OpenPOWER secure and trusted boot, Part 1: Using trusted boot on IBM OpenPOWER servers

Making your system safe against boot code cyberattacks

Dave Heller
Tim Block
February 17, 2017

IBM OpenPOWER servers provide a firmware-level security feature known as Trusted Boot. Trusted Boot helps defend against a boot code cyberattack by helping to verify that your server is running only authorized firmware. Integrity of your firmware is vital to the security of your system. Trusted Boot works by taking measurements of the executable boot code as the server boots and recording these measurements to a dedicated hardware security module known as the Trusted Platform Module (TPM). Together with a process known as remote attestation, you can use the data in the TPM to verify the integrity of your server’s boot code.

Introduction

IBM® OpenPOWER servers provide a firmware-level security facility known as Trusted Boot. Trusted Boot helps you to verify that your server is running only authorized firmware components from IBM or another trusted vendor. This allows you to detect and take corrective action in case of a boot code cyberattack – that is, any attempt to replace your trusted firmware with malicious code. If an attacker can inject malicious code at the firmware level, no amount of protection within the operating system can prevent the attacker from gaining control of your server.

“Trusted boot allows you to detect a boot code cyberattack – that is, any attempt to replace your trusted firmware with malicious code.”

TPMs, PCRs, and integrity measurements

Trusted Boot works by requiring the firmware to take a series of recordings, or measurements, as the server boots. Each measurement is a secure hash (for example, SHA1, SHA256, or SHA512) of a particular boot code component (an executable firmware image) as it is loaded from flash memory, before it runs on the system processor. Each executable image measures the next before passing control to that image. The measurement may also be a hash of some important configuration data, such as the properties that determine the server’s default boot device.
The measurements are recorded in a dedicated security processor known as the Trusted Platform Module (TPM). The TPM ensures that the measurements are stored securely, in a manner where they cannot be erased (until the next reboot) and cannot be easily counterfeited. The TPM has several dedicated registers, called Platform Configuration Registers (PCRs), allocated to hold the measurements. Each PCR contains a cryptographic history (in the form of a hash value) of all the measurements extended to the PCR. The extend is a special operation used by the TPM to add a measurement to a PCR – more on extends shortly. The important thing is: the TPM ensures that a specific series of measurements, in a specific order, will always produce the same resultant value—the digest value—of the PCR. And, it is virtually impossible to produce a given digest value without having the TPM extend that exact series of measurements, in the exact order.

After the server boots to the target OS or hypervisor, it is possible to connect over the network and ask the server for a list of all PCR values and a list of all the measurements that were recorded by the TPM. The list of measurements is referred to as the boot-time measurement list or event log and the list of PCR digest values is called the PCR digest list. The process of asking the TPM for this data is known as requesting a quote.

“The Trusted Platform Module is a dedicated security processor designed to hold integrity measurements.”

When the TPM creates the quote, it cryptographically signs the digest list in a manner that can be independently verified, using a key that can be validated as belonging to the unique TPM that created the quote. This key, in turn, is signed by a key that can be linked to the TPM’s manufacturer or vendor. The key used to link the TPM to its vendor is known as the Endorsement Key (EK), and the key that is used to create the quote is known as the Attestation Key (AK).

After the quote has been retrieved from the TPM and the EK and AK verified, the PCR data can be used to verify that the server has booted only the expected boot code and configuration. Any deviation will create an unexpected PCR digest value, and possibly an unexpected event log entry, which can be detected when examining the quote. This process of retrieving and verifying a quote is known as remote attestation.

How Trusted Boot protects you

So, how does this help you ensure that your system is secure? The assumption is, if you can be sure that your system executed only the expected boot code, using only the expected configuration, you can be sure that your system has not been compromised—at least as far as the boot code is concerned. The TPM, and the process of retrieving the quote, help you ensure that this is true. If any unexpected image or configuration was loaded, the TPM would produce an unexpected digest value in at least one PCR. As soon as you processed the quote, you would know something had changed.

