Contents

1 Introduction..............................................................8
   1.1 Overview........................................................8
   1.1.1 Trusted Computing........................................8

2 Intel® Security Libraries Components ................................12
   2.1 Trust Agent .....................................................12
   2.2 Verification Service ..........................................12
   2.3 Integration Hub ...............................................12

3 Intel® Security Libraries Installation ..............................14
   3.1 Building from Source ........................................14
   3.2 Recommended Service Layout ................................14
       3.2.1 Platform Integrity ......................................14
   3.3 Installing the Verification Service ..........................15
       3.3.1 Package Dependencies ..................................15
       3.3.2 Supported Operating Systems ..........................15
       3.3.3 Recommended Hardware ..................................15
       3.3.4 Installation ...............................................16
   3.4 Installing the Trust Agent for Linux ..........................16
       3.4.1 Package Dependencies ..................................16
       3.4.2 Supported Operating Systems ..........................17
       3.4.3 Creating Service Users ..................................17
       3.4.4 Prerequisites .............................................18
       3.4.5 Installation ...............................................19
   3.5 Installing the Trust Agent for Windows ......................20
       3.5.1 Supported Operating Systems ..........................20
       3.5.2 Prerequisites .............................................20
       3.5.3 Installation ...............................................21
   3.6 Trust Agent Provisioning ......................................22
   3.7 Trust Agent Registration .....................................23
   3.8 Importing the HOST_UNIQUE Flavor ..........................24
   3.9 Installing the Intel® SecL Integration Hub .................25
       3.9.1 Package Dependencies ..................................25
       3.9.2 Supported Operating Systems ..........................25
       3.9.3 Prerequisites .............................................25
       3.9.4 Recommended Hardware ..................................25
       3.9.5 Installation ...............................................26

4 Deployment Using Containers ......................................28
   4.1 Building the iSecL Service Docker Images .................28
       4.1.1 Prerequisites for Building Docker Images ..........28
       4.1.2 Verification Service ......................................29
       4.1.3 Integration Hub ..........................................30
       4.1.4 Trust Agent ...............................................31
   4.2 Deploying the Intel® SecL Service Containers ..............32
       4.2.1 Verification Service ......................................32
       4.2.2 Integration Hub ..........................................35
4.2.3 Trust Agent .................................................. 39

5 Connection Strings ........................................... 42
  5.1 Trust Agent (Windows and Linux) .............................. 42
  5.2 VMware ESXi ................................................. 42

6 Platform Attestation with Intel Security Libraries ............ 43
  6.1 Host Registration ............................................. 44
    6.1.1 Trust Agent .............................................. 44
    6.1.2 Registration via Verification Service API .................. 44
  6.2 Flavor Creation for Automatic Flavor Matching .................. 45
    6.2.1 Importing a Flavor from a Sample Host ..................... 46
    6.2.2 Creating a Flavor Manually ................................ 46
  6.3 Creating the Default SOFTWARE Flavor (Linux Only) ........... 47
  6.4 Creating and Provisioning Asset Tags ......................... 48
    6.4.1 Creating Asset Tag Certificates .......................... 48
    6.4.2 Deploying Asset Tags .................................... 49
  6.5 Retrieving Current Attestation Reports ......................... 52
  6.6 Retrieving Current Host State Information ....................... 52
  6.7 Upgrading Hosts in the Datacenter to a New BIOS or OS Version ... 52
  6.8 Removing Hosts From the Verification Service ................. 52
  6.9 Removing Flavors ............................................ 53
  6.10 Invalidating Asset Tags ..................................... 53
  6.11 Remediating an Untrusted attestation .......................... 53
  6.12 Attestation Reporting ....................................... 54
    6.12.1 Sample Call – Generating a New Attestation Report ...... 55
    6.12.2 Sample Call – Retrieving an Existing Attestation Report .... 55
  6.13 Integration ............................................... 56
    6.13.1 Integration Hub .................................... 56
    6.13.2 Integration with OpenStack ................................. 57
    6.13.3 Integration with Kubernetes ................................ 62

7 Flavor Management ............................................... 67
  7.1 Flavor Format Definitions .................................. 67
    7.1.1 Meta .................................................. 67
    7.1.2 Hardware ............................................. 67
    7.1.3 PCRs ................................................. 68
    7.1.4 Sample PLATFORM Flavor .................................. 69
    7.1.5 Sample OS Flavor .................................... 72
    7.1.6 Sample HOST_UNIQUE Flavor .............................. 78
    7.1.7 Sample ASSET_TAG Flavor .................................. 80
  7.2 Flavor Matching ............................................. 81
    7.2.1 When Does Flavor Matching Happen? .......................... 81
    7.2.2 Flavor Matching Performance ................................ 82
    7.2.3 Flavor Groups ............................................ 83
    7.2.4 Default Flavor Group ...................................... 83
    7.2.5 Flavor Match Policies ..................................... 83
    7.2.6 Flavor Match Event Triggers ................................ 85
    7.2.7 Sample Flavorgroup API Calls .............................. 86
# 11.3 Failover

## 12 TLS Policies

12.1 TLS Policy Types

12.2 Policy Scope

12.2.1 Per-Host

12.2.2 Shared

12.3 Default Policy Selection

12.4 Default TLS Policies

12.4.1 TRUST_FIRST_CERTIFICATE

12.4.2 INSECURE

## 13 User Management

13.1 Permissions

13.2 Roles

13.3 Creating a User for HTTP Basic Authentication

13.3.1 Creating a User – CLI

13.3.2 Creating a User – REST API

13.4 Certificate Authentication

13.4.1 Creating a Login Certificate

## 14 Uninstallation

14.1 Verification Service

14.2 Trust Agent

14.3 Integration Hub

## 15 Appendix

15.1 PCR Definitions

15.1.1 Microsoft Windows Server 2016 Datacenter

15.1.2 Red Hat Enterprise Linux

15.1.3 VMWare ESXi

A.1 Attestation Rules

A.2 Intel TXT and the Trusted Boot Process
## Revision History

<table>
<thead>
<tr>
<th>Document Number</th>
<th>Revision Number</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>• Updated for all GA Failures</td>
<td>May 2019</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>• Updated for version 1.5 release</td>
<td>July 2019</td>
</tr>
<tr>
<td></td>
<td>1.5.1</td>
<td>• Added RPM Support</td>
<td>Nov 2019</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Overview

Intel Security Libraries for Datacenter is a collection of software applications and development libraries intended to help turn Intel platform security features into real-world security use cases.

1.1.1 Trusted Computing

Trusted Computing consists of a set of industry standards defined by the Trusted Computing Group to harden systems and data against attack. These standards include verifying platform integrity, establishing identity, protection of keys and secrets, and more. One of the functions of Intel Security Libraries is to provide a “Trusted Platform,” using Intel security technologies to add visibility, auditability, and control to server platforms.

1.1.1.1 The Chain of Trust

In a Trusted Computing environment, a key concept is verification of the integrity of the underlying platform. Verifying platform integrity typically means cryptographic measurement and/or verification of firmware and software components. The process by which this measurement and verification takes place affects the overall strength of the assertion that the measured and verified components have not been altered. Intel refers to this process as the "Chain of Trust," whereby at boot time, a sequence of cryptographic measurements and signature verification events happen in a defined order, such that measurement/verification happens before execution, and each entity responsible for performing a measurement or verification is measured by another step earlier in the process. Any break in this chain leads to an opportunity for an attacker to modify code and evade detection.

1.1.1.2 Hardware Root of Trust

The Root of Trust, the first link in the chain, can be one of several different options. Anything that happens in the boot process before the Root of Trust must be considered to be within the "trust boundary," signifying components whose trustworthiness cannot be assessed. For this reason, it’s best to use a Root of Trust that starts as early in the system boot process as possible, so that the Chain of Trust during the boot process can cover as much as possible.

Multiple Root of Trust options exist, ranging from firmware to hardware. In general, a hardware Root of Trust will have a smaller "trust boundary" than a firmware Root of Trust. A hardware Root of Trust will also have the benefit of immutability – where firmware can easily be flashed and modified, hardware is much more difficult to tamper with.
1.1.1.2.1 Intel® Trusted Execution Technology (Intel® TXT)

Intel® Trusted Execution Technology is a hardware Root of Trust feature available on Intel® server platforms starting with the Grantley generation. Intel® TXT is enabled in the system BIOS (typically under the Processor > Advanced tab), and requires Intel® VT-d and Intel VT-x features to be enabled as prerequisites (otherwise the option will be grayed out). Intel® TXT will ship "disabled" by default.

1.1.1.2 Intel® BootGuard (Intel® BtG)

Intel® BootGuard is a hardware Root of Trust feature available on Intel® server platforms starting with the Purley-Refresh generation. Unlike Intel® TXT, Intel® BtG is configured in platform fuses, not in the system BIOS. Intel® BtG is fused into several "profiles" that determine the behavior of the feature. Intel® BtG supports both "verify" and "measure" profiles; in "verify" profiles, Intel® BtG will verify the signature of the platform Initial Boot Block (IBB). In "measure" profiles, Intel® BtG will hash the IBB and extend that measurement to a TPM PCR. It is recommended that Intel® BtG be fused into the "measure and verify" profile for maximum protection and auditability.

Because the Intel® BtG profile is configured using fuses, the server OEM/ODM will determine the profile used at manufacturing time. Please contact your server vendor to determine what Intel® BtG profiles are available in their product line.

Because Intel® BtG only measures/verifies the integrity of the IBB, it’s important to have an additional technology handle measurements later in the boot process. Intel® TXT can provide this function using tboot to invoke SINIT, and UEFI SecureBoot can alternatively provide similar functionality (note that Linux users should properly configure Shim and use a signed kernel for UEFI SecureBoot).

1.1.1.3 Supported Trusted Boot Options

Intel® SecL-DC supports several options for Trusted Computing, depending on the features available on the platform.
Note: Intel® recommends using UEFI boot mode, and either TXT + tboot, or TXT + BtG + UEFI SecureBoot, or BtG + UEFI SecureBoot. These options will avoid any incompatibilities and provide the best measured boot coverage. Take note that tboot is not currently compatible with UEFI SecureBoot.

1.1.4 Remote Attestation

Trusted computing consists primarily of two activities—measurement, and attestation. Measurement is the act of obtaining cryptographic representations for the system state. Attestation is the act of comparing those cryptographic measurements against expected values to determine whether the system booted into an acceptable state.

Attestation can be performed either locally, on the same host that is to be attested, or remotely, by an external authority. The trusted boot process can optionally include a local attestation involving the evaluation of a TPM-stored Launch Control Policy (LCP). In this case, the host’s TPM will compare the measurements that have been taken so far to a set of expected PCR values stored in the LCP; if there is a mismatch, the boot process is halted entirely.

Intel® SecL utilizes remote attestation, providing a remote Verification Service that maintains a database of expected measurements (or “flavors”), and compares the actual boot-time measurements from any number of hosts against its database to provide an assertion that the host booted into a “trusted” or “untrusted” state. Remote attestation is typically easier to centrally manage (as opposed to creating an LCP for each host and entering the policy into the host’s TPM), does not halt the boot process allowing for easier remediation, and separates the attack surface into separate components that must both be compromised to bypass security controls.
Both local and remote attestation can be used concurrently. However, Intel®
SecL, and this document, will focus only on remote attestation. For more
information on TPM Launch Control Policies, consult the Intel Trusted Execution
Technology (Intel TXT) Software Development Guide

1.1.1.5 Platform Integrity

Platform Integrity is the use case enabled by the specific implementation of the
Chain of Trust and Remote Attestation concepts. This involves the use of a
Root of Trust to begin an unbroken chain of platform measurements at server
boot time, with measurements extended to the Trusted Platform Module and
compared against expected values to verify the integrity of measured
components. This use case is foundational for other Intel® SecL use cases.

1.1.1.6 Data Sovereignty

Data Sovereignty builds on the Platform Integrity use case to allow physical
TPMs to be written with Asset Tags containing any number of key/value pairs.
This use case is typically used to identify the geographic location of the
physical server, but can also be used to identify other attributes. For example,
the Asset Tags provided by the Data Sovereignty use case could be used to
identify hosts that meet specific compliance requirements and can run
controlled workloads.

