Contents

1 Introduction .......................................................................................................................... 7

2 Audience ............................................................................................................................. 8

2.1 Intel® Trusted Execution Technology (Intel® TXT) and Intel® BootGuard (Intel® BtG) .................. 8

2.2 Supported Root of Trust Configurations ......................................................................... 8

2.3 Application Integrity ......................................................................................................... 9

2.4 Typical POC Deployment Architectures ........................................................................... 9

3 Use Case 1a: Platform Attestation (Linux*) ................................................................. 10

3.1 Scope .......................................................................................................................... 10

3.2 Requirements ................................................................................................................ 10

3.2.1 Hardware ............................................................................................................... 10

3.2.2 Software ................................................................................................................ 11

3.2.3 Operating System .................................................................................................. 11

3.2.4 Networking ............................................................................................................. 11

3.3 Installation .................................................................................................................. 11

3.3.1 Installing the Verification Service ..................................................................... 11

3.3.2 Installing the Trust Agent for Linux .............................................................. 13

3.4 Demonstrating the Platform Attestation Use Case ......................................................... 14

3.4.1 Verify Services ..................................................................................................... 14

3.4.2 Register Hosts ........................................................................................................ 15

3.4.3 List Hosts ............................................................................................................. 15

3.4.4 Retrieve Reports (Untrusted – No Flavors) ......................................................... 15

3.4.5 Import Flavors ...................................................................................................... 15

3.4.6 Import the default SOFTWARE Flavor .................................................................. 16

3.4.7 Retrieve Reports (Trusted) .................................................................................. 16

3.4.8 Demonstrate Untrusted State ............................................................................. 16

3.4.9 Returning the hosts to a Trusted state ................................................................. 19

4 Use Case 1b: Platform Attestation (Windows*) .......................................................... 21

4.1 Scope ........................................................................................................................ 21

4.2 Requirements ................................................................................................................ 21

4.2.1 Hardware ............................................................................................................. 21

4.2.2 Software .............................................................................................................. 22

4.2.3 Operating System ................................................................................................. 22

4.2.4 Networking .......................................................................................................... 22

4.3 Installation .................................................................................................................. 22

4.3.1 Installing the Verification Service ..................................................................... 22

4.3.2 Installing the Trust Agent for Windows .............................................................. 24

4.4 Demonstrating the Platform Attestation Use Case ......................................................... 25

4.4.1 Verify Services ..................................................................................................... 25

4.4.2 Register Hosts ........................................................................................................ 25

4.4.3 List Hosts ............................................................................................................. 25

4.4.4 Retrieve Reports (Untrusted – No Flavors) ......................................................... 25

4.4.5 Import Flavors ...................................................................................................... 26

4.4.6 Retrieve Reports (Trusted) .................................................................................. 26
5 Use Case 1c: Platform Attestation (VMware*) ........................................... 30
  5.1 Scope ........................................................................................................... 30
  5.2 Requirements ............................................................................................ 30
    5.2.1 Hardware ........................................................................................... 30
    5.2.2 Software ............................................................................................... 31
    5.2.3 Operating System ............................................................................... 31
    5.2.4 Installing the Verification Service ...................................................... 31
  5.3 Demonstrating the Platform Attestation Use Case ................................... 33
    5.3.1 Verify Services...................................................................................... 33
    5.3.2 Register Hosts ..................................................................................... 33
    5.3.3 List Hosts ............................................................................................. 33
    5.3.4 Retrieve Reports (Untrusted – No Flavors) ......................................... 33
    5.3.5 Import Flavors ...................................................................................... 34
    5.3.6 Retrieve Reports (Trusted) .................................................................... 34
    5.3.7 Demonstrate Untrusted State .............................................................. 34
    5.3.8 Returning the hosts to a Trusted state .................................................. 37

6 Use Case 2a: Asset Tags (Linux) ................................................................. 39
  6.1 Scope ........................................................................................................... 39
  6.2 Requirements ............................................................................................ 39
  6.3 Demonstrating the Asset Tag Use Case for Linux ................................... 39
    6.3.1 Verify that hosts are registered and Trusted ....................................... 39
    6.3.2 Create Asset Tag Certificates for each host ....................................... 39
    6.3.3 Deploy Asset Tags ............................................................................... 40
    6.3.4 Create new Attestation Reports for each host .................................... 41

7 Use Case 2b: Asset Tags (Windows) ............................................................ 42
  7.1 Scope ........................................................................................................... 42
  7.2 Requirements ............................................................................................ 42
  7.3 Demonstrating the Asset Tag Use Case for Windows ................................ 42
    7.3.1 Verify that hosts are registered and Trusted ....................................... 42
    7.3.2 Create Asset Tag Certificates for each host ....................................... 42
    7.3.3 Deploy Asset Tags ............................................................................... 43
    7.3.4 Create new Attestation Reports for each host .................................... 44

8 Use Case 2c: Asset Tags (VMware) ............................................................... 45
  8.1 Scope ........................................................................................................... 45
  8.2 Requirements ............................................................................................ 45
  8.3 Demonstrating the Asset Tag Use Case for VMware .................................. 45
    8.3.1 Verify that hosts are registered and Trusted ....................................... 45
    8.3.2 Create Asset Tag Certificates for each host ....................................... 45
    8.3.3 Deploy Asset Tags ............................................................................... 46
    8.3.4 Create Asset Tag Flavor ....................................................................... 47
    8.3.5 Create new Attestation Reports for each host .................................... 47

9 Use Case 3: Application Integrity ................................................................. 48
  9.1 Scope ........................................................................................................... 48
  9.2 Requirements ............................................................................................ 48
  9.3 Demonstrating the Application integrity Use Case for Linux .................... 48
9.3.1 Verify that hosts are registered and Trusted ........................................48
9.3.2 Create a New SOFTWARE Flavor ..........................................................48
9.3.3 Deploy the Application Manifest ..........................................................49
9.3.4 Create new Attestation Reports for each host ......................................49

10  Use Case 4: OpenStack Orchestration* .....................................................51

10.1 Scope ........................................................................................................51
10.2 Requirements ..........................................................................................51
    10.2.1 Hardware .........................................................................................51
    10.2.2 Software .........................................................................................52
    10.2.3 Operating System ............................................................................52
    10.2.4 Networking .....................................................................................52
    10.2.5 OpenStack .......................................................................................52
    10.2.6 Installation .......................................................................................53
10.3 Demonstrating the OpenStack Orchestration* Use Case ......................54
    10.3.1 Configuring a Tenant in the Integration Hub ..................................54
    10.3.2 Setting Launch Requirements in Image Traits ...............................55
    10.3.3 Launching an Instance on a Compliant Node ..................................56
    10.3.4 Launching an Instance where No Nodes are Compliant ..............57

11  Use Case 5: Kubernetes Orchestration .......................................................58

11.1 Scope ........................................................................................................58
11.2 Requirements ..........................................................................................59
    11.2.1 Hardware .........................................................................................59
    11.2.2 Operating System ............................................................................59
    11.2.3 Networking .....................................................................................59
    11.2.4 Kubernetes ......................................................................................59
    11.2.5 Installation .......................................................................................60
11.3 Demonstrating the Kubernetes Scheduling Use Case .........................62
    11.3.1 Configuring a Tenant in the Integration Hub .................................62
    11.3.2 Configuring Pods to Require Intel® SecL Attributes ......................64
    11.3.3 Launching a Pod on a Compliant Worker Node .............................67
    11.3.4 Launching a Pod where No Worker Nodes are Compliant ...........67
## Revision History

<table>
<thead>
<tr>
<th>Revision Number</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>• Initial release of the document.</td>
<td>Mar 2019</td>
</tr>
<tr>
<td>1.5</td>
<td>Updated as per Intel SecL-DC v1.5</td>
<td>Aug 2019</td>
</tr>
</tbody>
</table>
1 Introduction

This document defines the key prerequisites, requirements, scope and process for enabling Intel® Security Libraries for Data Center (Intel® SecL - DC) use cases in a Proof of Concept setting.
2 Audience

This document is intended for Systems Engineers who intend to complete a POC for one or more use cases for Intel® SecL – DC.

When using this document, be sure to read the sections for each use case you intend to enable as part of the POC to ensure that all requirements are met. Many use cases build upon prerequisite use cases, but add additional requirements that must be considered before starting a deployment.

2.1 Intel® Trusted Execution Technology (Intel® TXT) and Intel® BootGuard (Intel® BtG)

Intel® SecL – DC requires a server platform with support for Intel® Trusted Execution Technology (Intel® TXT), and/or Intel® BootGuard (Intel® BtG).

Generally any server platform after the Grantley generation will have support for Intel® TXT. The platform must have full firmware support for Intel® TXT including a valid, production-worthy SINIT ACM module, and the feature must be turned on in the system BIOS.

Platforms of the Purley-Refresh generation and later may have the Intel® BootGuard feature enabled. This security feature is enabled in platform fuses by the server OEM/ODM, and is not configurable after manufacture. Intel® BtG supports several “profiles” that define whether the feature will perform signature verification, or whether it will perform signature verification and also extend measurements to the platform TPM. Consult with your server manufacturer to determine whether your particular platform supports Intel® BtG and whether it is in a measurement profile.

Both of these security features can be enabled on the same platform.

2.2 Supported Root of Trust Configurations

Intel® SecL-DC supports several different combinations for Intel® TXT, Intel® BootGuard, and UEFI SecureBoot. Note that Intel® TXT uses “tboot” to initiate the Dynamic Root of Trust measurements on Linux platforms, and “tboot” is currently incompatible with UEFI SecureBoot. The following combinations are supported. All instances of Intel® BootGuard assume that a measurement profile is in use.

Intel® TXT + tboot

Intel® TXT + UEFI SecureBoot

Intel® TXT + Intel® BtG + tboot

Intel® TXT + Intel® BtG + UEFI SecureBoot

Intel® BtG + UEFI SecureBoot
2.3 Application Integrity

Added in the Intel® SecL-DC 1.5 release, Application Integrity allows any files and folders on a Linux host system to be included in the Chain of Trust integrity measurements. These measurements are attested by the Verification Service along with the other platform measurements, and are included in determining the host’s overall Trust status. The measurements are performed by a measurement agent called tbootXM, which is built into initrd during Trust Agent installation. Because initrd is included in other Trusted Computing measurements, this allows Intel® SecL-DC to carry the Chain of Trust all the way to the Linux filesystem.

2.4 Typical POC Deployment Architectures

For simplicity, Intel® SecL – DC POC deployments should include one to three physical server platforms with TPMs installed and Intel® TXT activated, as well as one additional server (physical or virtual) for deployment of the services other than the Trust Agent.

This includes (as applicable – the list differs based on the use cases being enabled in the POC) the Intel® SecL – DC Verification Service, Integration Hub, Trust Agent, and any applicable OpenStack* services (Nova*, Glance*, Horizon*, Keystone*, and so forth).
3 Use Case 1a: Platform Attestation (Linux*)

The Platform Attestation use case will enable external platform integrity verification using TCG-standard practices. This feature is the foundation for other Intel® SecL – DC use cases.

Platform Attestation involves taking measurements of system components during system boot and then cryptographically verifying that the actual measurements taken match a set of expected or approved values, ensuring that the measured components were in an acceptable or “trusted” state at the time of the last system boot.

Intel® SecL – DC leverages the Trusted Compute Group specification for a trusted boot process, extending measurements of platform components to registers in a Trusted Platform Module (TPM) and securely generating quotes of those measurements from the TPM for remote comparison to expected values (attestation).

The deployment will include the Verification Service and one or more Trust Agents. The Verification Service is typically installed on a Virtual Machine, while the Trust Agent must be installed on each physical server.

3.1 Scope

This use case will include the installation of the Verification Service and one or more Trust Agent compute nodes. After installation, the Platform Attestation use case will be demonstrated by registering the Trust Agent hosts, importing Flavors from the registered hosts, retrieving Attestation Reports for the registered hosts and performing a negative test to demonstrate an “Untrusted” result.