Now, this is not to say the reverse is true: that an unexpected measurement necessarily means your system is hacked. The system may have booted a valid firmware image, just not the one you were expecting (that is, a different version of an authorized image). Or, if the difference is due to
a configuration change, that change may not have resulted in a difference in the way the system booted. For example, if you set the system to boot from the network and there is no PXE server found, the system will likely proceed to its next choice and boot whatever medium it did before. This points out that it may be helpful to know not just that there was a change, but exactly what was changed. More on this when we delve deeper into remote attestation.

“If you can verify your machine has executed only trusted firmware, you can be sure your machine has not been compromised during boot.”

For the moment, it is important to understand that any change in the boot code or boot configuration will produce a change in one or more PCR values that will be detectable in the quote.

TPMs usually contain 24 or more PCRs, the first 16 of which cannot be reset without a reset of the system processors, as will occur during a cold boot. The quote returns all PCR values requested, typically PCR \([0-15]\), in the signed digest list. You can associate a digest list with a known-good configuration: a so-called reference or golden configuration. By remote attestation, you can compare the way a system booted to a reference configuration for that system, and if they match, you can be confident that your boot has not been compromised.

**The extend operation**

Before continuing it is helpful to explain how the extend works. The TPM is a cryptographic processor with some special characteristics. One of these characteristics is that the TPM's PCRs cannot be written directly; they can only be updated through the extend operation. The extend operation passes a chunk of data—typically a measurement (hash) of a component image or file—to the TPM along with the PCRs to be updated. The TPM concatenates the new data with the current register contents, performs a hash (using an algorithm such as SHA1, SHA256 or SHA512) and writes the result back to the PCR. In this way, the new PCR value is always dependent on the previous value. This is what ensures that the same sequence must be followed to produce the same result. The notation for the extend operation is:

\[
\text{digest new} \equiv \text{hash} (\text{digest old} \ || \ \text{data new})
\]

where \text{hash()} represents the operation of the secure hashing algorithm, and \(\ || \) means concatenation.

A difference between TPM 2.0 and the prior standard, TPM 1.2, is that TPM 2.0 supports multiple banks of PCRs, each supporting a different hash algorithm. Early in the boot process, one or more of these banks is designated as active. When measurements are recorded to the TPM, the extend operation is performed for all active banks.

**How PCRs are used**

The standard usage of the TPM PCRs reserved for firmware measurements is defined by the Trusted Computing Group (TCG) in the "PC Client Platform Firmware Profile Specification". The
lower-range PCRs [0-7] are reserved for the so-called pre-OS environment and the upper range PCRs [8-15] are reserved for the host platform's target operating system or static OS.

Table 1 shows the PCR usage for the lower-range PCR [0-7]. The first two PCRs, [0-1], are used to measure components of the base firmware such as Unified Extensible Firmware Interface (UEFI) or OpenPOWER firmware. PCRs [0-1] are said to be under Platform Vendor control. PCRs [2-3] are used to measure third-party vendor firmware such as UEFI drivers, or IBM Coherent Accelerator Processor Interface (CAPI) microcode for IBM POWER8® processor-based systems, and are said to be under Add-in Vendor control. PCRs [4-5] are dedicated to measurements of the target OS or hypervisor plus any associated bootloader code: the so-called boot state transition.

PCRs [6-7] are used a bit differently between UEFI and OpenPOWER. On UEFI, PCR [6] is reserved for general platform vendor use, and PCR [7] is reserved for Secure Boot Policy. On OpenPOWER, PCRs [6-7] are reserved for potential future IBM POWER® protocols.

Table 1. OpenPOWER PCR usage

<table>
<thead>
<tr>
<th>PCR role</th>
<th>PCR index</th>
<th>PCR usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host platform vendor control</td>
<td>0</td>
<td>Hostboot and other firmware components</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Configuration data and firmware container metadata</td>
</tr>
<tr>
<td>Add-in component vendor control</td>
<td>2</td>
<td>CAPI code</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>CAPI data</td>
</tr>
<tr>
<td>Boot state transition</td>
<td>4</td>
<td>OPAL firmware, Static OS (Linux kernel and initramfs)</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>TPM enabled flags, OPAL container metadata, boot sequence, static OS configuration (Linux kernel command line)</td>
</tr>
<tr>
<td>Reserved</td>
<td>6</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>

Establishing the core root of trust

What may be apparent from the description so far is that you must be able to trust the components that create these measurements, or else the system from which you are retrieving the quote may already be compromised and could be deceptive about what was measured. This problem is solved by establishing a core root of trust for measurement (CRTM) anchored in hardware. Figure 1 shows a simplified view of how CRTM establishment works on OpenPOWER.