1.1.1.7 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 Intel® Security Libraries

Components

2.1 Trust Agent

The Trust Agent resides on physical servers and enables both remote attestation and the extended chain of trust capabilities. The Agent maintains ownership of the server’s Trusted Platform Module, allowing secure attestation quotes to be sent to the Verification Service. Incorporating the Intel® SecL HostInfo and TpmProvider libraries, the Trust Agent serves to report on platform security capabilities and platform integrity measurements.

The Trust Agent is supported for Windows* Server 2016 Datacenter and Red Hat Enterprise Linux* (RHEL) 7.4 and later.

2.2 Verification Service

The Verification Service component of Intel® Security Libraries performs the core Platform Integrity and Data Sovereignty functionality by acting as a remote attestation authority.

Platform security technologies like Intel® TXT, Intel® BootGuard, and UEFI SecureBoot extend measurements of platform components (such as the system BIOS/UEFI, OS kernel, etc) to a Trusted Platform module as the server boots. Known-good measurements for each of these components can be directly imported from a sample server. These expected measurements can then be compared against actual measurements from registered servers, allowing the Verification Service to attest to the “trustiness” of the platform, meaning whether the platform booted into a “known-good” state.

2.3 Integration Hub

The Integration Hub acts as a middle-man between theVerification Service and one or more scheduler services (such as OpenStack* Nova), and "pushes" attestation information retrieved from the Verification Service to one or more scheduler services according to an assignment of hosts to specific tenants. In this way, Tenant A can receive attestation information for hosts that belong to Tenant A, but receive no information about hosts belonging to Tenant B.

The Integration Hub serves to disassociate the process of retrieving attestations from actual scheduler queries, so that scheduler services can adhere to best practices and retain better performance at scale. The Integration Hub will regularly query the Intel® SecL Verification Service for SAML attestations for each host. The Integration Hub maintains only the most
recent currently valid attestation for each host, and will refresh attestations when they would expire. The Integration Hub will verify the signature of the SAML attestation for each host assigned to a tenant, then parse the attestation status and asset tag information, and then will securely push the parsed key/value pairs to the plugin endpoints enabled.

The Integration Hub features a plugin design for adding new scheduler endpoint types. Currently the Integration Hub supports an OpenStack Nova endpoint plugin. Other integration plugins may be added.
3 Intel® Security Libraries Installation

3.1 Building from Source

Intel® Security Libraries is distributed as open source code, and must be compiled into installation binaries before installation.

Instructions and sample scripts for building the Intel® SecL-DC components can be found here:

https://01.org/intel-secl/documentation/build-installation-scripts

After the components have been built, the installation binaries can be found in the directories created by the build scripts:

<servicename>/packages/<servicename>/target/<servicename>-<version>-SNAPSHOT.bin

3.2 Recommended Service Layout

The Intel® SecL-DC services can be installed in a variety of layouts, partially depending on the use cases desired and the OS of the server(s) to be protected. In general, the Intel® SecL-DC applications can be divided into management services that are deployed on the network on the management plane, and host or node components that must be installed on each protected server.

Management services can typically be deployed anywhere with network access to all of the protected servers. This could be a set of individual VMs per service; containers; or all installed on a single physical or virtual machine.

Node components must be installed on each protected physical server. Typically this is needed for Windows and Linux deployments.

3.2.1 Platform Integrity

The most basic use case enabled by Intel® SecL-DC, Platform Integrity requires only the Verification Service and, to protect Windows or Linux hosts, the Trust Agent. This also enables the Application Integrity use case by default for Linux systems.

The Integration Hub may be added to provide integration support for OpenStack or Kubernetes. The Hub is often installed on the same machine as the Verification Service, but optionally can be installed separately.
3.3 Installing the Verification Service

This section details how to install the Intel® SecL-DC services. For instructions on running these services as containers, see the following section.

3.3.1 Package Dependencies

The Intel® Security Libraries Verification Service requires the following repositories:

- The Postgresql 9.4 package repository must be available. Add the repository appropriate for the distribution of Linux that will run the Verification Service from the link below:
  https://yum.postgresql.org/repopackages.php#pg94
- (For RHEL installation) The EPEL repository must be available (add by installing the “epel-release” package)
- The standard repositories for the chosen Linux distribution must be available

3.3.2 Supported Operating Systems

The Intel® Security Libraries Verification Service supports the following Linux distributions and versions:

- Red Hat Enterprise Linux 7.4 and later
- Fedora 29 and later

3.3.3 Recommended Hardware

- 4 vCPUs
- RAM: 8 GB
- 100 GB
- One network interface with network access to all managed servers
- (Optional) One network interface for Asset Tag provisioning (only required for “pull” tag provisioning; required to provision Asset Tags to VMware ESXi servers).
### 3.3.4 Installation

To install the Verification Service, follow these steps:

1. Copy the Verification Service RPM package to the `/root/` directory.
2. Create the `mtwilson.env` installation answer file for an unattended installation.

A sample minimal `mtwilson.env` file is provided below. For all configuration options and their descriptions, refer to the Intel® SecL Configuration section on the Verification Service.

```bash
### User credentials
export MC_FIRST_USERNAME=<Administrator username>
export MC_FIRST_PASSWORD=<Administrator password>

### Database Configuration
export DATABASE_USERNAME=<Database User>
export DATABASE_PASSWORD=<Database Password>

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

Install using the package manager or rpm utility:

```bash
dnf localinstall host-verification-service-4.5-1.el7.x86_64.rpm
```
or

```bash
rpm -i host-verification-service-4.5-1.el7.x86_64.rpm
```

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

```
# mtwilson status
```

### 3.4 Installing the Trust Agent for Linux

#### 3.4.1 Package Dependencies

The Trust Agent requires the following repositories or a suitable mirror:

- (For RHEL installation) The EPEL repository must be available (add by installing the “epel-release” package)
- The standard repositories for the chosen Linux distribution must be available
3.4.1.1 Tboot

- Tboot 1.9.7 (Optional, required for TXT-based deployments **without** UEFI SecureBoot only)

Tboot is required for TXT-enabled systems without UEFI SecureBoot enabled. Tboot executes the Dynamic Root of Trust measurements on a TXT-enabled platform – this includes measurements above the BIOS/UEFI like the Linux kernel, the grub commandline arguments, and initrd. Tboot requires reconfiguration of the grub bootloader, and so is not installed as part of the Trust Agent RPM; it must be installed as a prerequisite.

Tboot should not be installed if UEFI SecureBoot enabled on the platform, or if TXT is disabled.

**Note:** For Fedora installations of the Trust Agent, using tboot requires using Legacy BIOS boot mode. For RHEL installations, both Legacy BIOS and UEFI boot modes are supported (without UEFI Secure Boot) for tboot.

3.4.2 Supported Operating Systems

The Intel® Security Libraries Trust Agent for Linux supports the following Linux distributions and versions:

- Red Hat Enterprise Linux 7.4 and later
- Fedora 29 and later

Windows support is described in the section “Installing the Trust Agent for Windows.”

3.4.3 Creating Service Users

After installation is complete, users should be created for any services that will be integrated with the Verification Service (such as the Trust Agent or Integration Hub).

The administrative user created during installation (MC_FIRST_USERNAME) has full administrative privileges and can be used to all services and requests, but this is strongly not recommended for security reasons.

(Optional, required only if Trust Agent hosts will be registered.)

```bash
mtwilson login-password <username> <password> --permissions host_aliks:certify tpm_endorsements:create tpm_endorsements:search tpm_passwords:create tpm_passwords:retrieve tpm_passwords:search tpm_passwords:store tpm:endorse host_signing_key_certificates:create store_host_pre_registration_details:create
```

To also allow automatic registration and HOST_UNIQUE flavor imports during Trust Agent installation, add the following permissions:
3.4.4 Prerequisites

The following must be completed before installing the Trust Agent:

- Supported server hardware including an Intel® Xeon® processor with Intel Trusted Execution Technology activated in the system BIOS.
- Trusted Platform Module (version 2.0) installed and activated in the system BIOS, with cleared ownership status.
  
  **Note:** For Linux systems, TPM 1.2 and TPM resource sharing to applications other than the Trust Agent are not supported at this time. Do not install trousers or another TSS stack application after installing the Trust Agent on Linux systems.
- System must be booted to a boot boot option (for TXT-enabled systems only without UEFI Secure Boot enabled; Trust Agent installation will automatically install tboot if not present and if UEFI Secure Boot is not enabled, and then require a reboot before proceeding).

  **Note:** While both UEFI and Legacy Bios mode are supported, tboot is currently not supported when using UEFI Secure Boot. If UEFI Secure Boot is enabled while attempting to boot to tboot, the boot process will halt. UEFI Secure Boot must be disabled for Linux systems when using tboot. If UEFI Secure Boot is enabled, the Trust Agent installer will skip the installation of tboot and proceed to install normally.
- (Provisioning step only) Intel® SecL Verification Service server installed and active.

3.4.4.1 Installing Tboot

If tboot will be used (required if UEFI Secure Boot is not used), the tboot package must be installed before executing the Trust Agent installation. Tboot version 1.9.7 or later should be used. When tboot is detected during the Trust Agent installation, the Trust Agent will automatically reconfigure the Grub2 boot loader with a new tboot boot option.

dnf install tboot

Commented [KTE3]: Need to add instructions here, depending on how this is semi-automated.
Note: Tboot is required only for TXT-enabled platforms where UEFI SecureBoot is not enabled. Do not use Tboot when UEFI SecureBoot is enabled, as a known incompatibility exists that will cause the system to not boot.

If installing on Fedora, be sure to configure the server to boot in legacy BIOS mode if tboot will be used. For RHEL installations, both Legacy BIOS and UEFI boot modes are supported for tboot (with UEFI Secure Boot disabled).

### 3.4.5 Installation

Installation of the Trust Agent is split into two major steps: Installation, which covers the creation of system files and folders, and Provisioning, which involves the creation of keys and secrets and links the Trust Agent to a specific Verification Service.

To install the Trust Agent for Linux:

1. Create the trustagent.env answer file in the /root/ directory (for full configuration options, see section 9.2). The minimum configuration options for installation are provided below.

```
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=<Verification Service PrivacyCA username>
MTWILSON_API_PASSWORD=<Verification Service PrivacyCA password>
REGISTER_TPM_PASSWORD=y
TRUSTAGENT_LOGIN_REGISTER=true
GRUB_FILE=<path to grub.cfg>
CURRENT_IP=<Trust Agent IP address>
```

Note: The MTWILSON_API_USERNAME and password required by the Trust Agent can be satisfied by the PRIVACYCA_DOWNLOAD_USERNAME user created during the installation of the Verification Service, or by any user with the following Verification Service permissions:

- host_aiks:certify
- tpm_endorsements:create
- tpm_endorsements:search
- tpm_passwords:create
- tpm_passwords:retrieve
- tpm_passwords:search
- tpm_passwords:store
- tpm:endorse
- host_signing_key_certificates:create
- store_host_pre_registration_details:create

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

