but the OEM controls the valid PCR0 values. The values for the other PCRs are controlled by actions of the datacenter and are affected by changing BIOS settings, adding/removing I/O devices, changing boot order, selecting the boot device, installing the OS/VMM, and so on.

Impact of SINIT Updates

Revocation of the SINIT ACM is rare. It happens because Intel discovers a security flaw, and that has only happened once in the history of Intel® TXT. In case of an ACM revocation, the ACMs have their own mechanism and don't rely on LCP SINIT policy. With that being said, there are other reasons for generating new versions of the SINIT ACM. Typically, it is to add or enhance features (for example, support of signed policies) or to fix a feature that didn't work as expected.

The good news is that SINIT policy is satisfied with newer SINIT ACMs. That is because ACMs are signed modules whose integrity and authenticity are validated by the processor's microcode. The only reason that you would need to change the SINIT policy is to prevent using an older SINIT. We could make guesses about why you would want to specify a SINIT policy other than *All Versions,* but the reality is that so far no one has found a need to do so. In the event that you have a reason, it requires you to take one of the following actions:

- Update the Min SINIT Version in the TPM NV policy data on each platform.
- Update the Min SINIT Version in the MLE policy element of your policy data structure. You would want to use this method if you use signed policy, because you would avoid having to modify the TPM NVRAM for each platform. However, if your MLE policy element is not in a signed list, then changing the MLE policy element requires updating the TPM NVRAM with the new hash of the policy data structure.

Impact of Platform Configuration Change

You are the only one who can predict how often you make changes to the platform configuration. Typically, it does not happen very often. In most cases, it requires taking the platform offline to perform the modification. When this does occur, you will need to update the PCONF policy element in your policy data structure, but only if you specified a PCONF that specifies more than PCR0. This will also require updating the TPM NVRAM on each platform if you don't use a signed policy list.

Impact of a BIOS Update

It's likely that BIOS updates will happen much more frequently than configuration changes. When a BIOS update does occur, you will need to update the PCONF policy element in your policy data structure (if you have one). This will also require updating the TPM NVRAM on each platform if you don't use a signed policy list.

Impact of OS/VMM Update

Typically, this happens frequently. When the OS/VMM update does occur, you will need to update the MLE policy element in your policy data structure (if you have one). This will also require updating the TPM NVRAM on each platform if you don't use a signed policy list.

Managing Launch Control Policy

It would be nice if you could just set the launch control policy and never have to touch it again. But we have seen that anytime you make a change (update the BIOS, update the OS/VMM, change the platform configuration) you need to update the policy. Dealing with one or two platforms is not a significant challenge, but dealing with hundreds or thousands of platforms from different vendors over multiple platform generations is quite challenging. Here are some tips on how you might manage them.

Think Big

It is easier to manage a single policy for all servers.

- Create the policy data structure to contain PCONF measurements for all platforms instead of managing platforms individually. Measurements are essentially unique. Thus, the probability of a measurement of a corrupted BIOS or misconfigured platform being the same as a known good measurement on a different platform can be considered nonexistent. While not required, the set of PCRs included should be the same for all platforms. This will make managing PCONF easier and allow use of PCONF management tools that automatically detect the set of PCRs that you selected. If you have a reason for using a different set of PCRs on different platforms, by all means do so.
- You might want to create multiple security domains and manage each domain separately. By security domain, we simply mean a group of servers that have a common set of security requirements and use the same set of resources including boot services. This allows you to use a different set of launch policies for each domain while using automated procedures for updating the launch control policy.

- Unless you want to prevent a particular known-good OS/VMM from launching on some platforms while allowing it to launch on others, then you can use the same MLE policy on all platforms. If you do want to enforce restricting OS/VMM on a per platform basis, then most likely you already manage servers as groups (grouped by OS/VMM). In this case, create a different policy for each group.

Use a Signed List

Using signed rather than unsigned lists allows you to update the policy without having to touch the TPM. This is because the measurement of the policy data structure has to match the measurement stored in the TPM NVRAM. The measurement of an unsigned list is the hash of the entire list. So any change to that list causes a change to the policy measurement. However, the measurement of a signed list covers just the public key that is used to verify that the list has not been compromised. Thus, updating a signed list does not change the policy measurement (as long as the list is signed with the same key).