The boot process begins by running a tiny bit of code from the host POWER8 master processor's on-chip one-time programmable read only memory (OTPROM). This code is called the self boot engine (SBE) and is part of the POWER8 processor’s power on reset engine (PORE). Because it resides in the OTPROM, it is immutable and cannot be overwritten by an attacker.

“The Core Root of Trust for Measurement (CRTM) provides a basic level of trust in your machine’s boot firmware.”
The OTPROM code provides an entry point to another executable SBE image stored in the serial electrically erasable programmable read-only memory (SEEPROM) located on the POWER8 processor module. This SBE now begins loading additional executable images from processor-based NOR flash memory (PNOR). The first component to be loaded is called Hostboot. Hostboot is the first firmware component capable of performing an extend to the TPM, and Trusted Boot measurements start here.

It is important to understand that the first bit of code to run—the SBE code in OTPROM—is burned into hardware and cannot be overwritten by an attacker. You trust that to the extent you trust your hardware, meaning: your server was supplied by a trusted vendor who installed a genuine IBM PowerPC® processor. This is how the CRTM is anchored in hardware. However, every bit of code that runs subsequently is loaded from a non-volatile storage location that could be overwritten by an attacker. So, you need a way to validate each component that runs before Hostboot. To do this, you need a capability we haven't discussed yet: Secure Boot.

Trusted Boot compared to Secure Boot

Trusted Boot relies on Secure Boot to establish the core root of trust for measurement. The components that record measurements to the TPM must be trusted, and therefore, they must be verified before recordings begin. Secure Boot helps you do this.

Trusted Boot and Secure Boot have some qualities in common, and some differences:

- Secure Boot verifies firmware components during boot and stops the boot if verification fails.
- Trusted Boot records measurements to the TPM for later verification and continues the boot no matter what was measured.
- Secure Boot performs verification using cryptographic signatures.
- Trusted Boot enables verification using TPM quotes and Remote Attestation.

“Secure boot uses cryptographic signatures to verify your firmware.”

Secure Boot is similar to Trusted Boot in that both are used to validate critical firmware components. The difference is, Secure Boot performs the validation in place, during the boot, and will stop the boot if a validation fails; whereas, Trusted Boot makes a recording of the component for validation later and allows the boot to proceed regardless of what is measured. Trusted Boot is valuable for the reasons explained in this article. But because the boot process is not yet at the point that it can record measurements, it must rely on Secure Boot initially. Secure Boot uses cryptographic signatures to verify components. On OpenPOWER it works like this:

Stored in the processor's SEEPROM is a hash of the public portion of a set of hardware root keys. The private parts of these keys are held by the platform vendor (IBM or the original equipment manufacturer (OEM) vendor) and are used to sign any authorized firmware for the host platform. To be precise, the hardware (HW) root keys sign a set of firmware keys, which sign the actual firmware images. Each firmware image loaded by the SBE must pass a signature validation check before it can run. The SBE uses a bit of immutable ROM verification code to do this. Every executing module must signature-check the next before passing control. If any check fails, the Secure Boot function halts the boot. If the boot succeeds, you can be sure that every module was
verified and can be trusted (to the extent you trust the hardware root keys). So, by the time the system boots to Hostboot, you can be sure that your measurement code has been authenticated and you can trust what was written to the TPM. This is how the CRTM is established.

Figure 1. CRTM establishment process

One detail not shown in the diagram is how the SBE code in SEEPROM is validated. This code is stored with no signature data (SEEPROM space is limited). However, a copy of this same SBE code is stored in PNOR, and that copy includes the signature data. After the boot proceeds as far as Hostboot, Hostboot checks the SEEPROM code against the copy in PNOR. If the two do not match, Hostboot will update the SEEPROM code with the (verified) copy from PNOR, and immediately reboot the system. On the next boot, Hostboot performs the same check again. The check will now pass, and this ensures the system is running trusted SBE code.