```
cat /etc/host-verification-service/https.properties
```

from the Verification Service command line.

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

3. Install the Trust Agent using the Yum package manager or the RPM utility, and wait for the installation to complete.
dnf localinstall trustagent-4.5-1.el7.x86_64.rpm
Or
rpm -ifv trustagent-4.5-1.el7.x86_64.rpm

4. Once installation is completed, complete the provisioning of the Trust Agent by executing the following command:
   tagent provision-attestation trustagent.env

5. After Provisioning is completed, the Linux Trust Agent must be rebooted so that the default SOFTWARE Flavor manifest can be measured and extended to the TPM.

3.5 Installing the Trust Agent for Windows

3.5.1 Supported Operating Systems

The Trust Agent for Windows supports Windows Server 2016 Datacenter.

3.5.2 Prerequisites

The following must be completed before installing the Trust Agent:
- Supported server hardware including an Intel® Xeon® processor.
- Trusted Platform Module (version 1.2 or 2.0) installed and activated in the system BIOS, with cleared ownership status.
- Coreinfo (https://docs.microsoft.com/en-us/sysinternals/downloads/coreinfo) must be installed
- (Provisioning step only) Intel® SecL Verification Service server installed and active.

3.5.2.1 TPM Ownership

The Intel® SecL-DC Trust Agent for Windows requires the TPM ownership secret to be stored in the local system registry. To confirm that the secret is populated in the registry:

1) Open a Command Prompt as Administrator
2) Run the following command:
   REG_QUERY hkm\system\controlset001\services\tpm\wmi\admin
3) If the output contains the OwnerAuthFull key and a corresponding value, the ownership secret is present in the registry and no further action is needed.
If the output does not contain the secret, system must be configured to store the secret in the registry.

To configure GPO to store the ownership secret in the local registry:

1) Open a Command Prompt as an Administrator
2) Run gpedit.msc
3) In the GP Editor, browse to Computer Configuration\Administrative Templates\System\Trusted Platform Module Services\
4) Set the Operating System Managed TPM Authentication Level to "Full"
5) Clear the TPM ownership and reboot
   a) Open a Command Prompt as Administrator
   b) Run tpm.msc
   c) From the TPM Management Console that appears, click “Clear TPM”
   d) After the process is complete, reboot

3.5.3 Installation

Installation of the Trust Agent is split into two major steps: Installation, which covers the creation of system files and folders, and Provisioning, which involves the creation of keys and secrets and links the Trust Agent to a specific Verification Service. Both operations can be performed at the same time using an installation answer file. Without the answer file, the Trust Agent can be installed and left in an un-provisioned state regardless of whether a Verification Service is up and running, until such time as the datacenter administrator is ready to run the provisioning step and link the Trust Agent to a Verification Service.

To install the Trust Agent for Windows:

1. (Optional; required to perform Provisioning and Installation at the same time.) Create the trustagent.ini answer file in the C:\Temp directory (for full configuration options, see section 9.2). The minimum configuration options for installation are provided below.
Note: The MTWILSON_API_USERNAME and password required by the Trust Agent requires a Verification Service user with the "trustagent_provisioner" role. See the Verification Service installation subsection "Creating Service Users" for details on user creation.

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

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

from the Verification Service command line.

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.

If the trustagent.ini answer file was provided with the minimum required options, the Trust Agent will be installed and also Provisioned to the Verification Service specified in the answer file.

If no answer file was provided, the Trust Agent will be installed, but will not be Provisioned. TPM-related functionality will not be available from the Trust Agent until the Provisioning step is completed.

### 3.6 Trust Agent Provisioning

"Provisioning" the Trust Agent involves connecting to a Verification Service to download the Verification Service PrivacyCA certificate, create a new Attestation Identity Keypair in the TPM, and verify or create the TPM Endorsement Certificate and Endorsement Key. The Verification Service PrivacyCA root certificate is used to sign the EC, and the EC is used to generate the Attestation Identity Keypair. The AIK is used by the Verification Service to verify the integrity of quotes from the host’s TPM.

Provisioning can be performed separately from installation (meaning you can install the Trust Agent without Provisioning, and then Provision later). If the trustagent.env answer file is present and has the required Verification Service information during installation, the Agent will automatically run the Provisioning steps.
**Note:** The `trustagent.env` answer file must contain user credentials for a user with sufficient privileges. The minimum role required for performing provisioning is the "trustagent_provisioner" role.

**Note:** If the Linux Trust Agent is installed without being Provisioned, the Trust Agent process will not actually run until the Provisioning step has been completed.

If the answer file is not present during installation, the Agent can be Provisioned later by adding the `trustagent.env` file and running the following command:

```bash
tagent provision-attestation <trustagent.env or trustagent.ini file path>
```

### 3.7 Trust Agent Registration

Registration creates a host record with connectivity details and other host information 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.

**Note:** Host registration requires user credentials for a user with sufficient privileges. The minimum role required for performing host registration is the "host_manager" role.

The Trust Agent can register the host with a Verification Service by running the following command (the `trustagent.env` or `trustagent.ini` answer file must be present in the current working directory):

```bash
tagent create-host
```

Hosts can also be registered using a REST API request to the Verification Service:

```json
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": "intel:https://<hostname or IP address>:1443;u=<TAgent username>;p=<TAgent password>",
    "flavorgroup_name" : "",
    "description" : "<description>"
}
```

**Note:** When a new host is registered, the Verification Service will automatically attempt to match the host to appropriate Flavors. If appropriate Flavors are not found, the host will still be registered, but will be in an Untrusted state until/unless appropriate Flavors are added to the Verification Service.
3.8 Importing the HOST_UNIQUE Flavor

RHEL and VMWare ESXi hosts have measured components that are unique to each host. This means that a special HOST_UNIQUE flavor part needs to be imported for each RHEL and ESXi host, in addition to any other OS or Platform Flavors.

**Note:** Importing a Flavor requires user credentials for a user with sufficient privileges. The minimum role required for creating the HOST_UNIQUE Flavor part is the "host_unique_flavor_creator" role. This role can only create HOST_UNIQUE Flavor parts, and cannot create any other Flavors.

On Red Hat Enterprise Linux hosts with the Trust Agent, this can be performed from the Trust Agent command line (this requires the trustagent.env answer file to be present in the current working directory):

```
tagent create-host-unique-flavor
```

This can also be performed using a REST API (required for VMWare ESXi hosts):
3.9 Installing the Intel® SecL Integration Hub

**Note:** The Integration Hub is only required to integrate Intel® SecL with third-party scheduler services, such as OpenStack Nova or Kubernetes. The Hub is not required for usage models that do not require Intel® SecL security attributes to be pushed to an integration endpoint.

3.9.1 Package Dependencies

The Intel® SecL Integration Hub requires the following package repositories or a suitable mirror:

- The Postgresql 9.4 package repository must be available. Add the repository appropriate for the distribution of Linux that will run the Verification Service from the link below:
  https://yum.postgresql.org/repopackages.php#pg94
- (For RHEL installation) The EPEL repository must be available (add by installing the "epel-release" package)
- The standard repositories for the chosen Linux distribution must be available

3.9.2 Supported Operating Systems

The Intel Security Libraries Integration Hub supports:

- Red Hat Enterprise Linux 7.4 and later
- Fedora 29 and later

3.9.3 Prerequisites

The Intel® Security Libraries Integration Hub can be run as a VM or as a bare-metal server. The Hub may be installed on the same server (physical or VM) as the Verification Service.

3.9.4 Recommended Hardware

- 1 vCPUs
- RAM: 2 GB

Commented [KTES]: Even for remote DB installs? Sloppy if so.
• 1 GB free space to install the Verification Service services. Additional free space is needed for the Attestation Hub database and logs (database and log space requirements are dependent on the number of managed servers).
• One network interface with network access to the Verification Service.
• One network interface with network access to any integration endpoints (for example, OpenStack Nova).

3.9.5 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. See the sample file below.

```
ATTESTATION_HUB_PORT_HTTP=19082
ATTESTATION_HUB_PORT_HTTPS=19445

MTWILSON_API_URL="https://<Verification Service IP or hostname>:8443/mtwilson/v2"
MTWILSON_TLS=<sha384 of Verification Service TLS certificate>
MTWILSON_USERNAME=<username of Verification Service account with attestations:retrieve permissions>
MTWILSON_PASSWORD=<password for Verification Service user account>

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_POLL_INTERVAL=2
ATTESTATION_HUB_TENANT_CONFIGURATIONS_PATH=/opt/tenantconfig

ATTESTATION_HUB_DB_USERNAME=<Database administrative username>
ATTESTATION_HUB_DB_PASSWORD=<Database password>
```

**Note:** The MTWILSON_TLS value can be retrieved from the Verification Service using the command:
```
cat /etc/host-verification-service/https.properties
```
from the Verification Service command line.
3. Install the Integration Hub application
   ```
dnf localinstall attestation-hub-4.5-1.el7.x86_64.rpm
   Or
   rpm -i attestation-hub-4.5-1.el7.x86_64.rpm
   ```
4. Create an administrative user for the Integration hub. On the command line while logged in as root, run the following:

```
attestation-hub password admin password --permissions ",*"
```

This will create a new user named "admin" with password "password" that will be used to make REST API requests to the Hub.
4 Deployment Using Containers

The Intel® SecL-DC services can also be deployed as containers. There are a few important notes to consider:

- Each physical Linux server must run a dedicated Trust Agent container. This container must run in privileged mode so that it has access to host resources, including the physical TPM. The storage volume for the Trust Agent must stay with the host; the Agent and its configuration files are intrinsically linked to the physical TPM on the host when the initial installation happens.
- The Verification Service and Integration Hub services can be moved and run anywhere with no hardware restrictions.
- Several of the application commands (<servicename> start/stop/restart, uninstall, etc) are not supported in the containerized deployment. To restart services, simply stop the container and then start it again. To uninstall, simply delete the configuration volume.

The Intel® SecL-DC services are not distributed as ready-made containers, so deployment includes building the initial container image. To make this process easier, Intel distributes premade docker-compose and dockerfile files. These files can be modified for environment-specific settings.

4.1 Building the ISecL Service Docker Images

4.1.1 Prerequisites for Building Docker Images

- Some packages may require packages not available on the regular RHEL repositories. Install the following repos:
  - RDO ([https://download.docker.com/linux/centos/docker-ce.repo](https://download.docker.com/linux/centos/docker-ce.repo))
  - Docker-CE ([https://download.docker.com/linux/centos/docker-ce.repo](https://download.docker.com/linux/centos/docker-ce.repo))
- Docker-ce 1.18 must be installed (from [https://download.docker.com/linux/centos/docker-ce.repo](https://download.docker.com/linux/centos/docker-ce.repo))
- Docker-compose 1.23.1 or later must be installed (this version may not be available on the RHEL repositories; use PIP for installation of the supported version)
- Container-selinux must be installed
- The Docker daemon must be installed and running
- Locations must be identified for persistent data volumes for container configuration files and the database
- (Optional) An external Postgres 9.4 database server may be used. The provided files will automatically configure a Postgres 9.4 Docker container by default.

4.1.2 Verification Service

To build the Verification Service Docker image:

1) Download and extract the host-verification-service-docker.zip archive to /root/ on a Docker host server (this may be a physical or virtual server with the prerequisites installed)
2) If custom repositories will be needed, place the .repo definition files in the extracted yum.repos.d/ folder.
3) Run docker-compose to build the image. Use build arguments to set proxies if proxies are necessary for your network.

    docker-compose build --build-arg http_proxy=<http_proxy_server> \ 
    --build-arg https_proxy=<https_proxy_server> \ 
    --build-arg no_proxy=<repo_and_other_local_servers> \ 
    verification-service

After the build process completes, the image should appear when the “docker images” command is executed:

    # docker images
    
    REPOSITORY           TAG       IMAGE ID       CREATED        SIZE
    isecl-verification-service latest f8ca23672bf9 20 hours ago  2.26GB
The “Dockerfile” file contains the definition for the initial image creation. The build process will use the service installation binary to partially install the Verification Service, without generation of any deployment-specific secrets or keys. The remaining installation tasks will be performed when the container is launched the first time, and a persistent storage volume will be used to maintain the configuration when the container is restarted.

4.1.3 Integration Hub

To build the Integration Hub Docker image:

4) Download and extract the attestation-hub-docker.zip archive to /root/ on a Docker host server (this may be a physical or virtual server with the prerequisites installed)
5) If custom repositories will be needed, place the .repo definition files in the extracted yum.repos.d/ folder.
6) Run docker-compose to build the image. Use build arguments to set proxies if proxies are necessary for your network.

```bash
docker-compose build --build-arg http_proxy=<http_proxy_server> \ 
--build-arg https_proxy=<https_proxy_server> \ 
--build-arg no_proxy=<repo_and_other_local_servers> \ 
attestation-hub
```

After the build process completes, the image should appear when the “docker images” command is executed:

```bash
# docker images
REPOSITORY                         TAG                 IMAGE
isecl-attestation-hub              latest             1a9dd28e97e3
```

The “Dockerfile” file contains the definition for the initial image creation. The build process will use the service installation binary to partially install the
4.1.4 Trust Agent

To build the Trust Agent Docker image:

7) Download and extract the `trustagent-linux-docker.zip` archive to `/root/` on a Docker host server (this may be a physical or virtual server with the prerequisites installed)
8) If custom repositories will be needed, place the `.repo` definition files in the extracted `yum.repos.d/` folder.
9) Run `docker-compose` to build the image. Use build arguments to set proxies if proxies are necessary for your network.