3.2 Requirements

3.2.1 Hardware

- One or more server platforms with in a supported Root of Trust configuration.
- Each physical server must have a physical or firmware Trusted Platform Module version 2.0 installed and active in the system BIOS. The TPM must have “cleared” ownership (the TPM ownership can be cleared in the system BIOS).
- One server (which may be physical or virtual) dedicated for the Intel® SecL – DC Verification Service. This server does not require any Root of Trust configuration, but does require network access to the other POC servers.
3.2.2 Software

- (Optional, required for database modification for optional Untrusted state demonstration) A GUI-based remote database client that supports the PostgreSQL JDBC* driver (for example, SQL Workbench*) installed on a laptop or other system outside of the POC environment, but with network connectivity.
- cURL* or a REST* API utility like Postman* to execute API requests.

3.2.3 Operating System

- The Intel® SecL – DC Verification Service supports Red Hat Enterprise* Linux (RHEL*) 7.4 and above.
- The Intel® SecL – DC Trust Agent for Linux supports RHEL 7.4 and above.

3.2.4 Networking

- The Intel® SecL – DC Verification Service must be able to reach each physical server over the network.
- The Intel® SecL – DC installers will require access to package repositories; either internet access or access to suitable repository mirrors must be provided (as well as any applicable subscriptions to access the repositories).
- Hostname resolution is not necessary but is very helpful. IP addresses may be used in place of hostnames. Ensure that IP addresses and hostnames are resolvable from all Trust Agent hosts to the Verification Service and vice versa. Be consistent – use either all IP addresses, or all hostnames, do not mix-and-match.

3.3 Installation

The Platform Attestation use case requires two Intel® SecL – DC services: the Verification Service and the Trust Agent.

3.3.1 Installing the Verification Service

This section details how to install the Intel® SecL – DC services.

3.3.1.1 Package Dependencies

The Intel® SecL – DC Verification Service requires the following packages and their dependencies:
- Logback* (optional)
- Java* 8 JDK
- OpenSSL*
- Postgres* client and server 9.3 (server component optional if an external Postgres database is used)
- Unzip*
- Tboot (if UEFI SecureBoot is not enabled)

If they are not already installed, the Verification Service installer will attempt to install these automatically using the package manager. Automatic installation requires access to package repositories (the RHEL subscription repositories, the Extra Packages for Enterprise Linux* or EPEL* repository, or a suitable mirror), which may require an Internet connection.

If the packages are to be installed from the package repository, be sure to update the repository package lists before installation.

### 3.3.1.2 Supported Operating Systems

The Intel® SecL – DC Verification Service supports RHEL 7.4 and above.

### 3.3.1.3 Recommended Hardware

- 4 vCPUs
- RAM: 8 GB
- 100 GB
- One network interface with network access to all managed servers

### 3.3.1.4 Installation

To install the Verification Service, follow these steps:

1. Copy the Verification Service installation binary to the /root/ directory.
2. Create the mtwilson.env installation answer file:

```
### User credentials
export MC_FIRST_USERNAME=administrator
export MC_FIRST_PASSWORD=password

### Database Configuration
export DATABASE_USERNAME=root
export DATABASE_PASSWORD=password

### Service IP or Hostname definition
export MTWILSON_SERVER=<Verification Service IP address or Hostname>
```

3. Execute the installer binary.

When the installation completes, the Verification Service will be available. The services can be verified by running a mtwilson status from the Verification Service command line.

```bash
# mtwilson status
```
3.3.2 Installing the Trust Agent for Linux

The Intel® SecL – DC Trust Agent must be installed on each physical server that will be registered and attested. Each server must be in a supported Root of Trust configuration.

3.3.2.1 Package Dependencies

The Trust Agent requires the following packages and their dependencies:

- Tboot (if UEFI SecureBoot is not enabled)
- tpm-quote-tools
- tpm2-tools
- openssl

If they are not already installed, the Trust Agent installer will attempt to install these automatically using the package manager. Automatic installation requires access to package repositories (the RHEL subscription repositories, the EPEL repository, or a suitable mirror), which may require an Internet connection.

If the packages are to be installed from the package repository, be sure to update the repository package lists before installation.

3.3.2.2 Supported Operating Systems

The Intel® SecL – DC Trust Agent for Linux supports RHEL 7.4 and above.

3.3.2.3 Prerequisites

The following must be completed before installing the Trust Agent:

- Supported server hardware including an Intel® Xeon® processor in a supported Root of Trust configuration.
- TPM 2.0 installed and activated in the system BIOS, with cleared ownership status
- (required if UEFI SecureBoot is not enabled; not available as an option if UEFI SecureBoot is enabled) System must be booted to a tboot boot option (Trust Agent installation will automatically install tboot if not present, and then require a reboot before proceeding)
- (Provisioning step only) Intel® SecL – DC Verification Service server installed and active

3.3.2.4 Installation

To install the Trust Agent for Linux:

1. Create the trustagent.env answer file in the /root/ directory:
MTWILSON_API_URL=https://<Verification Service IP or Hostname>:8443/mtwilson/v2
MTWILSON_TLS_CERT_SHA384=<SHA384 of Verification Service TLS Certificate>
MTWILSON_API_USERNAME=admin
MTWILSON_API_PASSWORD=password
REGISTER_TPM_PASSWORD=y
TRUSTAGENT_LOGIN_REGISTER=true
PROVISION_ATTESTATION=y
CURRENT_IP=<Trust Agent IP address>
TRUSTAGENT_ADMIN_USERNAME=tagentadmin
TRUSTAGENT_ADMIN_PASSWORD=password

Note: The MTWILSON_API_USERNAME and password are required by the Trust Agent and can be satisfied by the PRIVACYCA_DOWNLOAD_USERNAME user created during the installation of the Verification Service. The MTWILSON_TLS_CERT_SHA384 value can be retrieved from the Verification Service using the following command from the Verification Service command line:
cat /opt/mtwilson/configuration/https.properties

2. Copy the Trust Agent installer binary to the /root/ directory.

3. Execute the Trust Agent installer, and wait for the installation to complete.
   — If UEFI SecureBoot is not enabled, the Trust Agent will install tboot and other prerequisites if not already present.
   — If tboot is installed by the Trust Agent installer, the installation will abort and reboot the host. This is because the Trust Agent requires the host to be booted into a tboot boot option, which populates the OS-level measurements in the host TPM.
   — After the host reboots, re-run the Trust Agent installation binary to resume the installation.

4. After installation is complete, reboot the server. This will allow the Application Integrity measurements to be performed and extended to the TPM.

3.4 Demonstrating the Platform Attestation Use Case

3.4.1 Verify Services

The Version API can be used to confirm that all services are up and running before proceeding to the actual use case demonstration.

GET https://verification.server.com:8443/mtwilson/v2/version
GET https://trustagent.server.com:1443/v2/version
3.4.2 Register Hosts

Registration creates a host record with connectivity details in the Verification Service database. This host record will be used by the Verification Service to retrieve TPM attestation quotes from the Trust Agent to generate an attestation report.

Each Trust Agent host will need to be registered with a separate call.

POST https://verification.service.com:8443/mtwilson/v2/hosts
{
    "host_name": "<hostname of host to be registered>",
    "tls_policy_id" : "TRUST_FIRST_CERTIFICATE",
    "connection_string": 
        "https://trustagent.server.com:1443;u=tagentadmin;p=password",
    "flavorgroup_name" : ",",
    "description" : "<description>",
}

3.4.3 List Hosts

After registration, the /hosts API can be used to list all registered hosts and confirm that the registrations were all successful.


3.4.4 Retrieve Reports (Untrusted – No Flavors)

You can now retrieve the latest Attestation Report for each host to demonstrate the “faults” shown when match Flavors do not exist in the Verification Service database. Currently no Flavors should be present, so all of these Reports will show an Untrusted status, with Faults showing that matching Flavors were required but not found.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json

3.4.5 Import Flavors

Next you will actually import Flavors. For simplicity, you can import all three Flavor parts (PLATFORM, OS, and HOST_UNIQUE) from each Trust Agent host.

Technically, only the HOST_UNIQUE part must come from each host; the PLATFORM and OS Flavors can be created just once per version (for example, if all of our Trust Agent hosts use BIOS version 1.23, we only need to import the PLATFORM Flavor for BIOS version 1.23 once, and all other hosts using the same BIOS version will be matched to the same Flavor).

POST https://verification.server.com:8443/mtwilson/v2/flavors
{
}
3.4.6 Import the default SOFTWARE Flavor

The default SOFTWARE Flavor part includes measurements for all of the Linux Trust Agent static files and folders, and should be created separately from the other Flavor parts. Only one default SOFTWARE Flavor needs to be created for each version of the Trust Agent. If the SOFTWARE Flavor for the same Trust Agent version is imported multiple times, subsequent imports will fail as the Flavor already exists.

To import the SOFTWARE Flavor part from a host:

```json
POST https://verification.server.com:8443/mtwilson/v2/flavors
{
    "connection_string": "https://trustagent.server.com:1443;u=tagentadmin;p=password",
    "partial_flavor_types": ["SOFTWARE"],
    "flavorgroup_name": "",
    "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}
```

3.4.7 Retrieve Reports (Trusted)

Now that all Flavors exist and all hosts have been registered, you can retrieve new Attestation Reports. New Reports are automatically generated whenever a host is matched to a new Flavor, which happened when we imported our Flavors.

This is the same request used previously, but this time all of the Reports should show that the “Overall” trust is “True,” and we should no longer see any Faults.

GET
https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true
Headers:
Accept: application/json

3.4.8 Demonstrate Untrusted State

Finally, it is possible to demonstrate what an Untrusted attestation looks like and what can cause it. There are a number of ways to force a host to attest as Untrusted.
In production, a host can become Untrusted if the BIOS or OS kernel are upgraded/downgraded to use a version where no Flavor exists, if a malicious attacker has inserted malicious code into a measured component, or so forth.

For testing purposes, the easiest ways to force a host to become Untrusted are to either boot the host to a non-tboot boot option (simply reboot, and at the boot menu option when RHEL boots, select the option without tboot).

Attestation Reports are valid for 24 hours by default (this is configurable); you need to force a new Report to be generated now, after the previous change:

POST https://verification.server.com:8443/mtwilson/v2/reports

Headers:
Accept: application/json
input: {"host_name":"<Trust Agent host>"}

The new Report will show that the BIOS Flavor part is still Trusted, but that the HOST_UNIQUE and OS Flavor Parts have mismatched measurements. This is because the platform performs the BIOS-related measurements before the OS boots, but tboot performs the other measurements.

Because the system booted without tboot, the TPM memory registers that store the tboot measurements will all be 0’s or F’s, and will mismatch the expected measurements in the Flavors.

3.4.8.1 (Optional) Database Modification

While slightly more complicated, it is also possible to edit one or more Flavors in the database and change the expected measurement results. Because of the size of the Flavor objects in the database, this is best performed with a remote GUI database client like SQL Workbench, as opposed to using the command line. This step is optional, and will show a different Fault more consistent with a platform that has been tampered with by a malicious attack.

First you need to enable remote access to the Verification Service database, which is disabled by default.

In the /var/lib/pgsql/9.3/data/postgresql.conf file, set the following:

listen_addresses='*

Next, edit /var/lib/pgsql/9.3/data/pg_hba.conf and change the last line to the following:

host all all 127.0.0.1/0 password

Finally, restart the Postgresql database:

systemctl restart postgresql-9.3

Now you should be able to access the database using a remote database client, like SQL Workbench. By default, the database name is mw_as. The username and password were configured in the mtwilson.env answer file during the installation; the example used “root” and “password”.
After connecting to the mw_as database, look at the mw_flavor table. You should see a number of Flavors, including PLATFORM, OS, and HOST_UNIQUE Flavors from when they were imported in an earlier step.