If your servers use network boot, then all you need to do is update the policy data file on the network boot drive(s). Otherwise, use a utility to push the updated policy file to the servers.

Make Use of Vendor-Signed Policies

Make use of signed lists provided by the OS/VMM vendor. The policy data structure allows for up to eight lists and each list may be signed independently from the others. For example, you could create and sign a list that contained just a PCONF policy element, and then create the policy data structure consisting of that list and the signed list provided by the OS/VMM vendor that contains the valid MLE measurements. This would require extracting their list from their policy data structure, but that is a simple task.

Use Multiple Lists for Version Control

Consider using multiple lists for version control. Note that the reason for removing old measurements from your policy is to prevent rollback to those versions. As an example, let's say all platforms have OS/VMM version *x* and you want to update to version *x* + 1. You want to allow version *x* and *x* + 1 until you have updated all platforms. After you have updated all platforms, you wish to only allow version *x* + 1. Let's also assume that you use the signed lists provided by the OSV/VMV so you do not have to sign the MLE lists.

You initially create your policy to contain three lists—one with a PCONF policy element and two (identical) signed lists that contain an MLE policy element for version *x*.

When the OSV/VMV releases version x + 1, you replace the first signed MLE list with a new signed list for version *x* + 1 and distribute. Now both versions are allowed.

When you complete your updates, you update the policy replacing the second signed MLE list (that is, the one for version *x*) with another copy of the signed list for version *x* + 1. The reason that you update with a duplicate instead of just removing the list for version *x* is that you want to preserve the policy measurement.

This example assumed that the OSV/VMV provided a signed list with an MLE element. If that is not the case, then you will need to sign your own list. However, that allows you to include a PCONF element in the list. Thus you only need two lists.

The same logic holds true for using two PCONF lists (or combined PCONF/MLE lists) for managing BIOS updates or configuration changes.

There is one problem with using signed lists for version control. That is, if an attacker is able to replace your policy file for version *x* + 1 with a copy of your version *x* policy file, the attacker prevents version *x* + 1 from launching. In addition, if the attacker is able to cause the platform to boot version *x* instead of *x* + 1, the attacker can exploit any security vulnerabilities that existed in the previous version. This is referred to as a *rollback attack*.

You can prevent a rollback by using the revocation counters. The TPM NV policy data contains eight revocation values (one for each possible list). Each time you sign a list, the tool automatically updates the revocation counter in

the list's signature block. Setting the revocation value in the TPM NVRAM to the value of the current list prevents older lists (lists with a lower revocation count) from being used. If malicious code was successful in replacing the policy data file for version $x + 1$ with an older version allowing version x, it would prevent the launch of version $x + 1$ (denial of service), but launching version x would not be permitted. If an attacker is able to get in and manipulate files in your boot directory, then you have bigger problems than a denial-of-service attack. The bottom line is that Intel TXT detects the attack and that the launch failure serves as an alarm.

Using the Simplest Policy

The more complex the launch control policy, the more difficult it is to manage it. Using a policy of ANY is very simple to implement and requires no maintenance. But it also offers the least amount of protection. The most protection comes from using both a PCONF policy and an MLE policy, but there is a tradeoff between maximizing your protection and manageable policies.

So take your time in evaluating what it means if you exclude a particular PCR from the PCONF policy. Is remote attestation in place to take up the slack? Let's take an example.

Excluding PCONF altogether means that the platform is allowed to launch with any platform configuration and any BIOS. This affords the least protection against reset attacks and root kits. However, early remote attestation can prevent any damage that can be done by a corrupted BIOS or a misconfigured platform if it is able to immediately quarantine the platform or prevent it from joining the production network. Such an action should prevent the platform from accessing protected data, and even if it does, the malicious software is not able to communicate that information externally.

Thus, in this case, the need for the PCONF launch policy is negated. The same holds true for the MLE launch policy. Several security management vendors have demonstrated remote attestation products that do quarantine platforms that are out of compliance. Such applications are expected to be released starting late 2013.

Other Tips

Here are some steps that you can take to make managing launch control policies easier.