```bash
docker-compose build --build-arg http_proxy=<http_proxy_server> \
--build-arg https_proxy=<https_proxy_server> \
--build-arg no_proxy=<repo_and_other_local_servers> \
iseclv1-agent
```

After the build process completes, the image should appear when the "docker images" command is executed:

```
# docker images
REPOSITORY                         TAG                 IMAGE
isecl-trustagent                   latest              9ec70e88e186
```

The "Dockerfile" file contains the definition for the initial image creation. The build process will use the service installation binary to partially install the Trust Agent, without generation of any deployment-specific secrets or keys. The remaining installation tasks will be performed when the container is launched the first time, and a persistent storage volume will be used to maintain the configuration when the container is restarted.
4.2 Deploying the Intel® SecL Service Containers

4.2.1 Verification Service

When the Verification Service container is first run, it will perform setup and configuration tasks. By default, this includes using a Postgres database container that will automatically be created.

The container uses several persistent storage volumes that contain configuration files, logs, secrets, and keys. If the Verification Service is restarted, the configuration in the storage volumes will be retained so that the initial setup does not need to be repeated, and so that secrets and keys do not change on container restart.

4.2.1.1 Container Configuration

Configuration settings can be changed by modifying the following files:

mtwilson.env

At minimum, this must define the Admin Username and password. This defines the initial administrative user created during installation. Additional variables can be set here as per the same mtwilson.env answer file options supported for the non-container based installer.

```
MC_FIRST_USERNAME=<Admin user>
MC_FIRST_PASSWORD=<Admin password>
```

docker-compose.yml

This file defines the container configuration for the Verification Service, and additionally a Postgres database container for the Verification Service to use. An external Postgres 9.4 database server may optionally be used instead of the container, in which case the Postgres portion can be commented out of this file. Be sure to set the appropriate database options in the Verification Service environment definition in this file if an alternative database server will be used.

This file can be modified to change configuration options for the container, but can be used as-is for a default installation. Be sure that the container hostname is resolvable from all Trust Agent nodes.
Update the file to set the Postgres password. This will be used by the Postgres database server, and also will be used by the Verification Service to connect to the database server.
4.2.1.2 Persistence Volume Creation

The Verification Service container uses the following persistence volumes:

- hvs-config-volume
- hvs-logs-volume

Additionally, by default the Verification Service `docker-compose.yml` defines a separate Postgres database container for the Verification Service to use. This database container requires a storage volume to retain database information.

- hvs-pgdata-volume

These volumes are created automatically from the definitions in the `docker-compose.yml` file. To change the path where these volumes will be stored, edit the `docker-compose` file.

4.2.1.3 Starting the Container

To start the Verification Service and associated Postgres database containers, run the following:

```
docker-compose up --remove-orphans --abort-on-container-exit
```

Once the container is down/stopped (`docker-compose down`), the containers no longer need the `.env` files and `db.password` to restart, since the setup activities are done during the first launch and the configurations are stored as part of the volumes.

The container status can be confirmed using the “docker ps” command:

```
# docker ps
CONTAINER ID        IMAGE                               COMMAND
```
To stop the containers, use the following:

docker-compose down

4.2.2 Integration Hub

When the Integration Hub container is first run, it will perform setup and configuration tasks. By default, this includes using a Postgres database container that will automatically be created.

The container uses several persistent storage volumes that contain configuration files, logs, secrets, and keys. If the Hub is restarted, the configuration in the storage volumes will be retained so that the initial setup does not need to be repeated, and so that secrets and keys do not change on container restart.

4.2.2.1 Prerequisites
- The Verification Service must be installed and running

4.2.2.2 Container Configuration

Configuration settings can be changed by modifying the following files:

attestation-hub.env

At minimum, this must define the connection details for the Verification Service, and the Integration Hub initial administrator user credentials, which will be created during setup. Additional variables can be set here as per the same attestation-hub.env answer file options supported for the non-container based installer.

MTWILSON_HOSTNAME=verification-service
MTWILSON_USERNAME=username
MTWILSON_PASSWORD=password
MTWILSON_TLS_CERT_SHA384=2c8b3d29bd64de927fc66568eee67c8fb36895c440f8455c19c1b2f9949d25b8f6656efc4a47217da3c43d28ddc5f9a

AHUB_USER_NAME=username

AHUB_USER_PASSWORD=password

docker-compose.yml

This file defines the container configuration for the Integration Hub, and additionally a Postgres database container for the Integration Hub to use. An external Postgres 9.4 database server may optionally be used instead of the container, in which case the Postgres portion can be commented out of this file. Be sure to set the appropriate database options in the Integration Hub environment definition in this file if an alternative database server will be used.

This file can be modified to change configuration options for the container, but can be used as-is for a default installation. Be sure that the MTWILSON_HOSTNAME value is resolvable to the Verification Service from the container.
Update the file to set the Postgres password. This will be used by the Postgres database server, and also will be used by the Integration Hub to connect to the database server.
4.2.2.3 Persistence Volume Creation

The Verification Service container uses the following persistence volumes:

- ahub-pgdata-volume:
- ahub-config-volume:
- ahub-logs-volume:
- ahub-tenant-config-volume:

Additionally, by default the Integration Hub `docker-compose.yml` defines a separate Postgres database container for the Integration Hub to use. This database container requires a storage volume to retain database information.

- ahub-pgdata-volume

These volumes are created automatically from the definitions in the `docker-compose.yml` file. To change the path where these volumes will be stored, edit the `docker-compose.yml` file.

4.2.2.4 Starting the Container

To start the Integration Hub and associated Postgres database containers, run the following:

```
docker-compose up --remove-orphans --abort-on-container-exit
```

Once the container is down/stopped (`docker-compose down`), the containers no longer need the `.env` files and `db.password` to restart, since the setup activities are done during the first launch and the configurations are stored as part of the volumes.

The container status can be confirmed using the “docker ps” command:

```
# docker ps
CONTAINER ID        IMAGE
```

To stop the containers, use the following:
4.2.3 Trust Agent

When the Trust Agent container is first run, it will perform setup and configuration tasks. This includes taking ownership of the TPM, generating a new AIK, and creation of other secrets and keys. After this setup is performed, the persistent data in the storage volume will be intrinsically tied to the physical TPM of the host, and cannot be used on any other host. Note that this requires that the Trust Agent also be run in privileged mode.

The container uses several persistent storage volumes that contain configuration files, logs, secrets, and keys. If the Trust Agent is restarted, the configuration in the storage volumes will be retained so that the initial setup does not need to be repeated, and so that secrets and keys do not change on container restart.

4.2.3.1 Prerequisites

- The Verification Service must be installed and running
- TPM ownership must be clear and the TPM must be activated
- Tboot must be installed on the host, and the host must be booted to a tboot boot option (TXT-enabled deployments with UEFI SecureBoot not enabled only; if UEFI SecureBoot is enabled, or if TXT is not enabled, tboot must not be installed)

4.2.3.2 Container Configuration

Configuration settings can be changed by modifying the following files:

```
trustagent.env
```

At minimum, this must define the connectivity details for the Verification Service. Additional variables can be set here as per the same `attestation-hub.env` answer file options supported for the non-container based installer. Using the `PROVISION_ATTESTATION=yes` option and the `TRUSTAGENT_LOGIN_REGISTER=yes` option are strongly recommended.

MTWILSON_HOSTNAME=verification-service

MTWILSON_API_USERNAME=admin
This file defines the container configuration for the Trust Agent.

This file can be modified to change configuration options for the container, but can be used as-is for a default installation. Be sure that the container hostname is resolvable and unique for each host – this will define the hostname seen in the Verification Service after registration.

```
version: "3.1"
services:
  isecv1-agent:
    image: isec-trustagent:latest
    build: .
    hostname: trustagent
    network_mode: host
    privileged: true
    environment:
      - TRUSTAGENT_DOCKER_HOST_MOUNT=/root/host_root
    secrets:
      - source: trustagent_env
        target: trustagent.env
    volumes:
      - tagent-config-volume:/opt/trustagent/configuration
      - tagent-logs-volume:/opt/trustagent/logs
      - /var/lib/tpm:/var/lib/tpm
      - /:/root/host_root
    extra_hosts:
      - "verification-service:<hostname or IP address>"
    volumes:
      tagent-config-volume:
      tagent-logs-volume:
    secrets:
      trustagent_env:
        file: trustagent.env
```
4.2.3.3 Persistence Volume Creation

The Verification Service container uses the following persistence volumes:

tagent-config-volume:/opt/trustagent/configuration
tagent-logs-volume:/opt/trustagent/logs
/var/lib/tpm
/:/root/host_root

These volumes are created automatically from the definitions in the docker-compose.yml file. To change the path where these volumes will be stored, edit the docker-compose file. Note that for the Trust Agent specifically, the / and /var/lib/tpm volumes must not be changed, as they allow the container access to the TPM and host information required for trust attestation.

4.2.3.4 Starting the Container

To start the Trust Agent container, run the following:

docker-compose up --remove-orphans --abort-on-container-exit

Once the container is down/stopped (docker-compose down), the containers no longer need the .env files and db.password to restart, since the setup activities are done during the first launch and the configurations are stored as part of the volumes.

The container status can be confirmed using the "docker ps" command:

```
# docker ps

CONTAINER ID        IMAGE
582bfd3da5ee        isecl-trustagent:latest
```

To stop the containers, use the following:

docker-compose down
5 **Connection Strings**

Connection Strings define a remote API resource endpoint that will be used to communicate with the registered host for retrieving TPM quotes. Connection Strings differ based on the type of host.

5.1 **Trust Agent (Windows and Linux)**

The Trust Agent connection string connects directly to the Trust Agent on a given host. If the Trust Agent is installed or provisioned with the option to pre-register the Trust Agent credentials in the Verification Service (TRUSTAGENT_LOGIN_REGISTER=true), no authentication details are needed; the Verification Service will use the credentials already stored in its database to make the connection. If the credential pre-registration option was not selected, the credentials for the Trust Agent will need to be provided in the connection string.

```
intel:https://<HostNameOrIp>:1443
```

If the "TRUSTAGENT_LOGIN_REGISTER" option was not used during installation, or if it was set to "false," credentials will need to be supplied as part of the connection string:

```
intel:https://<HostNameOrIp>:1443;u=<username>;p=<password>
```

5.2 **VMware ESXi**

The VMware ESXi connection string is actually directed to vCenter, not the actual ESXi host. Many ESXi hosts managed by the same vCenter server will use the same connection string. The username and password specified are vCenter credentials, and the vCenter "Validate Session" privilege is required for access.