In the "content" column is the actual Flavor object. Find a "PLATFORM" Flavor:

Find the “pcr_0” measurements (depending on the TPM version and your platform OEM, you may see only SHA1, only SHA256, or both measurements).

```
{"SHA1":{"pcr_0":{"value":"d2ed125942726641a7260c4f92beb67d531a0def"}}
"SHA256":{"pcr_0":{"value":"db83f0e8a1773c21164c17986037cdf8afcbdbclb815772c6da1b7f8a3"}}
```

Change a digit in each of these entries. Do not change the length, simply change the last digit of each “pcr_0” hash to a different hexadecimal digit.

Save the changes to the database.

Now force the creation of a new Report.

```
POST https://verification.server.com:8443/mtwilson/v2/reports
{
  "host_name": "<Trust Agent hostname>"
}
```

Do this for each host in the POC environment. When you get the Report for the host that was using the modified PLATFORM flavor, the OS and HOST_UNIQUE attestations will remain Trusted.

The PLATFORM Flavor part, however, will show that it is Untrusted with a “PcrValueMismatch” fault indicating that the actual measurement for PCR 0 does not match the expected value.

Because PCR 0 includes the hash of the core system BIOS, this replicates an event where the BIOS has been maliciously modified.
<markers>PLATFORM</markers>

<expectedPcr
digest_type="com.intel.mtwilson.core.common.model.PcrSha256">
  <index>pcr_0</index>
  <value>187c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9</value>
  <pcrBank>SHA256</pcrBank>
</expectedPcr>

<faults>
  <faults
  fault_name="com.intel.mtwilson.core.verifier.policy.fault.PcrValueMismatchSha256">
    <description>Host PCR 0 with value 987c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9 does not match expected value 187c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9</description>
    <pcrIndex>pcr_0</pcrIndex>
    <expectedValue>187c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9</expectedValue>
    <actualValue>987c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9</actualValue>
  </faults>
</faults>

3.4.9 Returning the hosts to a Trusted state

Now that you demonstrated an Untrusted state, you need to recover the hosts back to a Trusted state.

The easiest way to do this is to simply import new Flavors for each host:

POST https://verification.server.com:8443/mtwilson/v2/flavors
{
  "connection_string": "https://trustagent.server.com:1443;u=tagentadmin;p=password",
  "partial_flavor_types": ["PLATFORM", "OS", "HOST_UNIQUE"],
  "flavorgroup_name": "",
  "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}

Repeat this call for each registered host to import new Flavors. This will not overwrite the existing Flavors, but will add new Flavors imported directly from each host.

When new Flavors are imported, the Verification Service will automatically try to see which hosts (if any) should be matched to those Flavors. Because the new Flavors will actually match the PCR values seen on the host TPM, the
Verification Service will match the new Flavors instead of the Flavors that were modified with bad values.

After new Flavors have been imported from each host, retrieve the Reports for each host again to verify that each host is Trusted:

GET

https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json

The “Overall” value should be “true” for each host.
4 Use Case 1b: Platform Attestation (Windows*)

The Platform Attestation use case will enable external platform integrity verification using TCG-standard practices. This feature is the foundation for other Intel® SecL – DC use cases.

Platform Attestation involves taking measurements of system components during system boot, and then cryptographically verifying that the actual measurements taken match a set of expected or approved values, ensuring that the measured components were in an acceptable or “trusted” state at the time of the last system boot.

Intel® SecL – DC leverages the Trusted Compute Group specification for a trusted boot process, extending measurements of platform components to registers in a TPM, and securely generating quotes of those measurements from the TPM for remote comparison to expected values (attestation).

Deployment will include the Verification Service and one or more Trust Agents. The Verification Service is typically installed on a Virtual Machine, while the Trust Agent must be installed on each physical server.

4.1 Scope

This use case will include the installation of the Verification Service and one or more Trust Agent compute nodes. After installation, the Platform Attestation use case will be demonstrated by registering the Trust Agent hosts; importing Flavors from the registered hosts; retrieving Attestation Reports for the registered hosts; and performing a negative test to demonstrate an “Untrusted” result.

4.2 Requirements

4.2.1 Hardware

- One or more server platforms with a supported Root of Trust configuration.
- Each physical server must have a physical or firmware Trusted Platform Module installed and active in the system BIOS. Both TPM versions 1.2 and 2.0 are supported for Windows. The TPM must have “cleared” ownership (the TPM ownership can be cleared in the system BIOS) and must be “Enabled” in the system BIOS (TPM 1.2 only; TPM 2.0 is always “Enabled”).
- One server (which may be physical or virtual) dedicated for the Intel® SecL – DC Verification Service. This server does not require any Root of Trust configuration, but does require network access to the other POC servers.
4.2.2 Software
- (Optional, required for database modification for optional Untrusted state demonstration) A GUI-based remote database client that supports the PostgreSQL JDBC driver (for example, SQL Workbench) installed on a laptop or other system outside of the POC environment, but with network connectivity.
- cURL or a REST API utility like Postman to execute API requests

4.2.3 Operating System
- The Intel® SecL – DC Verification Service supports RHEL 7.4 and above.
- The Intel® SecL – DC Trust Agent for Windows supports Microsoft Windows Server* 2016 and above.

4.2.4 Networking
- The Intel® SecL – DC Verification Service must be able to reach each physical server over the network.
- The Intel® SecL – DC installers will require access to package repositories; either internet access or access to suitable repository mirrors must be provided (as well as any applicable subscriptions to access the repositories).
- Hostname resolution is not necessary but is very helpful. IP addresses may be used in place of hostnames. Ensure that IP addresses and hostnames are resolvable from all Trust Agent hosts to the Verification Service and vice versa. Be consistent – use either all IP addresses, or all hostnames, do not mix-and-match.

4.3 Installation
The Platform Attestation use case requires two Intel® SecL – DC services: the Verification Service and the Trust Agent.

4.3.1 Installing the Verification Service
This section details how to install the Intel® SecL – DC services.

4.3.1.1 Package Dependencies
The Intel® SecL – DC Verification Service requires the following packages and their dependencies:
- Logback* (optional)
- Java 8 JDK
- OpenSSL
Postgres* client and server 9.3 (server component optional if an external Postgres database is used)

Unzip

If they are not already installed, the Verification Service installer will attempt to install these automatically using the package manager.

Automatic installation requires access to package repositories (the RHEL subscription repositories, the EPEL repository, or a suitable mirror), which may require an Internet connection. If the packages are to be installed from the package repository, be sure to update the repository package lists before installation.

### 4.3.1.2 Supported Operating Systems

The Intel® SecL – DC Verification Service supports RHEL 7.4 and above.

### 4.3.1.3 Recommended Hardware

- 4 vCPUs
- RAM: 8 GB
- 100 GB
- One network interface with network access to all managed servers

### 4.3.1.4 Installation

To install the Verification Service, follow these steps:

1. Copy the Verification Service installation binary to the /root/ directory.
2. Create the mtwilson.env installation answer file:

   ```
   ### User credentials
   export MC_FIRST_USERNAME=administrator
   export MC_FIRST_PASSWORD=password
   
   ### Database Configuration
   export DATABASE_USERNAME=root
   export DATABASE_PASSWORD=password
   
   ### Service IP or Hostname definition
   export MTWILSON_SERVER=<Verification Service IP address or Hostname>
   ```
3. Execute the installer binary.

When the installation completes, the Verification Service will be available. The services can be verified by running a `mtwilson status` from the Verification Service command line.

# mtwilson status
Host Verification Service is running
4.3.2 Installing the Trust Agent for Windows

The Intel® SecL – DC Trust Agent must be installed on each physical TPM/Intel® TXT-enabled server that will be registered and attested.

4.3.2.1 Supported Operating Systems

The Intel® SecL – DC Trust Agent for Windows supports Microsoft Windows Server 2016 and above.

4.3.2.2 Prerequisites

The following must be completed before installing the Trust Agent:

- Supported server hardware including an Intel® Xeon® processor with a supported Root of Trust configuration.
- TPM (version 1.2 or 2.0) installed and activated in the system BIOS, with cleared ownership status.
- Intel® SecL – DC Verification Service server installed and active.

4.3.2.3 Installation

To install the Trust Agent for Windows:

1. Create the `trustagent.ini` answer file in the C:\Temp directory.

   
   ```
   [TRUST_AGENT]
   MTWILSON_API_URL=https://<Verification Service IP or Hostname>:8443/mtwilson/v2
   MTWILSON_TLS_CERT_SHA384=<SHA384 of Verification Service TLS Certificate>
   MTWILSON_API_USERNAME=administrator
   MTWILSON_API_PASSWORD=password
   REGISTER_TPM_PASSWORD=y
   TRUSTAGENT_LOGIN_REGISTER=true
   PROVISION_ATTESTATION=y
   CURRENT_IP=<Trust Agent IP address>
   ```

   The `MTWILSON_TLS_CERT_SHA384` value can be retrieved from the Verification Service using the following command from the Verification Service command line:

   ```
cat /opt/mtwilson/configuration/https.properties
   ```

2. Copy the Trust Agent installer executable to the C:\Temp directory.

3. Execute the Trust Agent installer, and wait for the installation to complete.
4.4 Demonstrating the Platform Attestation Use Case

4.4.1 Verify Services

The Version API can be used to confirm that all services are up and running, before proceeding to the actual use case demonstration.

GET https://verification.server.com:8443/mtwilson/v2/version
GET https://trustagent.server.com:1443/v2/version

4.4.2 Register Hosts

Registration creates a host record with connectivity details in the Verification Service database. This host record will be used by the Verification Service to retrieve TPM attestation quotes from the Trust Agent to generate an attestation report.

Each Trust Agent host will need to be registered with a separate call.

POST https://verification.service.com:8443/mtwilson/v2/hosts
{
   "host_name": "<hostname of host to be registered>",
   "tls_policy_id": "TRUST_FIRST_CERTIFICATE",
   "connection_string": "https://trustagent.server.com:1443;u=tagentadmin;p=password",
   "flavorgroup_name": "",
   "description": "<description>"
}

4.4.3 List Hosts

After registration, the /hosts API can be used to list all registered hosts and confirm that the registrations were all successful.


4.4.4 Retrieve Reports (Untrusted – No Flavors)

Now it is possible to retrieve the latest Attestation Report for each host to demonstrate the “faults” shown when match Flavors do not exist in the Verification Service database. Currently no Flavors should be present, so all of these Reports will show an Untrusted status, with Faults showing that matching Flavors were required but not found.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true
Headers:
Accept: application/json
4.4.5  Import Flavors

Next you will actually import Flavors. For simplicity, you can import all three Flavor parts (PLATFORM, OS, and HOST_UNIQUE) from each Trust Agent host.

Technically, only the HOST_UNIQUE part must come from each host; the PLATFORM and OS Flavors can be created just once per version (for example, if all of our Trust Agent hosts use BIOS version 1.23, we only need to import the PLATFORM Flavor for BIOS version 1.23 once, and all other hosts using the same BIOS version will be matched to the same Flavor).

POST https://verification.server.com:8443/mtwilson/v2/flavors
{
   "connection_string": "https://trustagent.server.com:1443;u=tagentadmin;p=password",
   "partial_flavor_types": ["PLATFORM", "OS", "HOST_UNIQUE"],
   "flavorgroup_name": "",
   "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}

4.4.6  Retrieve Reports (Trusted)

Now that all Flavors exist and all hosts have been registered, retrieve new Attestation Reports. New Reports are automatically generated whenever a host is matched to a new Flavor, which happened when we imported our Flavors.

This is the same request used previously, but this time all of the Reports should show that the “Overall” trust is “True”, and you should no longer see any Faults.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true
Headers:
Accept: application/json

4.4.7  Demonstrate Untrusted State

Finally, you need to demonstrate what an Untrusted attestation looks like and what can cause it. There are a number of ways to force a host to attest as Untrusted.

In production, a host can become Untrusted if the BIOS or OS kernel are upgraded/downgraded to use a version where no Flavor exists, or if a malicious attacker has inserted malicious code into a measured component, and so forth.