Containerization of components

One detail left out of the previous discussion is how the signature data for CRTM components is made available at boot. The components are containerized, meaning they are stored in PNOR flash along with some metadata that includes the hash digest of the component and a signature over that hash. As each component loads the next, it checks the hash and validates the signature. As mentioned, HW root keys sign the firmware keys, which in turn sign the payload hash. The public portions of these keys are included in the container metadata, so they're available to perform the boot-time signature checks. As executable components are loaded from flash memory, if the hash of the HW keys in the container header does not match the hash in SEEPROM, the boot sequence is halted.
Note that the container metadata contains no private keys or secrets, only enough public information to establish the trust chain for each signed component. The private portion of the firmware keys are held by the firmware provider, and the private portion of the HW root keys are held by the platform vendor (IBM or the OEM vendor).

**Remote attestation**

Now that you have established your core root of trust for measurement and enabled a way for the TPM to send a quote, there’s quite a bit you can tell about the way the system booted. The process of retrieving and analyzing the quote is known as *remote attestation*.

Remote attestation is a client/server protocol. The client (or *agent*, as sometimes called) is the component that runs on the node you want to verify. The client accesses the local TPM through an API known as TCG Software Stack (TSS) and requests the TPM to prepare a signed list of its current PCR values to be sent to the attestation server. The client is also responsible for retrieving the event log from firmware and sending it along with the quote. The attestation server runs on a trusted, independent node somewhere on the network (such as a management server) and is responsible for verifying the quote and processing the retrieved PCR digest list and event log. The details of how this works are beyond the scope of this article but we’ll describe it in a nutshell here. This description is based on the operation of the open source IBM TPM 2.0 Attestation Client Server.

Using the IBM TPM 2.0 Attestation Client Server you can retrieve and verify the measurements from the TPM. The code is written in C, and uses a MySQL database and a PHP web interface. As of this writing, the IBM TPM 2.0 Attestation Client Server is in the *alpha* development stage and is not yet included in any Linux distribution. But it is easy to build and configure, and can run in an environment similar to what most distributions refer to as a Linux, Apache, MySQL and PHP (*LAMP*) stack environment.

"Using Remote Attestation you can retrieve and verify the integrity measurements from your TPM."

**Quotes and remote attestation**

A *quote* is a list of the current PCR values, signed by the TPM. *Remote attestation* is a client/server process that helps you retrieve a quote the TPM.

Before sending a quote, the client TPM must be *enrolled*. The attestation server:

- Verifies the client TPM Endorsement Key (EK) against the root EK from the TPM manufacturer
- Presents a challenge to the client that can only be complete if the client TPM is the owner of the EK and a unique Attestation Key (AK)

Once enrolled, the client can send a quote. The attestation server:

- Verifies the integrity of the quote using a secure hash of all PCR values
• Verifies the signature over the quote against the TPM AK
• Verifies the event log by replaying the measurements and comparing the calculated PCR values to those in the quote

Before continuing, it is necessary to cover some more details about how the TPM handles keys. As mentioned previously, every TPM contains a unique, burned-in Endorsement Key (EK) that is signed by a Root Endorsement Key belonging to the TPM vendor. This signature over the (public part) of the TPM's EK is stored in a X509 certificate, pre-installed in the TPM, and this is one of the first bits of information the client sends to the attestation server when sending a quote. The Root EK certificate is a publicly available X509 certificate that can be obtained from the TPM vendor and imported into the attestation server truststore. Now, when the client sends its EK certificate, the attestation server can easily check it against the root EK certificate and verify that this key belongs to a TPM manufactured by this vendor. (Note that the server does not yet know this key belongs to any particular TPM, only one from this vendor).

The TPM does not use the EK to sign the quote; it generates an Attestation Key (AK) for this purpose. The client sends the public part of this AK to the attestation server at the same time it sends its EK certificate.

It is important to note that the private parts of these two keys, the EK and the AK, are contained within the TPM and cannot be extracted. These keys can only be used by the TPM and only in a manner intended by the TPM designer. (This is why it is important to know that the TPM came from a trusted vendor.) If these keys could be extracted and used externally, it would be easy for another TPM like function to masquerade as this TPM and destroy the authenticity of the quoted information.