```
vmware:https://<vCenterHostNameOrIp>:443/sdk;h=<hostname of ESXi host>;u=<username>;p=<password>
```
Platform Attestation is the cornerstone use case for ISecl. Platform attestation involves taking measurements of system components during system boot, and then cryptographically verifying that the actual measurements taken matched 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.

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

This section includes basic REST API examples for these workflows. See the Javadoc for more detailed documentation on REST APIs supported by ISecl.

Typical workflows in the datacenter might include:
- Creating a set of acceptable flavors for attestation with automatic flavor matching that represent the known-good measurements for acceptable BIOS and OS versions in the datacenter
- Registering hosts for attestation with automatic flavor matching
- Upgrading hosts in the datacenter to a new BIOS or OS version
- Removing hosts from the Verification Service
- Removing flavors
- Provisioning asset tags to hosts
- Invalidating asset tags
- Retrieving current attestation reports
- Retrieving current host state information
- Remediating an untrusted attestation
6.1 Host Registration

Registration creates a host record with connectivity details and other host information 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.

6.1.1 Trust Agent

6.1.1.1 Registration via Trust Agent Command Line

The Trust Agent can register the host with a Verification Service by running the following command:

tagent create-host <Verification Service base URL> <username> <password>

Note: Because VMWare ESXi hosts do not use a Trust Agent, this method is not applicable for registration of ESXi hosts.

6.1.2 Registration via Verification Service API

Any Trust Agent or VMware ESXi host can be registered using a Verification Service API request. Registration can be performed with or without a set of existing Flavors. Rules for Flavor matching can be set by using the Flavor Group in the request; if no Flavor Group is specified, the "mtwilson_automatic" Flavor Group will be used. See the Flavor Management section for additional details on Flavors, Flavor Groups, and Flavor matching.

6.1.2.1 Sample 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": "<connection string>",  
  "flavorgroup_name": "",  
  "description": "<description>"  
}

Requires the permission "hosts:create"
6.2 Flavor Creation for Automatic Flavor Matching

Flavor creation is the process of adding one or more sets of acceptable measurements to the Verification Service database. These measurements correspond to specific system components, and are used as the basis of comparison to generate trust attestations.

Flavors can be created manually, or can be imported from an example host.

Flavors are automatically matched to hosts based on the Flavorgroup used by the host and the Flavors, and the Flavor Match Policies of the Flavorgroup. The ISecL Verification Service creates a default Flavorgroups during installation called “automatic.” This Flavorgroup is configured to be used as a pool of all acceptable Flavors in a given environment, and will automatically match the appropriate Flavor parts to the correct host. This Flavorgroup is used by default and is expected to be useful for the majority of deployments. If no Flavorgroup is specified when creating a Flavor, it will be placed in the “automatic” Flavorgroup.

Flavors are also divided into Flavor parts, which correspond to the PLATFORM, OS, HOST_UNIQUE, SOFTWARE, and ASSET_TAG measurements. These can be created and maintained separately (so that users can manage acceptable OS and BIOS versions, rather than entire host configurations). By default, if not specified, the Verification Service will import Flavors as separate Flavor parts, as appropriate for the host type.

By using individual Flavor parts, individual versions of OS or PLATFORM measurements can be managed and automatically mapped. Whenever a host changes states (Untrusted, Connected, etc.) the Verification Service will attempt to match appropriate Flavors to that host. If a Flavor is removed or added, all appropriate hosts will be updated to use the new Flavor, or to no longer use the deleted Flavor. Hosts that are currently using a BIOS where that BIOS versions’ PLATFORM Flavor was deleted will now appear Untrusted, for example. This can be used to easily flag as Untrusted hosts that are using software that has been End-Of-Lifed, or perhaps an OS kernel with a known security vulnerability.

Note: See the Flavor Management section for additional details on how flavors can be managed, and how the Flavor matching engine works. The sample workflow provided here is intended to be an introduction only.
6.2.1 Importing a Flavor from a Sample Host

POST https://server.com:8443/mtwilson/v2/flavors
input:
{  "connection_string": "<connection string>",
"partial_flavor_types": ["PLATFORM", "OS", "HOST_UNIQUE"],
"flavorgroup_name": ",
"tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}

Requires the permission "flavors:create"

**Note:** The HOST_UNIQUE Flavor parts, used by Red Hat Enterprise Linux and VMWare ESXi host types, MUST be created for each registered host of that type, and should in general be imported from that host. This means that importing the HOST_UNIQUE flavor should always be done for each host registered (except for Windows hosts, which do not have HOST_UNIQUE measurements).

To import ONLY the HOST_UNIQUE Flavor part from a host:

POST https://server.com:8443/mtwilson/v2/flavors
input:
{  "connection_string": "<connection string>",
"partial_flavor_types": ["HOST_UNIQUE"],
"flavorgroup_name": ",
"tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}

Requires the permission "flavors:create"

6.2.2 Creating a Flavor Manually

Flavors can be directly created (rather than importing from a sample host) if the required information is known. If no Flavorgroup is specified, the Flavor will be placed in the "automatic" group. Note that the "label" is a required field and must be unique.

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

input:
{  "connection_string": ",
"tls_policy_id": ",
"flavor_collection":  
"flavors": [{
  "meta": 
  "vendor": "INTEL",
  "description": 
  "flavor_part": "PLATFORM",}]}
"label": "Intel Corporation SE5C610.86B.01.01.1008.031920151331_TPM1.2",
"bios_name": "Intel Corporation",
"bios_version": "SE5C620.86B.00.01.0004.071220170215",
"tpm_version": "2.0"
},
"hardware": {
"processor_info": "...",
"processor_flags": "...",
"feature": {
"tpm": {
"enabled": true,
"pcr_banks": [
"SHA1",
"SHA256"
]
},
"txt": {"enabled": true}
}
},
"pcrs": {
"SHA1": {
"pcr_0": {"value": "d2ed125942726641a726c4f92be67d531a0def"},
"pcr_17": {
"value": "1ec12004b371e3afbd43d04155abde7476a3794fa",
"event": ...
}

Requires the permission "flavors:create"

6.3 Creating the Default SOFTWARE Flavor (Linux Only)

As part of the new Application Integrity feature added in Intel® SecL-DC version 1.5, a new default SOFTWARE Flavor part is provided so that the Linux Trust Agent itself can be measured and included in the attestation process. The default SOFTWARE Flavor includes a manifest for the static files and folders in the Trust Agent. The manifest is automatically deployed to each Linux Trust Agent during the provisioning step.

Note: The Linux Trust Agent must be rebooted after the Provisioning step is completed (typically Provisioning happens during installation, based on whether all of the required variables are set in the trustagent.env file). Rebooting allows the default SOFTWARE Flavor manifest to be measured and extended to the TPM PCRs. If the reboot is not performed, the system will require a SOFTWARE Flavor, but the measurements will not exist, and the system will appear Untrusted. If an un-
rebooted host is used to create the SOFTWARE Flavor, the Flavor will be created based on measurements that do not exist, and will fail.

The SOFTWARE Flavor part should be created separately from the other Flavor parts. Only one default SOFTWARE Flavor needs to be created for each version of the Linux 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://server.com:8443/mtwilson/v2/flavors
input: 
{ "connection_string": "<connection string>",
"partial_flavor_types": ["SOFTWARE"],
"flavorgroup_name": "",
"tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}

Requires the permission "flavors:create"
```

### 6.4 Creating and Provisioning Asset Tags

Asset Tags represent a set of key/value pairs that can be associated with a host in hardware. This enables usages around restricting workflows to specific hosts based on tags, which could include location information, compliance tags, etc.

ISeCL creates Asset Tags by creating a certificate containing the list of key/value pairs to be tagged to the host, with the host's hardware UUID as the certificate subject. A hash of this certificate is then written to an NVRAM index in the host's TPM. This value is included in TPM quotes, and can be attested using an Asset Tag flavor that matches up the expected value and the actual key/value pairs.

#### 6.4.1 Creating Asset Tag Certificates

Asset Tag certificates can be created with a single REST API call, with any number of key/value pairs. Note that one certificate must be created for each host to be tagged, even if they will all be tagged with identical key/value pairs.

```json
POST https://verification.server.com:8443/mtwilson/v2/tag-certificates
{"hardware_uuid": "<hardware UUID of host to be tagged>",
"selection_content": [

```
6.4.2 Deploying Asset Tags

6.4.2.1 Windows and Red Hat Enterprise Linux

Asset Tags can be provisioned to a Windows or RHEL host via a REST API request on the Verification Service that will in turn make a request to the Trust Agent on the host to be tagged.

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

6.4.2.2 VMWare

Since VMWare ESXi hosts do not use a Trust Agent, the process for writing Asset Tags to a VMWare host is different from RHEL or Windows. A new interface has been added to ESXi via a new "esxcli" command starting in vSphere 6.5 Update 2 that allows the Asset Tag information to be written to the TPM via a command-line command. The older process is also described below.

The high-level workflow for using Asset Tags with VMWare ESXi is:
1. Create the Asset Tag Certificate for the host.
2. Calculate the Certificate Hash value.
3. Provision the Certificate Hash value to the host TPM and reboot
4. Create the Asset Tag Flavor.
**Note:** Asset Tag is currently not supported for VMWare hosts using TPM 2.0.

### 6.4.2.2.1 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:

   ```
   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 to generate the correct hash:

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

   This hash value will be what is actually written to the TPM NVRAM.

### 6.4.2.2.2 Provision the Certificate Hash to the Host TPM

Due to a new feature added in vSphere 6.5 Update 2, the process for provisioning Asset Tags on VMWare ESXi hosts has been significantly improved. Both the old and new process for provisioning Asset Tags is documented below. Intel recommends using vSphere 6.5 Update 2 or later due to the significant difference in the process.

#### vSphere 6.5 Update 2 or Later

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 command:

1. `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`

2. Reboot the host. After rebooting, the TPM PCR 22 will have the measured value of the hash.
vSphere 6.5 Update 1 or Older

There is no direct interface from VMWare vCenter or ESXi previous to vSphere 6.5 Update 2 that will write the Tag information to the host TPM.

Writing Asset Tag information to a TPM requires TPM ownership; VMWare ESXi takes TPM ownership with a secret password at boot time. This means that the process for writing Asset Tags to a VMWare host requires:

1. Clear TPM ownership.
   a. This can be done via the system BIOS, or using One Touch Activation through the IPMI interface (if enabled by the server OEM).

2. Reactivate TPM/TXT.
   a. This can be done via the system BIOS, or using One Touch Activation through the IPMI interface (if enabled by the server OEM).

3. Booting to an OS that has the ability to issue TPM commands
   a. Typically the provisioning OS used is Ubuntu or RHEL, booted temporarily using PXE.

4. Writing the Tag information
   a. The TPM index 0x40000010 must be defined, and the hash of the Asset Tag certificate must be written to that index.

5. Clear TPM ownership.
   a. This can be done via the system BIOS, or using One Touch Activation through the IPMI interface (if enabled by the server OEM).

6. Reactivate TPM/TXT.
   a. This can be done via the system BIOS, or using One Touch Activation through the IPMI interface (if enabled by the server OEM).

7. Boot back to VMWare ESXi.

When the system is rebooted to ESXi, the Trusted Boot process will extend the value to PCR22, and this value can be used during attestation.

6.4.2.2.3 Creating the Asset Tag Flavor (VMWare ESXi Only)

While for RHEL and Windows hosts the Asset Tag Flavor is automatically created during the Tag Provisioning step, for VMWare ESXi hosts the 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 vCenter connection string>",
 "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.

6.5 Retrieving Current Attestation Reports
GET https://verification.service.com:8443/mtwilson/v2/reports?latestPerHost=true

6.6 Retrieving Current Host State Information
GET https://verification.service.com:8443/mtwilson/v2/host-status?latestPerHost=true

6.7 Upgrading Hosts in the Datacenter to a New BIOS or OS Version
Software and firmware updates are a common occurrence in the datacenter. Automatic Flavor matching makes this process relatively simple:

1. Create a new Flavor for the new version. This may be manually created or imported directly from a sample host that has already received the upgrade. Be sure to create new Flavors for each TPM version represented in your datacenter.