For testing purposes, the easiest ways to force a host to become Untrusted are to either boot the host to a non-tboot boot option (simply reboot, and at the boot menu option when RHEL boots, select the option without tboot).

Attestation Reports are valid for 24 hours by default (this is configurable); you need to force a new Report to be generated now, after the change.

POST https://verification.server.com:8443/mtwilson/v2/reports
Headers:
The new Report will show that the PLATFORM Flavor part is still Trusted, but that the HOST_UNIQUE and OS Flavor Parts have mismatched measurements. This is because the platform performs the BIOS-related measurements before the OS boots, but tboot performs the other measurements.

Because the system booted without tboot, the TPM memory registers that store the tboot measurements will all be 0’s or F’s, and will mismatch the expected measurements in the Flavors.

4.4.7.1 (Optional) Database Modification

While slightly more complicated, it is also possible to edit one or more Flavors in the database and change the expected measurement results. Because of the size of the Flavor objects in the database, this is best performed with a remote GUI database client like SQL Workbench, as opposed to using the command line. This step is optional, and will show a different Fault more consistent with a platform that has been tampered with by a malicious attack.

First you need to enable remote access to the Verification Service database, which is disabled by default.

In the /var/lib/pgsql/9.3/data/postgresql.conf file, add the following line:
listen_addresses='**'

Next, edit /var/lib/pgsql/9.3/data/pg_hba.conf and change the last line to the following:
host all all 127.0.0.1/0 password

Finally, restart the Postgresql database:
systemctl restart postgresql-9.3

Now you should be able to access the database using a remote database client, like SQL Workbench. By default, the database name is mw_as. The username and password were configured in the mtwilson.env answer file during the installation; the example used “root” and “password”.

After connecting to the mw_as database, look at the mw_flavor table. You should see a number of Flavors, including PLATFORM, OS, and HOST_UNIQUE Flavors from when they were imported in an earlier step.

In the “content” column is the actual Flavor object. Find a “PLATFORM” Flavor:
Find the “pcr_0” measurements (depending on the TPM version and your platform OEM, you may see only SHA1, only SHA256, or both measurements).

{"SHA1":{"pcr_0":{"value":"d2ed125942726641a7260c4f92beb67d531a0def"}}

"SHA256":{"pcr_0":{"value":"db83f0e8a1773c21164c17986037cdf8afclbbdc1b815772c6dalbefb1a7f8a3"}}

Change a digit in each of these entries. Do not change the length, simply change the last digit of each “pcr_0” hash to a different hexadecimal digit.

Save the changes to the database.

Now force the creation of a new Report.
POST https://verification.server.com:8443/mtwilson/v2/reports
{
   "host_name":"<Trust Agent hostname>"
}

Do this for each host in the POC environment. When you get the Report for the host that was using the modified PLATFORM flavor, the OS and HOST_UNIQUE attestations will remain Trusted.

The PLATFORM Flavor part, however, will show that it is Untrusted, with a “PcrValueMismatch” fault indicating that the actual measurement for PCR 0 does not match the expected value.

Because PCR 0 includes the hash of the core system BIOS, this replicates an event where the BIOS has been maliciously modified.

< PLATFORM >
   <trust>false</trust>
   <rules>
       <rules>
           <rule
               rule_name="com.intel.mtwilson.core.verifier.policy.rule.PcrMatchesConstant"
               >
               <markers>
                   <markers>PLATFORM </markers>
               </markers>
               <expectedPcr
                   digest_type="com.intel.mtwilson.core.common.model.PcrSha256">
                   <index>pcr_0</index>
                   <value>187c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82edca79f9</value>
               </expectedPcr>
               <expectedPcr
                   digest_type="com.intel.mtwilson.core.common.model.PcrSha256">
                   <index>pcr_0</index>
                   <value>187c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82edca79f9</value>
               </expectedPcr>
           </rule>
       </rules>
   </rules>
</PLATFORM>
4.4.8 Returning the hosts to a Trusted state

Now that you demonstrated an Untrusted state, you need to recover the hosts back to a Trusted state.

The easiest way to do this is to simply import new Flavors for each host:

```plaintext
POST https://verification.server.com:8443/mtwilson/v2/flavors
{
    "connection_string":
    "https://trustagent.server.com:1443;u=tagentadmin;p=password",
    "partial_flavor_types": ["PLATFORM", "OS", "HOST_UNIQUE"],
    "flavorgroup_name": "",
    "tls_policy_id":"TRUST_FIRST_CERTIFICATE"
}
```

Repeat this call for each registered host to import new Flavors. This will not overwrite the existing Flavors, but will add new Flavors imported directly from each host.

When new Flavors are imported, the Verification Service will automatically try to see which hosts (if any) should be matched to those Flavors. Because the new Flavors will actually match the PCR values seen on the host TPM, the Verification Service will match the new Flavors instead of the Flavors that were modified with bad values.

After new Flavors have been imported from each host, retrieve the Reports for each host again to verify that each host is Trusted:

```plaintext
GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true
```

Headers:
Accept: application/json

The “Overall” value should be “true” for each host.
5 Use Case 1c: Platform Attestation (VMware*)

The Platform Attestation use case will enable external platform integrity verification using TCG-standard practices. This feature is the foundation for other Intel® SecL – DC use cases.

Platform Attestation involves taking measurements of system components during system boot, and then cryptographically verifying that the actual measurements taken match a set of expected or approved values, ensuring that the measured components were in an acceptable or “trusted” state at the time of the last system boot.

Intel® SecL – DC leverages the Trusted Compute Group specification for a trusted boot process, extending measurements of platform components to registers in a TPM, and securely generating quotes of those measurements from the TPM for remote comparison to expected values (attestation).

Deployment will include only the Verification Service, which is typically installed on a Virtual Machine.

Platform Attestation for VMware ESXi* servers does not require a Trust Agent. Instead, it requires communication with vCenter Server*; vCenter* and ESXi will handle the TPM-specific functions, and the Intel® SecL – DC Verification Service will retrieve the TPM quote for a given host from a vCenter API.

5.1 Scope

This use case will include the installation of the Verification Service. After installation, the Platform Attestation use case will be demonstrated by registering the VMware ESXi hosts, importing Flavors from the registered hosts, retrieving Attestation Reports for the registered hosts and performing a negative test to demonstrate an “Untrusted” result.

5.2 Requirements

5.2.1 Hardware

- One or more server platforms with Intel® TXT supported and enabled in the system BIOS
- Each physical server must have a physical or firmware TPM installed and active in the system BIOS. Both TPM versions 1.2 and 2.0 are supported for VMware. The TPM must have “cleared” ownership (the TPM ownership can be cleared in the system BIOS) and must be “Enabled” in the system BIOS (TPM 1.2 only; TPM 2.0 is always “Enabled”)


NOTE: for VMware servers, UEFI SecureBoot MUST be enabled to support TPM 2.0. To support TPM 1.2, UEFI SecureBoot MUST NOT be enabled.

- One server (which may be physical or virtual) dedicated for the Intel® SecL – DC Verification Service. This server does not require a TPM or Intel® TXT, but does require network access to the other POC servers

5.2.2 Software

- (Optional, required for database modification for optional Untrusted state demonstration) A GUI-based remote database client that supports the PostgreSQL JDBC driver (for example, SQL Workbench) installed on a laptop or other system outside of the POC environment, but with network connectivity
- cURL or a REST API utility like Postman to execute API requests

5.2.3 Operating System

- The Intel® SecL – DC Verification Service supports RHEL 7.4 and above
- The ESXi version required depends on the TPM version used. For TPM 1.2, ESXi 6.5 Update 2 or later is required; for TPM 2.0, vSphere 6.7 Update 1 or later is required. However, Asset Tags (described in the Asset Tags for VMware section) are not currently supported for VMware TPM 2.0 servers.
- An instance of VMware vCenter Server* is required. The specific version of vCenter required depends on the ESXi version that will be used; see VMware documentation for vCenter and ESXi interoperability requirements. Both the vCenter Virtual Appliance* and the Windows application are supported, but for ease of deployment the Virtual Appliance is recommended. Each ESXi host must be managed by vCenter and be added to a vCenter Cluster* object.

5.2.4 Installing the Verification Service

This section details how to install the Intel® SecL – DC services.

5.2.4.1 Package Dependencies

The Intel® SecL – DC Verification Service requires the following packages and their dependencies:
- Logback (optional)
- Java 8 JDK
- OpenSSL
- Postgres client and server 9.3 (server component optional if an external Postgres database is used)
- Unzip
If they are not already installed, the Verification Service installer will attempt to install these automatically using the package manager.

Automatic installation requires access to package repositories (the RHEL subscription repositories, the EPEL repository, or a suitable mirror), which may require an Internet connection. If the packages are to be installed from the package repository, be sure to update the repository package lists before installation.

5.2.4.2 Supported Operating Systems

The Intel® SecL – DC Verification Service supports RHEL 7.4 and above.

5.2.4.3 Recommended Hardware

- 4 vCPUs
- RAM: 8 GB
- 100 GB
- One network interface with network access to all managed servers

5.2.4.4 Installation

To install the Verification Service, follow these steps:

1. Copy the Verification Service installation binary to the /root/ directory.
2. Create the mtwilson.env installation answer file:

   ```
   ### User credentials
   export MC_FIRST_USERNAME=administrator
   export MC_FIRST_PASSWORD=password

   ### Database Configuration
   export DATABASE_USERNAME=root
   export DATABASE_PASSWORD=password

   ### Service IP or Hostname definition
   export MTWILSON_SERVER=<Verification Service IP address or Hostname>

3. Execute the installer binary.

When the installation completes, the Verification Service will be available. The services can be verified by running mtwilson status from the Verification Service command line.

# mtwilson status
Host Verification Service is running
5.3 Demonstrating the Platform Attestation Use Case

5.3.1 Verify Services

The Version API can be used to confirm that all services are up and running, before proceeding to the actual use case demonstration.

GET https://verification.server.com:8443/mtwilson/v2/version

5.3.2 Register Hosts

Registration creates a host record with connectivity details in the Verification Service database. This host record will be used by the Verification Service to retrieve TPM attestation quotes from vCenter to generate an attestation report.

Each ESXi host will need to be registered with a separate call. Note that the Connection String requires access to vCenter. Provide credentials for a vCenter user with at minimum the "Validate Session" permission (using an Administrator account is acceptable)

POST https://verification.service.com:8443/mtwilson/v2/hosts

{
    "host_name": "<hostname of ESXi host to be registered, as it appears in vCenter>",
    "tls_policy_id" : "TRUST_FIRST_CERTIFICATE",
    "connection_string": "vmware:https://vCenter.server.com/sdk;h=<hostname of ESXi host to be registered, as it appears in vCenter>;u=username@vSphere.local;p=password",
    "flavorgroup_name" : "",
    "description" : "<description>"
}

5.3.3 List Hosts

After registration, the /hosts API can be used to list all registered hosts and confirm that the registrations were all successful.


5.3.4 Retrieve Reports (Untrusted – No Flavors)

You can now retrieve the latest Attestation Report for each host to demonstrate the “faults” shown when match Flavors do not exist in the Verification Service database.

Currently no Flavors should be present, so all of these Reports will show an Untrusted status, with Faults showing that matching Flavors were required but not found.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true
5.3.5 Import Flavors

Next you will actually import Flavors. For simplicity, you can import all three Flavor parts (PLATFORM, OS, and HOST_UNIQUE) from each Trust Agent host.

Technically, only the HOST_UNIQUE part must come from each host, the PLATFORM and OS Flavors can be created just once per version (for example, if all of our Trust Agent hosts use BIOS version 1.23, we only need to import the PLATFORM Flavor for BIOS version 1.23 once, and all other hosts using the same BIOS version will be matched to the same Flavor).

POST https://verification.server.com:8443/mtwilson/v2/flavors
{
  "connection_string": "vmware:https://vCenter.server.com/sdk;h=<hostname of ESXi host to be registered, as it appears in vCenter>
    >;u=username@vSphere.local;p=password",
  "partial_flavor_types": ["PLATFORM", "OS", "HOST_UNIQUE"],
  "flavorgroup_name": "",
  "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}

5.3.6 Retrieve Reports (Trusted)

Now that all Flavors exist and all hosts have been registered, you can retrieve new Attestation Reports. New Reports are automatically generated whenever a host is matched to a new Flavor, which happened when we imported our Flavors.

This is the same request used previously, but this time all of the Reports should show that the “Overall” trust is “True,” and we should no longer see any Faults.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true
Headers:
Accept: application/json

5.3.7 Demonstrate Untrusted State

Finally, it is possible to demonstrate what an Untrusted attestation looks like and what can cause it. There are a number of ways to force a host to attest as Untrusted. In production, a host can become Untrusted if the BIOS or OS kernel are upgraded/downgraded to use a version where no Flavor exists, or if a malicious attacker has inserted malicious code into a measured component, and so forth.

For testing purposes, the easiest ways to force a host to become Untrusted are to either boot the host to a non-tboot boot option (simply reboot, and at the boot menu option when RHEL boots, select the option without tboot).
Attestation Reports are valid for 24 hours by default (this is configurable); you need to force a new Report to be generated now, after our change.

POST https://verification.server.com:8443/mtwilson/v2/reports

Headers:
Accept: application/json
input: {"host_name": "<Trust Agent host>"}

The new Report will show that the PLATFORM Flavor part is still Trusted, but that the HOST_UNIQUE and OS Flavor Parts have mismatched measurements. This is because the platform performs the BIOS-related measurements before the OS boots, but tboot performs the other measurements.

Because the system booted without tboot, the TPM memory registers that store the tboot measurements will all be 0’s or F’s, and will mismatch the expected measurements in the Flavors.

**5.3.7.1 (Optional) Database Modification**

While slightly more complicated, it is also possible to edit one or more Flavors in the database and change the expected measurement results. Because of the size of the Flavor objects in the database, this is best performed with a remote GUI database client like SQL Workbench, as opposed to using the command line. This step is optional, and will show a different Fault more consistent with a platform that has been tampered with by a malicious attack.

First you need to enable remote access to the Verification Service database, which is disabled by default.

In the /var/lib/pgsql/9.3/data/postgresql.conf file, add the following line:

```
listen_addresses='*' 
```

Next, edit /var/lib/pgsql/9.3/data/pg_hba.conf and change the last line to the following:

```
host all all 127.0.0.1/0 password 
```

Finally, restart the Postgresql database:
```
systemctl restart postgresql-9.3 
```

Now you should be able to access the database using a remote database client, like SQL Workbench. By default, the database name is `mw_as`. The username and password were configured in the mtwilson.env answer file during the installation; the example used "root" and "password".

After connecting to the `mw_as` database, look at the `mw_flavor` table. You should see a number of Flavors, including PLATFORM, OS, and HOST_UNIQUE Flavors from when they were imported in an earlier step.
In the “content” column is the actual Flavor object. Find a “PLATFORM” Flavor:

Find the “pcr_0” measurements (depending on the TPM version and your platform OEM, you may see only SHA1, only SHA256, or both measurements).

{"SHA1":{"pcr_0":{"value":"d2ed125942726641a7260c4f92beb67d531a0def"}}

"SHA256":{"pcr_0":{"value":"db83f0e8a1773c21164c17986037cdf8afclbbdc1b815772c6dalbefb1a7f8a3"}}

Change a digit in each of these entries. Do not change the length, simply change the last digit of each “pcr_0” hash to a different hexadecimal digit.

Save the changes to the database.

Now force the creation of a new Report.

POST https://verification.server.com:8443/mtwilson/v2/reports

{ "host_name":"<Trust Agent hostname>"}

Do this for each host in the POC environment. When you get the Report for the host that was using the modified PLATFORM flavor, the OS and HOST_UNIQUE attestations will remain Trusted.

The PLATFORM Flavor part, however, will show that it is Untrusted, with a “PcrValueMismatch” fault indicating that the actual measurement for PCR 0 does not match the expected value.

Because PCR 0 includes the hash of the core system BIOS, this replicates an event where the BIOS has been maliciously modified.
<index>pcr_0</index>

<value>187c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9</value>

<pcrBank>SHA256</pcrBank>

</expectedPcr>

</rule>

<faults>

<faults fault_name="com.intel.mtwilson.core.verifier.policy.fault.PcrValueMismatchSha256">

<description>Host PCR 0 with value 987c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9 does not match expected value 187c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9</description>

<pcrIndex>pcr_0</pcrIndex>

<expectedValue>187c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9</expectedValue>

<actualValue>987c560472a458a563a21e33b2e927383c7d379340d5a98c8529ab82ecda79f9</actualValue>

</faults>

</faults>

5.3.8 Returning the hosts to a Trusted state

Now that you demonstrated an Untrusted state, you need to recover the hosts back to a Trusted state.

The easiest way to do this is to simply import new Flavors for each host:

POST https://verification.server.com:8443/mtwilson/v2/flavors

```
{
    "connection_string": "vmware:https://vCenter.server.com/sdk;h=<hostname of ESXi host to be registered, as it appears in vCenter>;u=<username@vSphere.local>;p=<password>",
    "partial_flavor_types": ["PLATFORM", "OS", "HOST_UNIQUE"],
    "flavorgroup_name": "",
    "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}
```

Repeat this call for each registered host to import new Flavors. This will not overwrite the existing Flavors, but will add new Flavors imported directly from each host.

When new Flavors are imported, the Verification Service will automatically try to see which hosts (if any) should be matched to those Flavors. Because the new Flavors will actually match the PCR values seen on the host TPM, the Verification Service will match the new Flavors instead of the Flavors that were modified with bad values.

After new Flavors have been imported from each host, retrieve the Reports for each host again to verify that each host is Trusted:
GET
https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json

The “Overall” value should be “true” for each host.
6 Use Case 2a: Asset Tags (Linux)

6.1 Scope

This use case will expand on the Platform Attestation use cases to add attestation of hardware-based Asset Tags. This use case will not include any new software installations, and will instead use the existing software with new API requests to provision Asset Tags to hosts and then generate new Attestation Reports to see the results.

6.2 Requirements

Use Case 1a, Platform Attestation (Linux) must have been completed. The same hardware and software resources will be used to complete this use case.

No additional hardware or software is required beyond what was used for the previous use case.

6.3 Demonstrating the Asset Tag Use Case for Linux

6.3.1 Verify that hosts are registered and Trusted

To begin, you need to verify that all prerequisite steps were completed.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:

Accept: application/json

Verify that the response lists all of the hosts, and that each Report shows "OVERALL=true".

6.3.2 Create Asset Tag Certificates for each host

Asset Tags are a combination of an Asset Tag Certificate, and an Asset Tag Flavor based on that certificate that is associated with a host.

The Asset Tag Certificate is unique for each host, even if all of the key/value pairs are identical. The Subject of the certificate is the Hardware UUID of the server; this is used as part of the later Asset Tag Flavor matching, and to ensure that each Tag must be applied to a specific server.
The Asset Tag Certificate creation call consists of the Hardware UUID of the host, and then a list of any number of key/value pairs. The key/value pairs are the actual Tags that will be reflected in the Attestation Reports for the host, and can represent locations or any other information. For example, a server could be tagged with “Country=USA; Department=Finance; Compliance=PCI”.

Repeat this call for each host.

POST https://verification.server.com:8443/mtwilson/v2/tag-certificates
{"hardware_uuid": ">hardware UUID of host to be tagged"},
"selection_content": [
{
    "name": "Country",
    "value": "USA"
},
{
    "name": "Department",
    "value": "Finance"
},
{
    "name": "Compliance",
    "value": "PCI"
}
]

6.3.3 Deploy Asset Tags

After the Asset Tag Certificates are all created, they need to actually be deployed to the physical servers. On the backend, this process involves generating a cryptographic hash of the Certificate and writing that hash to a specific NVRAM index on the TPM of the server to be tagged. Making the Tag deployment request will also automatically generate the Asset Tag Flavor.

The deployment request is very simple, and only requires the ID of the Asset Tag Certificate to be deployed and the hostname or IP address (however the host is registered in the “host_name” field when the registration call is made).

The Verification Service will search for the host using the hostname or IP address and use the connection string provided at the time of registration to make a request to the Trust Agent running on that host. The Trust Agent will then perform the actual TPM commands to write the hash to the TPM.

POST https://verification.server.com:8443/mtwilson/v2/rpc/deploy-tag-certificate
{
    "certificate_id": "<certificate ID>",
    "host": "<Hostname of host to be tagged>"
}

This request will also cause a new Asset Tag Flavor to be created for the actual attestation process. Repeat this step for each host to be tagged.
6.3.4 Create new Attestation Reports for each host

Deploying the Asset Tags does not actually create a new Attestation Report, so you will need to create a new Report for each tagged host.

Instead of only retrieve existing automatically-generated Reports, you will create a new Report for each host.

POST https://verification.server.com:8443/mtwilson/v2/reports

Headers:
Accept: application/json
{"host_name":"<hostname or IP address>"}

Repeat this step for each registered host.

Note that each new Report will now contain an additional Flavor section for the Asset Tag; this will show whether the Tag is Trusted (meaning the most recent currently valid Asset Tag certificate for that host is actually reflected in the host TPM), and will expose all of the key/value pairs used in the Asset Tag Certificate that was deployed.
Use Case 2b: Asset Tags (Windows)

7.1 Scope

This use case will expand on the Platform Attestation use cases to add attestation of hardware-based Asset Tags. This use case will not include any new software installations, and will instead use the existing software with new API requests to provision Asset Tags to hosts and then generate new Attestation Reports to see the results.

7.2 Requirements

Use Case 1b, Platform Attestation (Windows) must have been completed. The same hardware and software resources will be used to complete this use case.

No additional hardware or software is required beyond what was used for the previous use case.

7.3 Demonstrating the Asset Tag Use Case for Windows

7.3.1 Verify that hosts are registered and Trusted

To begin, you need to verify that all prerequisite steps were completed.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json

Verify that the response lists all of the hosts, and that each Report shows "OVERALL=true".

7.3.2 Create Asset Tag Certificates for each host

Asset Tags are a combination of an Asset Tag Certificate, and an Asset Tag Flavor based on that certificate that is associated with a host.

The Asset Tag Certificate is unique for each host, even if all of the key/value pairs are identical. The Subject of the certificate is the Hardware UUID of the server; this is used as part of the later Asset Tag Flavor matching, and to ensure that each Tag must be applied to a specific server.
The Asset Tag Certificate creation call consists of the Hardware UUID of the host, and then a list of any number of key/value pairs. The key/value pairs are the actual Tags that will be reflected in the Attestation Reports for the host, and can represent locations or any other information. For example, a server could be tagged with “Country=USA; Department=Finance; Compliance=PCI”.

Repeat this call for each host.

POST https://verification.server.com:8443/mtwilson/v2/tag-certificates
{"hardware_uuid": "<hardware UUID of host to be tagged>",
"selection_content": [
{
   "name": "Country",
   "value": "USA"
},
{
   "name": "Department",
   "value": "Finance"
},
{
   "name": "Compliance",
   "value": "PCI"
}
]
}

7.3.3 Deploy Asset Tags

After the Asset Tag Certificates are all created, they need to actually be deployed to the physical servers. On the backend, this process involves generating a cryptographic hash of the Certificate and writing that hash to a specific NVRAM index on the TPM of the server to be tagged. Making the Tag deployment request will also automatically generate the Asset Tag Flavor.

The deployment request is very simple, and only requires the ID of the Asset Tag Certificate to be deployed, and the hostname or IP address (however the host is registered in the “host_name” field when the registration call is made).

The Verification Service will search for the host using the hostname or IP address and use the connection string provided at the time of registration to make a request to the Trust Agent running on that host. The Trust Agent will then perform the actual TPM commands to write the hash to the TPM.

POST https://verification.server.com:8443/mtwilson/v2/rpc/deploy-tag-certificate
{
   "certificate_id": "<certificate ID>",
   "host": "<Hostname of host to be tagged>"
}

This request will also cause a new Asset Tag Flavor to be created for the actual attestation process. Repeat this step for each host to be tagged.
7.3.4  Create new Attestation Reports for each host

Deploying the Asset Tags does not actually create a new Attestation Report, so you will need to create a new Report for each tagged host.

Instead of only retrieve existing automatically-generated Reports, you will create a new Report for each host.

POST https://verification.server.com:8443/mtwilson/v2/reports

Headers:
Accept: application/json
{"host_name":"<hostname or IP address>"}

Repeat this step for each registered host.

Note that each new Report will now contain an additional Flavor section for the Asset Tag. This will show whether the Tag is Trusted (meaning the most recent currently valid Asset Tag certificate for that host is actually reflected in the host TPM), and will expose all of the key/value pairs used in the Asset Tag Certificate that was deployed.
8 Use Case 2c: Asset Tags (VMware)

8.1 Scope

This use case will expand on the Platform Attestation use cases to add attestation of hardware-based Asset Tags. This use case will not include any new software installations, and will instead use the existing software with new API requests to provision Asset Tags to hosts and then generate new Attestation Reports to see the results.

8.2 Requirements

Use Case 1c, Platform Attestation (VMware) must have been completed. The same hardware and software resources will be used to complete this use case. No additional hardware or software is required beyond what was used for the previous use case. However, SSH must be enabled for the ESXi hosts to be tagged. See VMware documentation for instruction on enabling the SSH service on an ESXi host from vCenter. SSH can be safely disabled after the tags are written.

8.3 Demonstrating the Asset Tag Use Case for VMware

8.3.1 Verify that hosts are registered and Trusted

To begin, you need to verify that all prerequisite steps were completed.

GET https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true

Headers:
Accept: application/json

Verify that the response lists all of the hosts, and that each Report shows “OVERALL=true”.

8.3.2 Create Asset Tag Certificates for each host

Asset Tags are a combination of an Asset Tag Certificate, and an Asset Tag Flavor based on that certificate that is associated with a host.

The Asset Tag Certificate is unique for each host, even if all of the key/value pairs are identical. The Subject of the certificate is the Hardware UUID of the
server; this is used as part of the later Asset Tag Flavor matching, and to ensure that each Tag must be applied to a specific server.

The Asset Tag Certificate creation call consists of the Hardware UUID of the host and then a list of any number of key/value pairs. The key/value pairs are the actual Tags that will be reflected in the Attestation Reports for the host, and can represent locations or any other information. For example, a server could be tagged with “Country=USA; Department=Finance; Compliance=PCI”.

Repeat this call for each host.