- For platforms of the same model, use identical platform configurations. This includes using the same add-on devices if those devices contain option ROMs.
- Keep platforms updated with the latest BIOS version. Not only will this help keep platforms consistent, but it assures that you have the latest security patches, ACMs, and microcode patches.

Strategies

So let's factor in your learning curve, availability of tools, and completeness of solutions.

During your evaluation phase, it should be sufficient to start with the default policy established by the OS/VMM. Most likely it is a policy of ANY, or perhaps the OS/VMM vendor provides a signed MLE Only policy. This will allow you to establish trusted pools and evaluate various remote attestation solutions without the burden of setting up and maintaining a launch control policy.

Whether or not this is the right policy going forward depends on a number of factors:

- *Confidence.* Starting out with a policy of ANY and switching to a specific policy at a later time after you have evaluated various policies is a viable choice, even when other factors indicate a more complex policy is desired.
- *Training.* For both you and your staff. Knowing when a policy is likely to fail, the impact of a failed policy, and the recovery actions need to be well known by everyone who maintains the datacenter.

- *Available tools.* Some OS/VMMs provide more complete solutions than others. But they are still on their learning curve and continue to expand their capability with each release. This means that the tools and mechanisms that you need to manage a launch control policy might not be available, or might be very crude and difficult to use.
- *Risk.* More complex policies add the risk that you or your staff could make a configuration change without updating the policy, and thus cause platforms to drop out of the trusted pool or even become temporarily inoperable until the policy is updated.
- *Trusted Pools.* Are they based on remote attestation or on a launch control policy? This is becoming less and less of a problem because remote attestation solutions are becoming more prevalent. Some of the earlier management applications relied solely on a platform's ability to pass a launch control policy as the criteria for it to be added to the pool of trusted servers. This requires a stringent launch control policy. As more and more remote attestations services are introduced, management software will most likely migrate toward using remote attestation services, because they offer more flexibility and capability. At least for managing public clouds. For private clouds, a launch control policy may remain the choice because of its simplicity.
- *Reset protections.* Reset attacks can still occur even with remote attestation. Platforms that use autopromotion for the SBIOS policy rely on the launch control policy to establish if the BIOS is trusted. Thus, a PCONF policy provides greater protection against reset attacks because BIOS trust is established before secrets are placed in memory. This is not an issue for platforms that implement signed BIOS policy because the platform manufacturer provides the known good BIOS measurements.
- *Remote attestation.* By itself, remote attestation does not replace launch control policy. Initially, most remote attestation applications are looking at the dynamic PCRs. This should evolve to cover static PCRs (that is, platform configuration). As applications that rely on remote attestations evolve, they will be able to provide the same protections as the launch control policy, but only if their policy is set to evaluate all of the PCRs that you feel are pertinent. This means that over time, launch control policy could become less important. Before you can dismiss the importance of PCONF policy, you will need to evaluate your management applications for their ability to detect BIOS and configuration changes, as well as mitigation of BIOS corruption.

It is recommended that you start with the default policy and migrate to a more restrictive policy as soon as the proper tools are available. As remote attestation capabilities evolve, you will be able migrate back to the default policy to reduce the burden of policy management.

Starting with an MLE policy and PCONF policy of PCR0 is considered sufficient by many. So this should be your first step toward a more restrictive policy. If you are paranoid (and remember that "only the paranoid survive") you would want to include all the PCRs. Here are some reasons that you might want to exclude them:

- *PCR0*. Always include.
- *PCR1*. Many variations make policy management exceptionally difficult.
- *PCR2* and *PCR3*. You can't control when option ROM code is updated, and thus must avoid such an update from making platforms unavailable.
- *PCR4* and *PCR5*. Any boot source is okay because MLE policy allows only an authorized OS/VMM to perform a measured launch.
- *PCR6*. Not useful because your servers don't support S3 sleep state.
- *PCR7*. Not useful because it is not defined.

- *Any PCR that does not maintain a consistent value from launch to launch*. Excluding that PCR is only a short-term solution. Contact the OEM and ask for a BIOS update that will fix the problem since TCG requirements forbid dynamic values from being extended into PCRs.
- *Any PCR that yields a different value on identical platforms*. Excluding that PCR is only a short-term solution. Contact the OEM and ask for a BIOS update that will fix the problem, because TCG requirements forbid platform identity (or anything that could be used to identify an individual platform) from being extended into PCRs.