```json
POST https://server.com:8443/mtwilson/v2/flavors
input:
{
  "connection_string": "<connection string>",
  "partial_flavor_types": ["PLATFORM", "OS", "HOST_UNIQUE"],
  "flavorgroup_name": "",
  "tls_policy_id": "TRUST_FIRST_CERTIFICATE"
}
```

2. Update the hosts to the new software or firmware version as normal. On the next attestation attempt, the Verification Service will automatically match the updated hosts to the new Flavor.

3. (Optional) If desired, delete the Flavor for the older version after the update is completed. This will cause any hosts that are still using the old version to attest as Untrusted. Which can easily flag hosts that missed the upgrade for remediation.

DELETE https://verification.service.com:8443/mtwilson/v2/flavors/<Flavor ID>

6.8 Removing Hosts From the Verification Service
Hosts can be deleted at any time. Reports for that host will remain in the Verification Service database for audit purposes.
The Host ID can be retrieved either at the time the host is created, or by searching hosts using the host’s hostname.

6.9 Removing Flavors

Flavors can be deleted; this will cause any hosts that match the deleted Flavor to evaluate as Untrusted. This can be done if, for example, an old BIOS version needs to be retired and should no longer exist in the datacenter. By deleting the PLATFORM Flavor, hosts with the old BIOS version will attest as Untrusted, flagging them for easy remediation.

DELETE https://verification.service.com:8443/mtwilson/v2/flavors/<Flavor ID>

6.10 Invalidating Asset Tags

Asset Tags can be deleted in two ways.

Deleting the ASSET_TAG Flavor part will retain the Asset Tag certificate in the database, but will cause the host using this Tag to no longer use the Asset Tag for attestation (the Tag result will be disregarded and no tags will be exposed in the attestation Reports).

DELETE https://verification.service.com:8443/mtwilson/v2/flavors/<Asset Tag Flavor ID>

Deleting the actual Asset Tag certificate will remove the certificate from the database, but will not actually affect attestation results (the authority for attestation results is the Flavor).

DELETE https://verification.service.com:8443/mtwilson/v2/tag-certificates/<Asset Tag Certificate ID>

6.11 Remediating an Untrusted attestation

Hosts can become Untrusted for a wide variety of causes. The first clue to finding the root cause for an Untrusted attestation is the attestation Report itself – the Report will show Trust results for the PLATFORM, OS, HOST_UNIQUE, and ASSET_TAG Flavor parts individually, along with the OVERALL trust. If the Report shows that the PLATFORM Flavor part trust is “false” for example, it means that the PLATFORM measurements did not match any Flavors in the host’s Flavorgroup.

Untrusted attestation Reports will contain “faults” that describe the specific attestation rules that were not satisfied. This often shows enough information to describe the cause of the Untrusted status. A fault like “RequiredButNotDefined” means that a Flavor part is required by the Flavorgroup policy, but no Flavors for that Flavor part exist in the Flavorgroup.
(for example, generally Flavorgroups should always require a PLATFORM Flavor part; if no PLATFORM Flavors are in the Flavorgroup, hosts in the Flavorgroup will attest with this fault).

Other faults include:

**“PcrMatchesConstant”** - describes a rule that evaluates whether a TPM PCR has a specific value

**“PcrEventLogIntegrity”** - the module event log is replayed during attestation to verify that the resulting measurement matches the actual value in the module PCR. If the replay does not match, it indicates the event log cannot itself be trusted.

**“AikCertificateTrusted”** – This rule evaluates whether the TPM quote was signed by the TPM associated with this host. As part of host registration, the public half of the Attestation Identity Keypair is captured, and this public key is used to verify the signature on TPM quotes from that host.

See the Appendix for a full list of the rules evaluated during Attestation.

The Flavor matching engine will use the most-similar Flavor for the attestation Report in the case of an Untrusted result.

The fault will explain in a general sense what rule the host attestation violated. To remediate, the rule will need to be satisfied. This could mean creating a new Flavor to match the actual observed values, or it could mean that the host has been tampered with and should have its BIOS flashed or OS reloaded.

### 6.12 Attestation Reporting

Attestation results are delivered in the form of Host Reports. A Report can delivered in several different formats, which can change the type of data returned.

The preferred format for Host Reports is a SAML attestation. A SAML-formatted report includes a chain or signatures that provides auditability for the Report. The SAML attestation will include the base trust status of the host, as well as the overall trust for each individual Flavor used in the attestation. The Report will also contain host information, such as TPM version, Operating System name and version, BIOS version, etc. The SAML Report will not, however, contain individual measurements and comparisons of values. This format of the Report is ideal for securely communicating the trust status of a host and for audit history.

Attestation Reports can also be retrieved in JSON or XML format. These formats will not include the signature chain provided in the SAML format, but will contain the actual measurement values and expected Flavor values used for comparison. These reports are typically used for remediation, because they will show specifically why a given Host attested as Untrusted.
The format for a Report is determined by the “Accept” header in the request.

Attestations are automatically generated in the Verification Service by a repeating scheduled background process. This process looks for Attestation Reports that are close to expiration, and triggers a new Attestation Report. By default, Attestation Reports are valid for 90 minutes, and the background refresh process will trigger a new attestation when a Report is found to be within 3 minutes of expiration.

A user can either retrieve the most recent currently valid Attestation Report for a given host, or may trigger a new Attestation Report to be generated. Typically, it is best to retrieve an existing Report for performance reasons. Generating a new Attestation Report requires the generation of a new TPM quote from the TPM of the host being attested; TPM performance differs greatly between vendors, and a quote can take anywhere between 2-7 seconds to generate.

6.12.1 Sample Call – Generating a New Attestation Report

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

input: {"host_name":"host-1"}

Requires the permission “reports:create”

6.12.2 Sample Call – Retrieving an Existing Attestation Report

GET https://server.com:8443/mtwilson/v2/reports?hostName=HostName.server.com

Below are the supported criteria options in order of precedence. If no host filter criteria is specified, then results are returned for all active hosts.

- id - unique UUID of the report entry in the database
- hostId - unique UUID of the host entry in the database
- hostName - name of the host
- hostHardwareId - hardware UUID of the host
- hostStatus - current state of the host, which supports the following options:
  - CONNECTED - host is in connected state
  - QUEUE - host is in queue to be processed
  - CONNECTION_FAILURE - connection failure
  - UNAUTHORIZED - unauthorized
  - AIK_NOT_PROVISIONED - AIK certificate is not provisioned
  - EC_NOT_PRESENT - endorsement certificate is not present
MEASURED_LAUNCH_FAILURE - TXT measured launch failure
TPM_OWNERSHIP_FAILURE - TPM ownership failure
TPM_NOT_PRESENT - TPM is not present
UNSUPPORTED_TPM - unsupported TPM version
UNKNOWN - unknown host state

Requires the permissions "reports:search"

Other search criteria may also be used. By default, the most recent currently valid attestation is returned. However, different query parameters can be used to retrieve all attestations for a specific host over the last 30 days, for example.

6.13 Integration

Intel® SecL can be integrated with scheduler services (or potentially other services) to provide additional security controls. For example, by integrating Intel® SecL with the OpenStack scheduler service, the OpenStack placement service can incorporate the Intel® SecL security attributes into VM scheduling.

6.13.1 Integration Hub

The Integration Hub acts as the central integration point between the Verification Service and any number of third party services. The primary purpose of the Hub is to collect and maintain up-to-date attestation information, and to "push" that information to the external services. The secondary purpose is to allow for multitenancy. The Verification Service does not allow for permissions to be applied for specific hosts; a user with the "attestation" role can access all attestations for all hosts. The Integration Hub allows hosts to be associated with specific tenants; the tenants do not have direct access to the Verification Service, and the Hub will push attestations only for the associated hosts to a given tenant's integration endpoints.

For example, Tenant A is using hosts 1-10 for an OpenStack environment. Tenant B is using hosts 11-15 for a Docker environment. The Attestation Hub can push the information for hosts 1-10 to Tenant A’s OpenStack endpoint, and hosts 11-15 to Tenant B’s Docker endpoint. Neither tenant will have access to the Verification Service, and will not be able to see attestation or other host details regarding infrastructure used by other tenants.

Different integration endpoints can be added to the Integration Hub through a plugin architecture. By default, the Attestation Hub includes a plugin for OpenStack.
6.13.2 Integration with OpenStack

Starting in the Rocky release, OpenStack can now use "Traits" to provide qualitative data about Nova Compute hosts, and to establish Trait requirements for VM instances. The updated scheduler will place VMs requiring a given Trait on Nova Compute nodes that meet the Trait requirements.

Intel SecL-DC uses the Integration Hub to continually push platform integrity and Asset Tag information to the OpenStack Traits resources. This means the OpenStack scheduler natively supports workload scheduling incorporating Intel SecL-DC security attributes, including attestation report Trust status and Asset Tags. The OpenStack Placement Service will automatically attempt to place images with Trait requirements on compute nodes that have those Traits.

**NOTE:** This control only applies to instances launched using the OpenStack scheduler, and the Traits functions will not affect manually-launched instances where a specific Compute Node is defined (since this does not use the scheduler at all). Intel SecL-DC uses existing OpenStack interfaces and does not modify OpenStack code. The datacenter owner or OpenStack administrator is responsible for the security of the OpenStack workload scheduling process in general, and Intel recommends following published OpenStack security best practices.

6.13.2.1 Prerequisites

- Verification Service must be installed and running.
- OpenStack* Rocky Nova, Glance, Horizon, and Keystone services must be installed and running
- The Integration Hub must be installed and running.

6.13.2.2 Setting 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 set the Image Traits for Intel SecL-DC, a specific naming convention is used. This naming convention will match the Traits that the Integration Hub will automatically push to OpenStack. Two types of Traits are currently supported – one Trait is used to require that the Compute Node be Trusted in the Attestation Report, and the other Trait is used to require specific Asset Tag key/value pairs.

To require a "Trusted" Attestation Report:
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_TRUSTED=required

For example, to define a Trait that will require an Asset Tag where “State = CA,” use the following:

CUSTOM_ISECL_AT_STATE_CA= required

These Traits can be set using CLI commands for OpenStack Glance:

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

To remove a Trait so that it is no longer required for an Image:

```
openstack image unset --property trait:CUSTOM_ISECL_AT_STATE_CA <image_name>
openstack image unset --property trait:CUSTOM_ISECL_TRUSTED <image_name>
```

### 6.13.2.3 Configuring the Integration Hub for Use with OpenStack

After installation, the Integration Hub will automatically start retrieving Reports from the Verification Service. However, the Hub needs to be configured with endpoint information defining credentials and URLs for the OpenStack environment, and needs to have Hosts assigned to specific Tenants so that the right host info can be pushed to the correct OpenStack.

### 6.13.2.4 Integration Hub Account Creation

From the command line, run the following:

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

The username and password defined will be the credentials used for API requests.

Only one Hub admin account is needed; this step can be skipped if an administrative user has already been created.
6.13.2.5 Integration Hub Tenant Creation

At least one tenant must be created to receive the attestations. For the Hub, a single tenant is typically a single OpenStack controller. Below is an example using OpenStack where the api.endpoint is Nova and the auth.endpoint is Keystone.

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

{
   "name": "<Tenant name>",
   "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>"
      }, {
         "key": "plugin.provider",
         "value": "com.intel.attestationHub.plugin.nova.NovaPluginImpl"
      }]
   }
}
```
6.13.2.6 **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
```

6.13.2.7 **Assign Hosts to Tenants**

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.

```
POST https://server.com:19445/v1/host-assignments
{
  "tenant_id": "DC02284A-F525-6094-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.

6.13.2.8 **Scheduling Instances**

Once Trait requirements are set for Images and the Integration Hub is configured to push attributes to OpenStack, instances can be launched in OpenStack as normal. As long as the OpenStack Nova scheduler is used to schedule the workloads, only compliant Compute Nodes will be scheduled to run instances of controlled Images.

**NOTE:** This control only applies to instances launched using the OpenStack scheduler, and the Traits functions will not affect manually-launched instances where a specific Compute Node is defined (since this does not use the scheduler at all). Intel SecL-DC uses existing OpenStack interfaces and does not modify OpenStack code. The datacenter owner or OpenStack administrator is responsible for the security of the OpenStack workload scheduling process in general, and Intel recommends following published OpenStack security best practices.
6.13.3 Integration with Kubernetes

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.

6.13.3.1 Prerequisites

- Verification Service must be installed and running.
- Kubernetes Master Node must be installed and running
- Kubernetes Worker Nodes must be configured as physical hosts and attached to the Master Node
- The Integration Hub must be installed and running.

6.13.3.2 Installing the Intel® SecL Custom Resource Definitions

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-k8s-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-k8s-extensions/bin/
     type: ""
     name: extendedsched

3) Copy the isecl-k8s-extensions.bin installer to the Kubernetes Master and execute the installer

   ./isecl-k8s-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

**6.13.3 Integration Hub Account Creation**

From the command line, run the following:

   attestation-hub password username password --permissions "!"

The username and password defined will be the credentials used for API requests.

Only one Hub admin account is needed; this step can be skipped if an administrative user has already been created.
6.13.3.4 Integration Hub Tenant Creation

At least one tenant must be created to receive the attestations. For the Hub, a single tenant is typically a single OpenStack controller. Below is an example using OpenStack where the api.endpoint is Nova and the auth.endpoint is Keystone.

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

```json
{
   "name": "TenantName",
   "plugins": [
      {
         "name": "kubernetes",
         "properties": [
            {
               "key": "api.endpoint",
               "value": "https://kubernetes-master.server.com:6443"
            },
            {
               "key": "tenant.name",
               "value": "TenantName"
            },
            {
               "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.

### 6.13.3.5 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
```

### 6.13.3.6 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": "E2528A-F525-4C94-EE01-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.

### 6.13.3.7 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) Add the following to any Pod creation files:

```yaml
spec: 
  affinity: 
    nodeAffinity: 
      requiredDuringSchedulingIgnoredDuringExecution: 
        nodeSelectorTerms: 
          - matchExpressions: 
              - key: isecl.trusted 
                operator: In 
                values: 
                  - "true" 
              - key: TAG_Country 
                operator: In 
                values: 
                  - CA 
                  - US 
              - key: TAG_Customer 
                operator: In 
                values: 
                  - Coke 
                  - Pepsi 
              - key: TAG_State 
                operator: In 
                values: 
                  - CA 
```

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=US” Asset Tag, the pod should be launched with the matchExpression key “TAG_COUNTRY” with the value “US”.