```json
POST https://verification.server.com:8443/mtwilson/v2/tag-certificates
{
  "hardware_uuid": "<hardware UUID of host to be tagged>",
  "selection_content": [
    {
      "name": "Country",
      "value": "USA"
    },
    {
      "name": "Department",
      "value": "Finance"
    },
    {
      "name": "Compliance",
      "value": "PCI"
    }
  ]
}
```

---

### 8.3.3 Deploy Asset Tags

#### 8.3.3.1 Calculate the Certificate Hash Value

Only the hash value of the Asset Tag Certificate can be provisioned to the TPM, due to the low size of the NVRAM.

1. Retrieve the Asset Tag Certificate. The Asset Tag Certificate can be retrieved either from the response when the Asset Tag certificate is created or by using a GET API request to retrieve the certificate:

```bash
GET https://verification.server.com:8443/mtwilson/v2/tag-certificates?subjectEqualTo=<HardwareUUID>
```

2. Copy only the “certificate” value (this will be the certificate in encoded format) and write the data to a file on a Linux system. Remove any line breaks and save the file. Assuming the filename used is “tag-cert”, use the following string to generate the correct hash:

```bash
cat tag-cert | base64 --decode | openssl dgst -sha1 | awk '{print $2}'
```

This hash value will be what is actually written to the TPM NVRAM.
8.3.3.2 **Provision the Certificate Hash to the Host TPM**

Starting in ESXi 6.5u2, you can now use SSH to write Asset Tags directly with no need for TPM clears, reboots, PXE or BIOS access. SSH to the ESXi host using root credentials, then use the next command:

```
esxcli hardware tpm tag set -d <hash>
```

You can use the following command to verify that the tag was written:

```
esxcli hardware tpm tag get
```

Reboot the host. After rebooting, the TPM PCR 22 will have the measured value of the hash.

8.3.4 **Create Asset Tag Flavor**

For VMware ESXi hosts, the Asset Tag Flavor must be created by importing it from the host after the Tag has been provisioned.

POST https://verification.server.com:8443/mtwilson/v2/flavors

```
{
  "connection_string": "vmware:https://vCenter.server.com/sdk;h=<hostname of ESXi host to be registered, as it appears in vCenter>;u=<username@vSphere.local>;p=<password>",
  "tls_policy_id": "TRUST_FIRST_CERTIFICATE",
  "partial_flavor_types": ["ASSET_TAG"]
}
```

Once the Asset Tag Flavor is imported, the host can be attested including Asset Tags as normal.

8.3.5 **Create new Attestation Reports for each host**

Deploying the Asset Tags does not actually create a new Attestation Report, so you will need to create a new Report for each tagged host.

Instead of only retrieve existing automatically-generated Reports, you will create a new Report for each host.

POST https://verification.server.com:8443/mtwilson/v2/reports

Headers:

```
Accept: application/json
```

```
{"host_name": "<hostname or IP address>"}
```

Repeat this step for each registered host.

Note that each new Report will now contain an additional Flavor section for the Asset Tag; this will show whether the Tag is Trusted (meaning the most recent currently valid Asset Tag certificate for that host is actually reflected in the host TPM), and will expose all of the key/value pairs used in the Asset Tag Certificate that was deployed.
9 Use Case 3: Application Integrity

9.1 Scope
This use case will expand on the Platform Attestation use cases to add attestation of user-defined applications. This use case will not include any new software installations, and will instead use the existing software with new API requests to specify new files and folders to be measured and attested.

9.2 Requirements
Use Case 1a, Platform Attestation (Linux) and Use Case 2a, Asset tags (Linux) must have been completed. The same hardware and software resources will be used to complete this use case.

No additional hardware or Intel® SecL software is required beyond what was used for the previous use cases.

9.3 Demonstrating the Application integrity Use Case for Linux

9.3.1 Verify that hosts are registered and Trusted
To begin verify that all prerequisite steps were completed.
GET
https://verification.server.com:8443/mtwilson/v2/reports?latestPerHost=true
Headers:
Accept: application/json

Verify that the response lists all of the hosts, and that each Report shows "OVERALL=true".

9.3.2 Create a New SOFTWARE Flavor
Creating a new SOFTWARE Flavor requires specifying a sample host where the application, files or folders that will be measured are currently present. The measurements specified in the manifest will be captures when this call is executed, and the Verification Service will communicate with the Trust Agent and create a SOFTWARE Flavor based on the file measurements.
The software manifest below includes files related to the Trust Agent, but these can be defined for any application on a bare-metal server by specifying the static files and folders that comprise that application.

POST https://server.com:8443/mtwilson/v2/flavor-from-app-manifest

Input:
<ManifestRequest xmlns="lib:wml:manifests:req:1.0">
<connectionString>intel:https://trustagent.server.com:1443;u=trustagentUsername;p=trustagentPassword</connectionString>
<Manifest xmlns="lib:wml:manifests:1.0" DigestAlg="SHA384" Label="Samplev1" Uuid=""> +
  <Dir Type="dir" Include=".*" Exclude="" Path="/opt/trustagent/hypertext/WEB-INF" />
  <Symlink Path="/opt/trustagent/bin/tpm_nvinfo" />
  <File Path="/opt/trustagent/bin/module_analysis_da.sh" />
</Manifest>
</ManifestRequest>

9.3.3 Deploy the Application Manifest

Once the new SOFTWARE Flavor is created, its manifest can be deployed to any number of Trust Agent servers. Deployment can be performed with an API request to the Verification Service, and requires the ID of the host, and the ID of the SOFTWARE Flavor.

POST https://server.com:8443/mtwilson/v2/rpc/deploy-software-manifest
Input:
{
  "flavor_id":"a2345ff4-6dc7-4c74-82be-578592e7e3ba",
  "host_id":"45874ff4-6d37-5875-82be-12392e7e123"
}

After the manifest has been deployed, the Trust Agent server must be rebooted. This will allow tbootXM to measure the files and folders specified in the manifest, and extend them to the TPM for attestation.

9.3.4 Create new Attestation Reports for each host

After rebooting each server after deploying the SOFTWARE Flavor manifest, a new Attestation Report must be generated to reflect the new Flavor.

Instead of only retrieve existing automatically-generated Reports, you will create a new Report for each host.
POST https://verification.server.com:8443/mtwilson/v2/reports
Headers:
Accept: application/json
{"host_name":"<hostname or IP address>"}

Repeat this step for each registered host.
Note that each new Report will now contain an additional SOFTWARE Flavor, in addition to the default Trust Agent SOFTWARE Flavor. Because the default Flavor Match Policy for SOFTWARE Flavors is “all_of”, both of the SOFTWARE Flavors must match the host attestation for the host to be Trusted.
10 Use Case 4: OpenStack Orchestration*

OpenStack Orchestration* with Intel® SecL – DC involves using security attributes ("Trust" and Asset Tag key/value pairs) to define image launch requirements and populate the compute nodes with traits reflecting their status in the Intel® SecL – DC Verification Service.

In this way, an image owner can define requirements for the image so that instances of that image are always launched on Trusted compute nodes with specific Asset Tag key/value pairs.

10.1 Scope

This use case will build on Use Cases 1a and 2a and add OpenStack integration to demonstrate using Intel® SecL – DC to control where Virtual Machines (VMs) are allowed to launch.

The installation of OpenStack will not be described here. For OpenStack installation instructions, see OpenStack documentation. Also see Section 3.2 for the list of OpenStack services required for this use case.

In addition to the Intel® SecL – DC components previously installed in Use Cases 1a and 2a, Integration Hub* will also be installed. The Integration Hub is the Intel® SecL – DC component that assigns OpenStack environments to specific tenants and handles pushing the required attributes to OpenStack.

After installation, the Use Case will be demonstrated by setting Image Traits to represent the required attributes for hosts to launch instances of protected images. A successful launch on a Trusted host whose Traits match the requirements in the Image Traits will be demonstrated, followed by a failed launch where the Asset Tags or Trust Status are not met by any available compute node.

10.2 Requirements

Use Cases 1a and 2a must have been completed prior to starting the OpenStack Orchestration use case. The same software and hardware from those previous use cases will be used and expanded upon (no new hardware is required, but additional software will need to be installed).

10.2.1 Hardware

- One or more server platforms with a supported Root of Trust configuration.
- Each physical server must have a physical or firmware Trusted Platform Module version 2.0 installed and active in the system BIOS. The TPM must
have “cleared” ownership (the TPM ownership can be cleared in the system BIOS).

- One server (which may be physical or virtual) dedicated for the Intel® SecL – DC Verification Service. This server does not require a TPM or Intel® TXT, but does require networking access to the other POC servers.
- One server (which may be physical or virtual) for the Intel® SecL – DC Integration Hub. This server does not require a TPM or Intel® TXT, but does require networking access to the other POC servers. This server can be the same server used for the Verification Service.

10.2.2 Software

- (Optional, required for database modification for optional Untrusted state demonstration) A GUI-based remote database client that supports the PostgreSQL JDBC driver (for example, SQL Workbench) installed on a laptop or other system outside of the POC environment, but with network connectivity.
- cURL or a REST API utility like Postman to execute API requests.

10.2.3 Operating System

- The Intel® SecL – DC Verification Service supports RHEL 7.4 and above.
- The Intel® SecL – DC Trust Agent for Linux supports RHEL 7.4 and above.
- The Intel® SecL – DC Integration Hub supports RHEL 7.4 and above.

10.2.4 Networking

- The Intel® SecL – DC Verification Service must be able to reach each physical server over the network.
- The Intel® SecL – DC Integration Hub must be able to reach the Intel® SecL – DC Verification Service and OpenStack services over the network.
- The Intel® SecL – DC installers will require access to package repositories. Either internet access or access to suitable repository mirrors must be provided (as well as any applicable subscriptions to access the repositories).
- Hostname resolution is not necessary but is very helpful, particularly with OpenStack. IP addresses may be used in place of hostnames. Ensure that IP addresses and hostnames are resolvable from all Trust Agent hosts to the Verification Service and vice versa. Be consistent – use either all IP addresses, or all hostnames, do not mix-and-match.

10.2.5 OpenStack

- OpenStack Rocky*: Nova*, Glance*, Horizon*, and Keystone* services must be installed and running. Note that specifically OpenStack Rocky or later is required.
A minimum of one OpenStack compute node with the Intel® SecL – DC Trust Agent deployed as per Use Case 1a. Generally this means installing OpenStack Nova Compute* on the same server(s) used as Trust Agent servers in the previous use cases.

10.2.6 Installation

To install the Integration Hub, follow these steps:

1. Copy the Integration Hub installation binary to the /root/ directory.

2. Create the `attestation-hub.env` installation answer file in /root/.

   ```
   MTWILSON_API_URL="https://<Verification Service IP or hostname>:8443/mtwilson/v2"
   MTWILSON_TLS=<sha384 of Verification Service TLS certificate>
   MTWILSON_USERNAME=administrator
   MTWILSON_PASSWORD=password
   