Impact of Changing TPM Ownership

There is a difference between *changing* the TPM owner's authorization *value* (TPM password) and *resetting* the TPM password. Changing the TPM password requires knowing the current password and can be performed as many times as necessary without impacting the launch control policy or other TPM resources (other than changing their authorization).

Resetting the TPM password (referred to as *clearing* the TPM) is performed via the BIOS (or the TPM owner) and will invalidate every TPM object that uses owner authorization. This means that the Platform Owner policy in the TPM NVRAM will be deleted. Thus platform owner policy will revert to the platform supplier policy, which is typically ANY or "ANY MLE plus PCR0 matching the signed list provided by the OEM." In any case, it is not the policy that you had established. Your policy data structure still exists, but you will have to re-create the Platform Owner policy in the TPM NVRAM. But that can only be done after taking ownership of the TPM and establishing a new TPM password.

Now you might think that the only reason that you would do that is if you forgot the TPM password (or wanted to sell or otherwise remove the platform from the datacenter). There are other reasons:

- *Switching to a different OS/VMM*. Some OS/VMMs establish exclusive TPM ownership (such as VMware ESXi) and do not allow the TPM password to be known. In order to uninstall such an OS/VMM and install another most likely requires resetting the TPM password. If you don't clear the TPM, you will have to use the original OS/VMM's tools for managing the TPM (if that is even possible). This implies that you will need to maintain a license for the original OS/VMM and that you will have the limitations of those original tools, which are most likely crafted for the needs of the original OS/VMM.
- *Reinstalling the same OS/VMM*. Note that ESXi does not have this problem. If you reinstall it, it detects that it already has exclusive ownership, and thus installs and executes with full Intel TXT capability. Other OS/VMMs might not be as accommodating, and thus require clearing ownership in order to install.
- *In theory, any OS/VMM that takes exclusive ownership of the TPM might be subject to inadvertent corruption of the TPM password*. For example, let's say the TPM password was sealed to the wrong value, or the key that the OS/VMM used to seal the password was inadvertently deleted or corrupted. This would be equivalent to you forgetting the TPM password, and thus require having to clear the TPM and reestablish TPM Ownership.

If for any reason you are required to clear the TPM (or do it accidentally—there is no un-clear), then you will have to reestablish TPM ownership and re-create the Platform Owner policy in the TPM NVRAM.

Decision Matrix

It would be nice if we were able to provide you with a decision matrix that would tell you which launch control policy is best. But that is not possible because it is too difficult to quantify the value of various protections. The best we can do is to provide you with a list of questions for you to consider when determining your launch control policy. The term *OS/VMM default policy* refers to the LCP established by the OS/VMM when it is installed. If OS/VMM does not establish a default policy, then the default would assume that you created a platform owner policy of ANY.

Here is the list of questions for you to answer to help you select the appropriate policy:

- Is remote attestation in place that mitigates the need for a launch control policy? If so, will the OS/VMM default policy be acceptable? Do any of the remote attestation applications rely on launch control policy?
- What is the default OS/VMM policy? Is it ANY or does the OS/VMM provide the list of known good MLE values?
- Do you have the tools to create and manage your own policy? If not, can you use the OS/VMM default policy until you acquire the proper tools?
- How important is it to protect against platform configuration changes? What is the risk that such an attack could occur? Is including PCR1 needed? Sufficient? Or do you need to include PCR4 and PCR5?
- How important is it to protect against unauthorized BIOS modification? What is the risk that such an attack could occur?
- How important is it to protect against an unauthorized OS/VMM performing a measured launch? What is the risk that this could occur?
- How difficult is it to update the policy when there is a BIOS update? An OS/VMM update? A configuration change?
- What is the risk of not updating the policy when there is a BIOS update? An OS/VMM update? A configuration change? Will it result in excessive downtime? What percentage of platforms would be impacted?
- Can you live with the result if any of these risks are realized? Would your company's reputation be tarnished? Would your company survive? Would your job be in jeopardy?

No one said that managing launch control policy was going to be easy, and you will most likely have to make tradeoffs. If you choose anything less than PCONF(PCR0) + MLE, then make sure you can justify your decision.