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.
7 Flavor Management

7.1 Flavor Format Definitions

A Flavor is a standardized set of expectations that determines what platform measurements will be considered "trusted." Flavors are constructed in a specific format, containing a metadata section describing the Flavor, and then various other sections depending on the Flavor type or Flavor part.

7.1.1 Meta

The first part of a Flavor is the "meta" section:

```
"meta": {
    "vendor": "INTEL",
    "description": {
        "flavor_part": "PLATFORM",
        "bios_name": "Intel Corporation",
        "bios_version": "SE5C620.86B.00.01.0004.071220170215",
        "tpm_version": "2.0"
    }
}
```

This section defines the Flavor part and any versioning information.

**Note:** Even when the BIOS or OS version remains the same, the actual measurements in the measured boot process will be different between TPM 1.2 and TPM 2.0, and so the TPM version is captured here as well. The attributes in the Meta section are used by the Flavor matching engine when matching Flavors to Hosts.

7.1.2 Hardware

The "hardware" section is unique to PLATFORM flavor parts:

```
"hardware": {
    "processor_info": "54 06 05 00 FF FB EB BF",
    "processor_flags": "fpu vme de ...",
    "feature": {
        "tpm": {
            "enabled": true,
            "pcr_banks": [
                "SHA1",
                "SHA256"
            ]
        }
    }
}
```
This part of the Flavor defines expected hardware attributes of the host, and contains processor and TPM-related attributes.

### 7.1.3 PCRs

The last section of a Flavor is the "PCRs" section, which contains the actual expected measurements for any PCRs. This section will contain PCR measurements for each applicable algorithm supported by the TPM (SHA1 only for TPM 1.2, SHA256 and SHA1 sections for TPM 2.0).

Some PCRs simply have a value and nothing else. Other PCRs, however, contain different "event" measurements. This indicates that separate individual platform or OS components are independently measured and extended to the same PCR. PCRs with event measurements will contain an "Event" array that lists, in the correct order, all of the events in the measurement event log that are extended to this PCR. When the Verification Service attests a host against a given Flavor, each measurement event is compared to the Flavor value, and all of the events are replayed to confirm that a replay of all of the measurement extensions do in fact result in the hash seen in the PCR value. In this way, the Verification Service can ensure that the measurement event log contents are secure, and the individual measurements can be attested so that the cause for an Untrusted attestation can easily be seen.

The full PCRs section is not shown here due to length; see the sample Flavor sections for a full sample.
7.1.4 Sample PLATFORM Flavor

The PLATFORM Flavor part encompasses measurements that are unique to a specific platform, including the server OEM, BIOS version, etc. A PLATFORM Flavor can be “shared” across all hosts of the same model that have the same BIOS version.

```json
{
  "flavor_collection": {
    "flavors": [null]
  },
  "vendor": "INTEL",
  "description": {
    "flavor_part": "PLATFORM",
    "bios_name": "Intel Corporation",
    "bios_version": "SE5C620.86B.00.01.0004.071220170215",
    "tpm_version": "2.0"
  }
}
```

```json
"hardware": {
  "processor_info": "54 06 05 00 FF EB BP",
  "processor_flags": "fpu vme de ...",
  "feature": {
    "tpm": {
      "enabled": true,
      "pcr_banks": [
        "SHA1",
        "SHA256"
      ],
      "txt": {"enabled": true}
    }
  },
  "pcrs": {"SHA1": {
    "pcr_0": {
      "value": "d2ed12594276641a7260c4f92e92eb7d5f6b5def"},
    "pcr_17": {
      "value": "1a123004b371e3af43d04155a0def7476a3794fa",
      "event": {
        "digest_type": "com.intel.mtwilson.11.common.model.MeasurementSha1",
        "value": "2f2b7d57d0c4af9ac08a5b0d315d9c9a044",
        "label": "HASH_START",
        "info": {
          "ComponentName": "HASH_START",
          "EventName": "OpenSource.EventName"
        }
      }
    }
  }
}
```
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "ffb1806465d3a1b7531f5052a4efaad7c05a0e7",
"label": "BIOSAC_REG_DATA",
"info": {
  "ComponentName": "BIOSAC_REG_DATA",
  "EventName": "OpenSource.EventName"
}
},
{
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "3c585604e87f855973f38a3a21fba9392d2fc",
"label": "CPU_SCRTM_STAT",
"info": {
  "ComponentName": "CPU_SCRTM_STAT",
  "EventName": "OpenSource.EventName"
}
},
{
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "9069ca78e7450a285173431b3e52c5c25299e473",
"label": "LCP_CONTROL_HASH",
"info": {
  "ComponentName": "LCP_CONTROL_HASH",
  "EventName": "OpenSource.EventName"
}
},
{
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "5ba93c9db0cff93f52b521d7420e43f6e5da2784f",
"label": "LCP_DETAILS_HASH",
"info": {
  "ComponentName": "LCP_DETAILS_HASH",
  "EventName": "OpenSource.EventName"
}
},
{
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "3c585604e87f855973f38a3a21fba9392d2fc",
"label": "OSSINITDATA_CAP_HASH",
"info": {
  "ComponentName": "OSSINITDATA_CAP_HASH",
  "EventName": "OpenSource<EventName"
}
},
{
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "5ba93c9db0cff93f52b521d7420e43f6e5da2784f",
"label": "STM_HASH",
"info": {
  "ComponentName": "STM_HASH",
  "EventName": "OpenSource.EventName"
}
}
"value": "3d42560dcf165a5557b3156a21583f206d8ef10w",
"label": "MLE_HASH",
"info": {
"ComponentName": "MLE_HASH",
"EventName": "OpenSource.EventName"
},
},
{
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "274f5299dab88b98a7031bb09d9e613c2a28e56f",
"label": "NV_INFO_HASH",
"info": {
"ComponentName": "NV_INFO_HASH",
"EventName": "OpenSource.EventName"
},
},
{
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "ca96d6412b428020e570d3013dfc0b4b20250a",
"label": "tb_policy",
"info": {
"ComponentName": "tb_policy",
"EventName": "OpenSource.EventName"
},
},
{
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "d123e2f2b30f1effa8d522667af0dag4f48cfe",
"label": "vmlinuz",
"info": {
"ComponentName": "vmlinuz",
"EventName": "OpenSource.EventName"
},
},
{
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "f3742133e1a0deb48177a4ed225418e57f3fd1",
"label": "initrd",
"info": {
"ComponentName": "initrd",
"EventName": "OpenSource.EventName"
},
},
},
{SHA256: {
"pcr_0": {"value": "db83f0e8a1773c2116cc17986037cd81afcc6b1c8815572c6dalbebfba7f8a3"},
"pcr_1": {"value": "f5bdf5847a189356aeeff493245cfe785f0d5b2e0e1c0c3e6e9eb9812d8d91b0", "event": {
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value": "9301981c093654d5aa3430ba05c880a52eb22b9e8248f5f93e1f4dab1c947"},
},
}
7.1.5 Sample OS Flavor

An OS Flavor encompasses all of the measurements unique to a given OS. This includes the OS kernel and other measurements.
"pcrs": {"SHA1": {"pcr_17": {
    "value": "1ec12004b371e3af43d04155abde7476a3794fa",
    "event": [
        "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
        "value": "2fb7d57dcc545af9ae08d82bdf315dbcc59a044",
        "label": "HASH_START",
        "info": {
            "ComponentName": "HASH_START",
            "EventName": "OpenSource.EventName"
        }
    ],
    "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
    "value": "ffb180646522de17531fd5a2a6effaad7c5a047",
    "label": "BIOSAC_REG_DATA",
    "info": {
        "ComponentName": "BIOSAC_REG_DATA",
        "EventName": "OpenSource.EventName"
    }
},
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "3c585604e87e55973f731fed42c9392d2fc",
"label": "CPU_SCRTM_STAT",
"info": {
    "ComponentName": "CPU_SCRTM_STAT",
    "EventName": "OpenSource.EventName"
}
},
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "5ba939dc05ff93f2521d7420e43f6eda2784f",
"label": "LCP_DETAILS_HASH",
"info": {
    "ComponentName": "LCP_DETAILS_HASH",
    "EventName": "OpenSource.EventName"
}
}}
"ComponentName": "vmлинуз",

"EventName": "OpenSource.EventName"
}

"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha1",
"value": "f3742133e1a0deb48177a74ed225418e5cf73fd1",
"label": "initrd",
"info": {
"ComponentName": "initrd",
"EventName": "OpenSource.EventName"
}

"SHA256": {
"pcr_17": {
"value": "50bd59407a1893056eacff493245cfe785f045b2c0e1cc3e6e9eb5812d8d91bd",
"event": {
"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value": "9301981c093654d5aa3430ba05c880a52eb22b9e18248f5f93e1fe1dabc9b47",
"label": "HASH_START",
"info": {
"ComponentName": "HASH_START",
"EventName": "OpenSource.EventName"
}
}
}

"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value": "2785d1ed6f6b5d5b555dc24ce5e06a44ce8740fe7e01e15a10b1ff66caca90",
"label": "BIOSAC_REG_DATA",
"info": {
"ComponentName": "BIOSAC_REG_DATA",
"EventName": "OpenSource.EventName"
}

"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value": "67abdd721024f0ff4e6b3b4c2fc13bc5bad242d0b7851d456d8d203d15aa450",
"label": "CPU_SCRM_STAT",
"info": {
"ComponentName": "CPU_SCRM_STAT",
"EventName": "OpenSource.EventName"
}
}
"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value":
"df3f619804a92f4b405792dc43dd748e778ad52bc498e80524c01488119",
"label": "LCP_CONTROL_HASH",
"info":
    { 
        "ComponentName": "LCP_CONTROL_HASH",
        "EventName": "OpenSource.EventName"
    }
},
"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value":
"6e340b9ccf379a39ca544e6bb780a2c78901d3fb33738768511a30617afa01d",
"label": "LCP_DETAILS_HASH",
"info":
    { 
        "ComponentName": "LCP_DETAILS_HASH",
        "EventName": "OpenSource.EventName"
    }
},
"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value":
"6e340b9ccf379a39ca544e6bb780a2c78901d3fb33738768511a30617afa01d",
"label": "STM_HASH",
"info":
    { 
        "ComponentName": "STM_HASH",
        "EventName": "OpenSource.EventName"
    }
},
"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value":
"67abdd721024f0f4e0b3f4c2fc13bc5bad42d2b789d456d80d203d15aa3450",
"label": "OSSINITDATA_CAP_HASH",
"info":
    { 
        "ComponentName": "OSSINITDATA_CAP_HASH",
        "EventName": "OpenSource.EventName"
    }
},
"digest_type":
"com.intel.mtwilson.lib.common.model.MeasurementSha256",
"value":
"26e1d98742f79c950c637f8067b0b72a1b0e8ff75b4e609e7e17321ac3f4",
"label": "MLE_HASH",
"info":
    { 
        "ComponentName": "MLE_HASH",
        "EventName": "OpenSource.EventName"
    }
}
"digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
  "value": "0f6e0c7a5944963d7081ea494ddff1e9afa689e148e39f684db06578869ea33b",
  "label": "NV_INFO_HASH",
  "info": {
    "ComponentName": "NV_INFO_HASH",
    "EventName": "OpenSource.EventName"
  }
},
{
  "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
  "value": "27808f64e6383982cd8b3c19fcb3457c0b65f465f779d89b668839eaf263a67",
  "label": "tb_policy",
  "info": {
    "ComponentName": "tb_policy",
    "EventName": "OpenSource.EventName"
  }
},
{
  "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
  "value": "c89ad1d1e9adaa7ecf8f23687639b2472685f7d1b9f379b9a49974b66ed9638",
  "label": "vmlinuz",
  "info": {
    "ComponentName": "vmlinuz",
    "EventName": "OpenSource.EventName"
  }
},
{
  "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
  "value": "81b88a268e697ccf1790d4b9d748a8f395acfd47aa67c9845479d4e8456f77",
  "label": "initrd",
  "info": {
    "ComponentName": "initrd",
    "EventName": "OpenSource.EventName"
  }
}
"flavorgroup_name": "mtwilson_automatic"
### 7.1.6 Sample HOST_UNIQUE Flavor

Host-Unique flavors define measurements for a specific host. This can be either a single large flavor that incorporates all of the host measurements into a single flavor document used only to attest a single host, or can be a small subset of measurements that are specific to a single host. For example, some VMWare module measurements will change from one host to the next, while most others will be shared assuming the same ESXi build is used. The full Flavor requirement for such a host would include Host-Unique flavors to cover the measurements that are unique to only this one host, and would still use a generic PLATFORM and OS flavor for the other measurements that would be identical for other similarly configured hosts.