   ATTESTATION_HUB_DB_NAME="attestation_hub_pu"
   ATTESTATION_HUB_DB_HOSTNAME="localhost"
   ATTESTATION_HUB_DB_PORTNUM="5432"
   ATTESTATION_HUB_DB_DRIVER="org.postgresql.Driver"
   ATTESTATION_HUB_DB_USERNAME=root
   ATTESTATION_HUB_DB_PASSWORD=password
   ATTESTATION_HUB_PORT_HTTP=19082
   ATTESTATION_HUB_PORT_HTTPS=19445
   
   Note: The MTWILSON_TLS value can be retrieved from the Verification Service using the following command from the Verification Service command line:
   cat /opt/mtwilson/configuration/https.properties
   
   3. Execute the installer binary.
   
   4. Create an administrative user. On the command line while logged in as root, run the following:

      ```
      attestation-hub password hubadmin password --permissions *:*
      ```

      This will create a new Hub user named “hubadmin” with password “password” that will be used to make REST API requests to the Hub.
10.3 Demonstrating the OpenStack Orchestration* Use Case

10.3.1 Configuring a Tenant in the Integration Hub

10.3.1.1 Create the Tenant

At least one tenant must be created to receive the attestations. For the Hub, a single tenant is typically a single OpenStack controller. A Tenant defines the connection and authentication details to reach the OpenStack services.

POST https://hub.server.com:19445/v1/tenants

{
    "name": "DemoTenant",
    "plugins": [
        {
            "name": "nova",
            "properties": [
                {
                    "key": "api.endpoint",
                    "value": "http://<Nova API endpoint>/compute/v2.1"
                },
                {
                    "key": "auth.endpoint",
                    "value": "http://<Keystone API endpoint>:5000/identity"
                },
                {
                    "key": "auth.version",
                    "value": "v3"
                },
                {
                    "key": "user.name",
                    "value": "<Username for Nova API>"
                },
                {
                    "key": "user.password",
                    "value": "<Password for Nova API>"
                },
                {
                    "key": "tenant.name",
                    "value": "<Name of tenant in OpenStack>"
                },
                {
                    "key": "domain.name",
                    "value": "<Name of Domain in OpenStack>"
                }
            ]
        }
    ]
}
10.3.1.2 Assign Hosts to the Tenant

Hosts must be assigned to a tenant before Intel® SecL – DC security attributes will be pushed to the OpenStack Traits. Any number of hosts may be assigned to one tenant. Multiple hosts can be assigned to a tenant in a single request by using a comma-separated list of hardware_uuids.

Hosts are assigned using the Tenant ID (returned in the Create Tenant step) and the Hardware UUID of one or more hosts. List each OpenStack Compute Node’s Hardware UUID in the array.

POST https://hub.server.com:19445/v1/host-assignments

```
{
  "tenant_id": "<Tenant ID>",
  "hardware_uuids": [
    "<Host 1 Hardware UUID>", "<Host 2 Hardware UUID>"
  ]
}
```

10.3.1.3 Verify that the Hub is Retrieving Reports

Next we want to list the hosts as seen in the Hub to ensure the Hub is communicating with the Verification Service and retrieving Reports.

GET https://hub.server.com:19445/v1/hosts

This will return a list of all of the hosts seen by the Hub with their most recent Report status. By default, the Hub will poll the Verification Service for new Reports every 2 minutes, refresh this list and then send updates to all Tenant endpoints according to which hosts were assigned to which Tenant.

10.3.2 Setting Launch Requirements in Image Traits

Image Traits define the policy for which Traits are required for that Image to be launched on a Nova Compute node. By setting these Traits to “required,” the OpenStack scheduler will require these same Traits to be present on a Nova Compute node in order to launch instances of the image.

To require a “Trusted” Attestation Report:

```
CUSTOM_ISECL_TRUSTED=required
```

The naming convention for Asset Tags is more flexible, and any number of these Traits can be used simultaneously.
**Note:** All of the Traits must be present on the Compute Node for the scheduler to allow instances to land, so be sure not to set mutually exclusive Asset Tag values.

CUSTOM_ISECL_AT_TAG_<key>_<value>=required

In Use Case 2a, we applied Asset Tags to the hosts using the following key/value pairs:

Country=USA; Department=Finance; Compliance=PCI

We can then make an Image Trait that will require hosts to be tagged with "Country=USA" to meet the placement requirements.

CUSTOM_ISECL_AT_TAG_COUNTRY_USA=required

These Traits can be set using CLI commands for OpenStack Glance*, using the name of your VM image:

```
openstack image set --property trait:CUSTOM_ISECL_AT_TAG_COUNTRY_USA=required <image_name>
```

```
openstack image set --property trait:CUSTOM_ISECL_TRUSTED=required <image_name>
```

This image will now require hosts that are currently Trusted, and that have the "Country=USA" tag. Other tags will be disregarded. Additional Traits can be set to require more than one Asset Tag key/value pair, in which case all of the required Traits must be present – partial matches will fail.

### 10.3.3 Launching an Instance on a Compliant Node

Simply launch an instance of the image using the OpenStack Placement service. This can be done using the OpenStack Horizon UI*, the commandline, or using a REST API.

Below is an example of the REST API for launching a VM. See OpenStack documentation for other examples:

```
POST http://openstack.compute.com:port/v2.1/servers
{
    "server" : {
        "accessIPv4": "1.2.3.4",
        "name" : "LaunchOnTrustedWithTags",
        "imageRef" : "<Image ID or path to image>",
        "flavorRef" : <flavor>",
        "metadata" : {
            "My Server Name" : "LaunchOnTrustedWithTags"
        }
    }
}
```

Because the Trust Status of the compute node(s) is currently Trusted, and because the tag “Country=USA” is actually present on the host(s), the new VM instance should launch successfully.
10.3.4 Launching an Instance where No Nodes are Compliant

The OpenStack Placement Service filters the compute nodes and will skip those that do not meet the requirements of the VM (like available CPU, memory, or custom attributes like Intel® SecL – DC Trust and Asset tags).

This means that if the `CUSTOM_ISECL_TRUSTED=required` Trait is present in the Image Properties, any “Untrusted” host (where “Overall=false” in the Trust Report, or if no Report is available, or if the Report is expired), will be left out of the schedule.

This is easily demonstrated by applying an Asset Tag requirement for a key/value pair that does not exist on any host.

To do this, apply the Trait `CUSTOM_ISECL_AT_TAG_EXIST_FALSE=required` to the Image Properties:

```
openstack image set --property trait:CUSTOM_ISECL_AT_TAG_EXIST_FALSE=required <image_name>
```

This can be applied in addition to the Trait requirements that already exist. In this way we'll have a set of requirements, some of which can be met by our compute node(s) and one of which cannot.

Once the Trait is set on the Image Properties, launch a new instance of the Image.

The launch will fail, with an error saying that no suitable host was found.

In the OpenStack Nova API log, an error will show that the Image Traits requirements were not met on any available host.
Through the use of Custom Resource Definitions for the Kubernetes Master, Intel® Security Libraries can make Kubernetes aware of Intel® SecL security attributes and make them available for pod orchestration. In this way, a security-sensitive pod can be launched only on “Trusted” physical worker nodes, or on physical worker nodes that match specified Asset Tag values.

**NOTE:** This control only applies to pods launched using the Kubernetes scheduler, and these scheduling controls will not affect manually-launched instances where a specific worker node is defined (since this does not use the scheduler at all). Intel SecL-DC uses existing Kubernetes interfaces and does not modify Kubernetes code, using only the standard Custom Resource Definition mechanism to add this functionality to the Kubernetes Master. The datacenter owner or Kubernetes administrator is responsible for the security of the Kubernetes workload scheduling process in general, and Intel recommends following published Kubernetes security best practices.

### 11.1 Scope

This use case will build on Use Cases 1a and 2a and add Kubernetes integration to demonstrate using Intel® SecL – DC to control where Kubernetes pods are allowed to launch.

The installation of Kubernetes will not be described here. For Kubernetes installation instructions, see Kubernetes documentation.

This document does not specify a particular pod or specific container images to use.

In addition to the Intel® SecL – DC components previously installed in Use Cases 1a and 2a, Integration Hub* will also be installed. The Integration Hub is the Intel® SecL – DC component that assigns Kubernetes environments to specific tenants and handles pushing the required attributes to Kubernetes.

After installation, the Use Case will be demonstrated by configuring pods with matchExpressions to represent the required attributes for hosts to launch instances of protected pods. A successful launch on a Trusted host whose security attributes match the requirements will be demonstrated, followed by a failed launch where the Asset Tags or Trust Status are not met by any available compute node.
11.2 Requirements

11.2.1 Hardware

- One or more server platforms with in a supported Root of Trust configuration.
- Each physical server must have a physical or firmware Trusted Platform Module version 2.0 installed and active in the system BIOS. The TPM must have “cleared” ownership (the TPM ownership can be cleared in the system BIOS).
- One server (which may be physical or virtual) dedicated for the Intel® SecL – DC Verification Service. This server does not require any Root of Trust configuration, but does require network access to the other POC servers.
- (Optional, required for database modification for optional Untrusted state demonstration) A GUI-based remote database client that supports the PostgreSQL JDBC driver (for example, SQL Workbench) installed on a laptop or other system outside of the POC environment, but with network connectivity.
- cURL or a REST API utility like Postman to execute API requests.

11.2.2 Operating System

- The Intel® SecL – DC Verification Service supports RHEL 7.4 and above.
- The Intel® SecL – DC Trust Agent for Linux supports RHEL 7.4 and above.
- The Intel® SecL – DC Integration Hub supports RHEL 7.4 and above.

11.2.3 Networking

- The Intel® SecL – DC Verification Service must be able to reach each physical server over the network.
- The Intel® SecL – DC Integration Hub must be able to reach the Intel® SecL – DC Verification Service and OpenStack services over the network.
- The Intel® SecL – DC installers will require access to package repositories. Either internet access or access to suitable repository mirrors must be provided (as well as any applicable subscriptions to access the repositories).
- Hostname resolution is not necessary but is very helpful, particularly with Kubernetes. IP addresses may be used in place of hostnames. Ensure that IP addresses and hostnames are resolvable from all Trust Agent hosts to the Verification Service and vice versa. Be consistent – use either all IP addresses, or all hostnames, do not mix-and-match.