After the attestation server has received the public parts of the EK and AK, it can create a challenge to verify whether this client TPM is truly the holder of these keys. This allows the server to complete what is referred to as the **enrollment**, as shown in figure 2. In essence, the challenge is constructed so that the client must have the private portions of both keys to complete it. Additionally, the challenge is performed in a way that can only be completed on the client's TPM; that is, it cannot be performed in software on the client.

Contained in the challenge is a **secret** encrypted with the client's public EK. Also contained in the challenge is a reference to the client's public AK, known as the key **name**. The client TPM will reject the challenge if the **name** does not match the client's true AK. After the client decrypts and returns the secret, the attestation server can be sure that the client has performed the operation on a genuine, trusted vendor's TPM and that the Attestation Key can be trusted. When this is completed, the client is **enrolled** at the attestation server, and this is shown in figure 2.
Figure 2. Remote attestation enrollment process

Note that although the attestation server can now uniquely identify this TPM as the holder of the given AK, it still has no way of knowing that the system can be trusted, in the sense that this system belongs to your organization, and should in fact be on the network. In future, there will need to be some way for the client to be signed by some organizational root key, and the attestation server will need to have some organizational certificate installed in its truststore to enable it to trust the client. In the meantime some external step needs to be taken to ensure that the client can be trusted, and the server will need to be updated to flag the client as trusted, or implicitly trust each client that sends a quote.

Now that you have the necessary infrastructure to verify a signed quote from the client, and to ensure (by the CRTM) that the components that generated the measurements can be trusted, you can request a quote from the client whenever you want. And because the quote always contains the current PCR values, you can easily see if anything has changed since the last quote.

Event log replay and validation

We said that the quote contains the PCR digest list and the boot-time event log. The digest list is generated by the TPM, and signed by the TPM when sending the quote. The event log, however, is not managed by the TPM. The event log is managed by the platform firmware stack (that is, by
UEFI in the case of x86 and by OpenPOWER firmware in the case of PowerPC). For this reason, it is often referred to as the firmware event log. On UEFI systems, the event log is maintained in an Advanced Configuration and Power Interface (ACPI) table and the TPM is represented as a ACPI device object. On OpenPOWER, the event log is maintained in protected main system memory and the TPM is represented by an entry in the open firmware device tree.

Both UEFI and OpenPOWER systems use the Crypto Agile event log format for TPM 2.0 standardized by TCG. You can find a full description of the specification in the "Trusted Computing Group EFI Protocol Specification". Each event log record contains a value that was extended to a PCR (a measurement event) or some sort of non-extend event that serves as a separator between boot stages or possibly an error marker. On UEFI systems, the firmware event log is retrieved through the EFI TCG protocol. On OpenPOWER systems, the event log is retrieved through an entry in the device tree.

"Event log replay validates your event log by calculating the expected PCR values and comparing them to what's actually in the TPM."

So, the PCR digest list is signed by the TPM but the firmware event log is not. However, after you have a signed PCR digest list there is a way to establish the veracity of the event log: this is called event log replay, and it works as follows.

Each event log entry contains the hash value that was extended to the PCR as the measurement was taken. It is critical to the design of the firmware that an event log entry must be made for every PCR extension. As described previously, the extend updates the PCR using the input hash value and the current value of the PCR digest. If you know the complete list of extensions (you should, if you have the complete event log) and you know the starting value of the PCR digest (zero, when measurements start after a TPM reset), then you have sufficient information to replay the event log and re-create the expected PCR values. This replay can be performed anywhere; it does not require the client TPM (or any TPM at all). It is simply a matter of walking through the list of extends, in the expected order, to calculate what the final PCR value should be. If the calculated value matches the actual value contained in the quote, the event log must contain the correct list of extends for that PCR. If you do this for each PCR you have, in essence, validated your event log. The IBM TPM 2.0 Client Server performs this replay whenever you request a quote; if the replay fails, the attestation server will flag the event log as invalid.

Figure 3 shows the client/server exchange for retrieving the PCR digest list and event log. For simplicity, the diagram omits a few details. For example, the attestation server sends a 32 byte field of random data, called a nonce, when requesting the quote. The client includes the nonce in the signed data it sends to the server, and when decrypting the data the server verifies whether the nonce matches the one it sent. This ensures that the quote is current and prevents any attempt by an attacker to retransmit a stale quote.
Validation by simple PCR snapshot

Now that you can get a signed quote from the client, you can see how this helps in validating your system. We said that the quote contains the PCR digest list—the current values of all the TPM’s PCRs—and that these values were created by extending a series of measurements to a PCR, where the measurements are secure hashes of all your important firmware components, plus any configuration data that governs how the system boots. So, the PCR digest list is essentially a record of exactly what executed when the system booted.