11.2.4 Kubernetes

- Kubernetes Master Node must be installed and running
A minimum of one Kubernetes Worker Node with the Intel® SecL – DC Trust Agent deployed as per Use Case 1a. Generally this means installing the Kubernetes Worker Node on the same server(s) used as Trust Agent servers in the previous use cases.

11.2.5 Installation

11.2.5.1 Installing the Integration Hub

To install the Integration Hub, follow these steps:

5. Copy the Integration Hub installation binary to the /root/ directory.
6. Create the attestation-hub.env installation answer file in /root/. See the next sample file:

```
MTWILSON_API_URL="https://<Verification Service IP or hostname>:8443/mtwilson/v2"
MTWILSON_TLS=<sha384 of Verification Service TLS certificate>
MTWILSON_USERNAME=administrator
MTWILSON_PASSWORD=password

ATTESTATION_HUB_DB_NAME="attestation_hub_pu"
ATTESTATION_HUB_DB_HOSTNAME="localhost"
ATTESTATION_HUB_DB_PORTNUM="5432"
ATTESTATION_HUB_DB_DRIVER="org.postgresql.Driver"
ATTESTATION_HUB_DB_USERNAME=root
ATTESTATION_HUB_DB_PASSWORD=password
ATTESTATION_HUB_PORT_HTTP=19082
ATTESTATION_HUB_PORT_HTTPS=19445
```

**Note:** The MTWILSON_TLS value can be retrieved from the Verification Service using the following command from the Verification Service command line:
cat /opt/mtwilson/configuration/https.properties

7. Execute the installer binary.
8. Create an administrative user. On the command line while logged in as root, run the following:

```
attestation-hub password hubadmin password --permissions *:*
```

This will create a new Hub user named “hubadmin” with password “password” that will be used to make REST API requests to the Hub.

11.2.5.2 Installing the Intel® SecL-DC Custom Resource Definitions for Kubernetes

Intel® SecL uses Custom Resource Definitions to add the ability to base orchestration decisions on Intel® SecL security attributes to Kubernetes. These CRDs allow Kubernetes administrators to configure pods to require
specific security attributes so that the Kubernetes Master Node will schedule those pods only on Worker Nodes that match the specified attributes.

Perform the following steps on the Kubernetes Master Node:

1) Add a mount path to the kube-scheduler.yaml file for the Intel SecL scheduler extension:

   - mountPath: /opt/isecl-extensions/bin/
     name: extendedsched
     readOnly: true

2) Add a volume path to the kube-scheduler.yaml file for the Intel SecL scheduler extension:

   - hostPath:
     path: /opt/isecl-extensions/bin/
     type: ""
     name: extendedsched

3) Copy the isecl-extensions.bin installer to the Kubernetes Master and execute the installer

   ./isecl-extensions.bin

4) The installer will output a set of keystores upon completion into attestation-hub-keystores/. These contain keys that will be used by the Integration Hub to communicate with this Kubernetes Master. Copy the contents of this directory to the Integration Hub:

   scp -r /root/attestation-hub-keystores/* root@integration-hub.server.com:/opt/attestation-hub/configuration/

   Note that the Integration Hub can manage multiple Kubernetes Master environments at the same time, but the keystores must be kept separate. To do this, create subfolders in the Hub configuration directory for each separate Kubernetes environment, and copy the appropriate keystores to the matching subfolder.

5) Copy the Integration Hub public key to the Kubernetes Master:

   scp attestation-hub.server.com:/opt/attestation-hub/configuration/hub_public_key.pem /etc/kubernetes/pki/

6) Run the command systemctl restart kubelet to restart all the control plane container services, including the base scheduler.

7) (Optional) Verify that the Intel® SecL Custom Resource Definitions have been started:

   kubectl get crds
   kubectl get -o json hostattributes.isecl.intel.com
11.3 Demonstrating the Kubernetes Scheduling Use Case

11.3.1 Configuring a Tenant in the Integration Hub

11.3.1.1 Create the Tenant

At least one tenant must be created to receive the attestations. For the Hub, a single tenant is typically a single OpenStack controller. A Tenant defines the connection and authentication details to reach the OpenStack services.

POST https://hub.server.com:19445/v1/tenants

```json
{
    "name": "DemoTenant",
    "plugins": [
        {
            "name": "kubernetes",
            "properties": [
                {
                    "key": "api.endpoint",
                    "value": "https://kubernetes-master.server.com:6443"
                },
                {
                    "key": "tenant.name",
                    "value": "DemoTenant"
                },
                {
                    "key": "plugin.provider",
                    "value": "com.intel.attestationhub.plugin.kubernetes.KubernetesPluginImpl"
                },
                {
                    "key": "kubernetes.client.keystore",
                    "value": "/opt/attestation-hub/configuration/root_k8s_client.jks"
                },
                {
                    "key": "kubernetes.server.keystore",
                    "value": "/opt/attestation-hub/configuration/root_k8s_trust.jks"
                },
                {
                    "key": "kubernetes.client.keystore.password",
                    "value": "<Keystore password>"
                },
                {
                    "key": "kubernetes.server.keystore.password",
                    "value": "<Keystore Password>"
                }
            ]
        }
    ]
}
```
NOTE: the value of kubernetes.client.keystore and kubernetes.server.keystore must be the filesystem path on the Integration Hub that contains the Kubernetes Master keystores output from the scheduler extensions for this tenant. The value of kubernetes.server.keystore.password and kubernetes.server.keystore.password must be the keystore passwords output by the scheduler extensions installer.

11.3.1.2 List Hosts

The Integration Hub periodically queries the Verification Service for the list of all new Reports; only Reports generated after the timestamp of the most recent query are returned. Because host registration will trigger the generation of a new Report, any new hosts added to the Verification Service will be seen in the Hub on the next refresh (determined by the value of the POLL_INTERVAL variable during install).

The list of hosts known to the Integration Hub can be retrieved using the below API sample.

GET https://server.com:19445/v1/hosts

11.3.1.3 Assign Hosts to Tenants

Hosts must be assigned to a tenant before Intel SecL-DC security attributes will be pushed to Kubernetes. Any number of hosts may be assigned to one tenant. Multiple hosts can be assigned to a tenant in a single request by using a comma-separated list of hardware_uuids.

POST https://server.com:19445/v1/host-assignments
{
  "tenant_id": "DC02284A-F525-4094-BA01-E317FE28E15F",
  "hardware_uuids": [ 
    "00886b98-994d-e411-906e-0017a4403562"
  ]
}

The Hub will “push” Intel SecL-DC attributes as OpenStack Traits to the tenant’s configured endpoints (in this case, Nova) every time it looks for new attestations.

11.3.1.4 Assign Hosts to the Tenant

Hosts must be assigned to a tenant before Intel® SecL – DC security attributes will be pushed to the OpenStack Traits. Any number of hosts may be assigned to one tenant. Multiple hosts can be assigned to a tenant in a single request by using a comma-separated list of hardware_uuids.
Hosts are assigned using the Tenant ID (returned in the Create Tenant step) and the Hardware UUID of one or more hosts. List each OpenStack Compute Node’s Hardware UUID in the array.

POST https://hub.server.com:19445/v1/host-assignments

```
{
  "tenant_id": "<Tenant ID>",
  "hardware_uuids": [ 
    "<Host 1 Hardware UUID>", "<Host 2 Hardware UUID>"
  ]
}
```

11.3.1.5 Verify that the Hub is Retrieving Reports

Next we want to list the hosts as seen in the Hub to ensure the Hub is communicating with the Verification Service and retrieving Reports.

GET https://hub.server.com:19445/v1/hosts

This will return a list of all of the hosts seen by the Hub with their most recent Report status. By default, the Hub will poll the Verification Service for new Reports every 2 minutes, refresh this list and then send updates to all Tenant endpoints according to which hosts were assigned to which Tenant.

11.3.2 Configuring Pods to Require Intel® SecL Attributes

1) (Optional) Verify that the worker nodes have had their Intel® SecL security attributes populated:

```
kubectl get nodes --show-labels
```

The output should show the Trust status and any Asset Tags applied to all of the registered Worker Nodes.

2) In Use Case 2a, we applied Asset Tags to the hosts using the following key/value pairs:

```
Country=USA; Department=Finance; Compliance=PCI
```

We can now configure a Pod to require a Trusted status, and the Asset Tags previously set. Add the following to the Pod creation files:

```yaml
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: isecl.trusted
                operator: In
                values: [ "true"
              - key: TAG_Country
                operator: In
```
values:
  - USA

  - key: TAG_Department
    operator: In
    values:
      - Finance

  - key: Compliance
    operator: In
    values:
      - PCI
Below is a full sample Pod config file:

```yaml
---
apiVersion: v1
class: Pod
data:
name: samplepod
spec:
affinity:
nodeAffinity:
requiredDuringSchedulingIgnoredDuringExecution:
nodeSelectorTerms:
- matchExpressions:
  - key: isecl.trusted
    operator: In
    values:
    - "true"
  - key: TAG_Country
    operator: In
    values:
    - USA
  - key: TAG_Department
    operator: In
    values:
    - Finance
  - key: TAG_Compliance
    operator: In
    values:
    - PCI
containers:
- image: nginx
  imagePullPolicy: IfNotPresent
  name: nginx
```

The “isecl.trusted” key defines the requirement for a Trusted host. Only one of these keys should be used. The “TAG_” keys indicate Asset Tags; if the workload should only launch on hosts with the “COUNTRY=USA” Asset Tag, the
pod should be launched with the matchExpression key “TAG_COUNTRY” with the value “USA”.

Note: All of the matchExpression definitions must be true for a given worker node to launch the pod – in the example above, the host must be attested as Trusted with Asset Tags “Country=US,” “Customer=Customer1,” and “State=CA”. If the worker node has additional Asset Tags beyond the ones required, the pod will still be able to be launched on that node. However, if one of the specified Tags is missing or has a different value, that worker node will not be used for that pod.

11.3.3 Launching a Pod on a Compliant Worker Node

Simply launch any Pod using a config file containing the matchExpression policy requirements for Trust and Asset Tags. The Intel® SecL CRDs on the Kubernetes Master will use the Worker Node labels propagated from the Intel® SecL Integration Hub to ensure the Pod is launched according to the policy requirements.

Below is a sample command for the Kubernetes Master to launch a Pod:

```bash
cubectl create -f pod_config.yaml
```

We can verify that the Master Node CRDs contain attributes successfully pushed from the Integration Hub:

```bash
cubectl get crds
cubectl get -o json hostattributes.iseca.intel.com
```

We can list the Worker Nodes and their labels:

```bash
cubectl get nodes --show-labels
```

Finally, we can list the running Pods to confirm that the Pod launched and is running on a compliant Worker Node:

```bash
cubectl get pods
```

11.3.4 Launching a Pod where No Worker Nodes are Compliant

The Kubernetes Master with the Intel® SecL CRDs installed will require the matchExpressions defined in the Pod config file to be true on a given Worker Node to launch the Pod. All of the conditions must be true for a Worker Node to be selected.

To demonstrate a failure to launch when no compliant Worker Nodes are found, we can simply modify the Pod config file to require a set of Asset Tags that do not exist on any Worker Node:
apiVersion: v1
type: Pod
metadata:
  name: samplepod
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: isecl.trusted
                operator: In
                values:
                  - "true"
              - key: TAG_NoHostHasThisTag
                operator: In
                values:
                  - True
  containers:
    - image: nginx
      imagePullPolicy: IfNotPresent
      name: nginx

We can now try to launch this Pod. Since no Worker Nodes have the Asset Tag "NoHostHasThisTag=True", the Master Node will not be able to find a compliant Worker Node, and the Pod launch will fail.

cubectl create -f pod_config.yaml