As mentioned previously, if firmware extends the same set of measurements to a PCR in the same sequence, the resulting PCR digest will be the same. And that’s exactly what tends to happen when a UEFI or OpenPOWER server boots. Recall the description of how PCRs [0-7] are used. If the system boots the same set of firmware components and the same boot-time configuration, the values of PCRs [0-1] and PCRs [2-3] will be identical, from one boot to the next. And if the system goes on to boot the same Target OS from the same media, the values of PCRs [4-5] should be identical as well.

This provides a simple way to check system integrity. You can ask the TPM for a quote at any time and it will return the PCR values for the current boot. If the values match those produced on some previous boot, you can be certain that the system booted exactly the same way it did before. If you have a snapshot of the PCR values from the system in a known-good state (that is, when you know it booted only the expected firmware and configuration) you can designate that snapshot as a reference or golden state, which you can use to check system integrity in the future.
“With a simple PCR snapshot you can verify your firmware integrity by comparing to a reference or golden state.”

A drawback of this approach is that if a PCR value does not match with the reference state, it is difficult to know exactly what changed. It might be the loading of an unauthorized component or it might be a simple configuration change that has no impact on the system security.

Also, this approach requires the reference state to be recorded in advance. The snapshot must be recorded when the system is in a known-good state and you know it has not been compromised. When you update any component or change the system's boot configuration, you must record a new snapshot: again, in a state where you are sure it has not been compromised. To do this, you must first verify whether the system matches the previous state, make your changes, and record a new snapshot. And you must do this in an atomic fashion where you can be sure that nothing unexpected has changed in the interim. This might require taking the system off the production network to perform the operation.

This recording of new reference states must be performed throughout the system lifecycle. And though it may be easy to capture the initial reference state when the system is first deployed, what about when the system has already been in production for some time? Can you perform any validation check at that point? Or, can that only be done when a reference state was recorded at the start of the system lifecycle? To answer these questions you need to examine the firmware event log retrieved with the quote. We would cover this topic and more in a future article.

**Conclusion**

Using Trusted Boot on IBM OpenPOWER helps you verify the integrity and authenticity of your firmware and ensure that your server has booted securely. Together with remote attestation, you can confirm that your server is running only authorized firmware from IBM or another trusted platform vendor of choice. In this article, you have seen how the Core Root of Trust is established, how integrity measurements are recorded to the Trusted Platform Module and how you can use the IBM TPM 2.0 Attestation Client Server to verify those measurements. By comparing the measurements to those recorded on a previous boot you can determine if your server booted to a known-good, reference state.

In a future article we might examine secure boot in more detail, and explore the value of the boot-time measurement list.

**Resources**

- **Attestation Protocols** (Ken Goldman, September 26, 2016); Learn more about the operation of the IBM TPM 2.0 Attestation Client Server.
- **The OpenPOWER Firmware Build Environment**; Download the source code and build your own firmware stack for your IBM OpenPOWER server, and follow the development of Secure and Trusted Boot for OpenPOWER.
- **Secure and Trusted Boot for OpenPOWER** (OpenPOWER Summit 2016); Learn in more detail how the Core Root of Trust is established during boot on IBM POWER8 systems.
• **CAPI accelerated GZIP Compression Adapter User's guide**; Learn more about the IBM Coherent Accelerator Processor Interface (CAPI) for IBM POWER with this guide to using CAPI for file compression.

• **The Linux on Power Community**; Join the community of users running Linux and Linux applications on IBM POWER Systems.

• **The OpenPOWER Foundation**; Follow the latest developments from IBM and other participating members of the OpenPOWER Foundation.

**Acknowledgment**

The authors would like to thank the following folks from IBM for helpful discussion and technical review: Warren Grunbok, Nayna Jain, Elaine Palmer, George Wilson, and Mimi Zohar.

© Copyright IBM Corporation 2017
**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)