**Note:** The HOST_UNIQUE Flavors are unique to a specific host, and should always be imported directly from the specific host. Windows hosts do not require a HOST_UNIQUE flavor part.

```json
{"flavors": [{
  "meta": { 
    "id": "4d387cbd-f72b-4742-bde5-c5b0f6ed59e0",
    "vendor": "INTEL",
    "description": { 
      "flavor_part": "HOST_UNIQUE",
      "source": "Purley11",
      "bios_name": "Intel Corporation",
      "bios_version": "SE5C620.86B.00.01.0004.071220170215",
      "os_name": "RedHatEnterpriseServer",
      "os_version": "7.4",
      "tpm_version": "2.0",
      "hardware_uuid": "00448C61-46F2-E711-906E-001560A04062"
    }
  },
  "pcrs": { 
    "SHA256": { 
      "pcr_i?": { 
        "value": "f9ef8c53ddfc8096d36eda5506436c524b4bfa2bd451e89aa102f03181722176",
        "event": { 
          "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
          "value": "df3f619804a92fda4057192de3cd43ed74e2838edc52bcb499be3e80524e014b81119",
          "label": "LCP_CONTROL_HASH",
          "info": { 
            "ComponentName": "LCP_CONTROL_HASH",
            "EventName": "OpenSource.EventName"
          }
        }
      }
    }
  }
}]}
```
"value": "09f468dfc1d98a1fee86eb7297a56b0e097d57be66db4eeae539061332da2e723",
"label": "initrd",
"info": {
  "ComponentName": "initrd",
  "EventName": "OpenSource.EventName"
}
}
}
,"pcr_18": {
  "value": "c1f7bfae5f270d9f13aa8620b8977951d6b759f1131fe9f9289317f3a56efa1",
  "event": [
    {
      "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
      "value": "df3f619804a92fd4057192dc43dd748ea778adc52bc498cc805240c14981119",
      "label": "LCP_CONTROL_HASH",
      "info": {
        "ComponentName": "LCP_CONTROL_HASH",
        "EventName": "OpenSource.EventName"
      }
    }
  ]
}
}
,"SHA1": {
  "pcr_17": {
    "value": "48695f747a3d494710bd14d20cb0a93c7a485cc",
    "event": [
      {
        "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
        "value": "9069ca78e7450a28517341b3e52c5c25299e473",
        "label": "LCP_CONTROL_HASH",
        "info": {
          "ComponentName": "LCP_CONTROL_HASH",
          "EventName": "OpenSource.EventName"
        }
      },
      {
        "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
        "value": "b1f8db372e396bb128280821b7e0ac54a5ec2791",
        "label": "initrd",
        "info": {
          "ComponentName": "initrd",
          "EventName": "OpenSource.EventName"
        }
      }
    ]
  }
}
,"pcr_18": {
  "value": "983ec7db975ed31e2c85ef8e375c038d6d037efb",
  "event": [
    {
      "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
      "value": "c1f7bfae5f270d9f13aa8620b8977951d6b759f1131fe9f9289317f3a56efa1",
      "label": "LCP_CONTROL_HASH",
      "info": {
        "ComponentName": "LCP_CONTROL_HASH",
        "EventName": "OpenSource.EventName"
      }
    }
  ]
}
,"pcr_17": {
  "value": "48695f747a3d494710bd14d20cb0a93c7a485cc",
  "event": [
    {
      "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
      "value": "9069ca78e7450a28517341b3e52c5c25299e473",
      "label": "LCP_CONTROL_HASH",
      "info": {
        "ComponentName": "LCP_CONTROL_HASH",
        "EventName": "OpenSource.EventName"
      }
    },
    {
      "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
      "value": "b1f8db372e396bb128280821b7e0ac54a5ec2791",
      "label": "initrd",
      "info": {
        "ComponentName": "initrd",
        "EventName": "OpenSource.EventName"
      }
    }
  ]
}
,"pcr_18": {
  "value": "983ec7db975ed31e2c85ef8e375c038d6d037efb",
  "event": [
    {
      "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha256",
      "value": "c1f7bfae5f270d9f13aa8620b8977951d6b759f1131fe9f9289317f3a56efa1",
      "label": "LCP_CONTROL_HASH",
      "info": {
        "ComponentName": "LCP_CONTROL_HASH",
        "EventName": "OpenSource.EventName"
      }
    }
  ]
}
,"pcr_17": {
  "value": "48695f747a3d494710bd14d20cb0a93c7a485cc",
  "event": [
    {
      "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
      "value": "9069ca78e7450a28517341b3e52c5c25299e473",
      "label": "LCP_CONTROL_HASH",
      "info": {
        "ComponentName": "LCP_CONTROL_HASH",
        "EventName": "OpenSource.EventName"
      }
    },
    {
      "digest_type": "com.intel.mtwilson.lib.common.model.MeasurementSha1",
      "value": "b1f8db372e396bb128280821b7e0ac54a5ec2791",
      "label": "initrd",
      "info": {
        "ComponentName": "initrd",
        "EventName": "OpenSource.EventName"
      }
    }
  ]
}
7.1.7 Sample ASSET_TAG Flavor

Asset Tag flavor parts are unique to Asset Tag attestation. These flavors verify that the Asset Tag data in the host's TPM correctly matches the most recently created, currently valid Asset Tag certificate that has been deployed to that host.

```json
{
  "meta": {
    "id": "b3e0c056-5b6c-4b6b-95c4-de5f1473cac0",
    "description": {
      "flavor_part": "ASSET_TAG",
      "hardware_uuid": "<Hardware UUID of the server to be tagged>",
    }
  },
  "external": {
    "asset_tag": {
      "tag_certificate": {
        "encoded": "<Tag certificate in base64 encoded format>",
        "issuer": "CN=assetTagService",
        "serial_number": 151915354161,
        "subject": "<Hardware UUID of the server to be tagged>",
        "not_before": "2018-02-20T11:05:41-0800",
        "not_after": "2019-02-20T11:05:41-0800",
        "fingerprint_sha256": "2YjKiugD4kqQ2NhP34vyzdXurf8BpYMfweifpY1U12c",
        "attribute": [
          {"attr_type": {"id": "2.5.4.789.2"}, "attribute_values": [{"objects": {}}]},
          {"attr_type": {"id": "2.5.4.789.2"}, "attribute_values": [{"objects": {}}]},
          {"attr_type": {"id": "2.5.4.789.2"}, "attribute_values": [{"objects": {}}]}
        ]
      }
    }
  }
}
```
7.2 Flavor Matching

Flavors are matched to host by objects called “Flavor Groups.” A Flavor Group represents a set of rules to satisfy for a set of flavors to be matched to a host for attestation. For example, a Flavor Group can require that a PLATFORM Flavor and an OS Flavor be used for attestation. Without this level of association, a host that matches measurements for only a PLATFORM flavor, for example, can be attested as Trusted, even though the OS Flavor would attest the host as Untrusted.

Flavor matching can be automatic (the default), or can explicitly specify a host to which the Flavor Group must apply.

Automatic flavor matching allows for more ease in datacenter lifecycle management with updates and patches that may cause the appropriate flavors to change over time. Automatic flavor matching will trigger a new matching action when a new flavor is added, when an existing flavor is deleted, or when a host is initially attested as Untrusted. The system will automatically attempt to find a new set of flavors that match the Flavor Group rules that will attest the host as Trusted. For example, if a host in your datacenter has recently had a BIOS update, the next attestation will cause the host to appear Untrusted (because the PLATFORM measurements will now differ). Using automatic flavor matching, the Verification Service will automatically search for a new PLATFORM flavor that matches the actual BIOS version and measurement seen on the host. If a new BIOS version is successfully found, the Verification Service will use the new version for attestation, and the host will appear Trusted. If no matching PLATFORM flavor is found, the host will appear Untrusted. When automatic flavor matching is used, think of the various flavors in the Verification Service as a collection of valid configurations, and an attested host matching any combination of those configurations (within the confines of the Flavor Group requirements for which flavor types must be present) will be attested as Trusted.

Host-based flavor matching explicitly maps a specific host to a flavor. Host-based attestation requires that a host saves its entire configuration in a composite flavor document in the system, and then later validates against this flavor to detect any changes. In this case, if a host received a BIOS upgrade, the host will attest as Untrusted, and no attempt will be made to re-match a new flavor. An administrator will need to explicitly specify a new flavor to be used for that host.

7.2.1 When Does Flavor Matching Happen?

Generally speaking, a new Flavor match operation is triggered whenever a host is registered, whenever a host is attested and would be untrusted, and whenever a Flavor is added to or removed from a Flavor group.
When a new host is registered, the Verification Service will retrieve the Host Report and derive the platform information needed for Flavor matching (BIOS version, server OEM, OS type and version, TPM version, etc.). The Verification Service then searches through the Flavors in the same Flavor group that the host is in, and finds any Flavors that match the platform information.

If a Flavor is deleted, the Verification Service finds any hosts that are currently associated with that Flavor, and attempts to match them to alternative Flavors.

If a Flavor is added, the Verification Service looks for any hosts in the same Flavor group that are not currently matched to a Flavor of the appropriate Flavor part, and checks to see whether those hosts should be mapped to the new Flavor.

If a new Report is generated for a host and would not result in a Trusted attestation, the Verification Service will first repeat the Flavor matching process to be sure that no matching Flavors exist in the host’s Flavor group that would result in a Trusted attestation. If the Service still finds no matching Flavors, the host will appear as Untrusted.

7.2.2 **Flavor Matching Performance**

Flavor matching causes affected hosts to be moved into the "QUEUE" state while the host and Flavor are evaluated to determine whether the host and Flavor should be linked. Hosts can remain in the QUEUE state for varying amounts of time based on the extent of the Flavor match required. This means that the trust status of a host will not be actually updated to reflect a new Flavor until after the process finishes, which may take a few seconds or minutes depending on the number of registered hosts, Flavors in the same Flavor group, etc.

If a new host is registered, only that host will be added to the queue, and other hosts will be unaffected. The Verification Service will look for only the HOST_UNIQUE flavor part applicable to that specific host, and then will look at all PLATFORM and OS Flavors in the same Flavor group has the host, using the Flavor metadata and host info to narrow the results. The Service will match the new host to the most similar Flavors, and then move the host to the "CONNECTED" state and generate a new trust report.

When a new PLATFORM or OS Flavor is created, the Service will instead add all hosts in the same Flavor group as the new Flavors to the queue. Each host in the queue will then be re-evaluated against every PLATFORM and OS Flavor in the Flavorgroup to determine the closest match.

This means that adding a new Flavor can cause more hosts to each spend more time in the QUEUE state, as compared to adding a new host. For this reason, as a best practice for initial population of Flavors and hosts for a new deployment, it is suggested that Flavors be created before registering hosts. This is not a concern after the initial population of Flavors and hosts.
7.2.3 **Flavor Groups**

Flavor Groups represent a collection of one or more Flavors that are possible matches for a collection of one or more hosts. Flavor Groups link to both Flavors and hosts – a host in Flavor Group "ABC" will only be matched to Flavors in Flavor Group "ABC."

7.2.4 **Default Flavor Group**

By default the Verification Service includes a Flavor Group named “automatic” and another named “unique.” During host registration, the “automatic” Flavor Group is used as a default selection if no other Flavor Group is specified.

7.2.4.1 **automatic**

The automatic Flavor Group is used as the default Flavor Group for all hosts and all Flavor parts. If no other Flavor Groups are specified when creating Flavors or Hosts, all Hosts and Flavors will be added to this group. This is useful for datacenters that want to manage a single set of acceptable configurations for all hosts.

7.2.4.2 **unique**

The unique Flavor Group is used to contain HOST_UNIQUE Flavors. This Flavor group is used by the backend software and should not be managed manually.

7.2.5 **Flavor Match Policies**

Flavor Match Policies are used to define how the Flavor Match engine will match Flavors to hosts for attestation for a given Flavor Group. Each Flavor part can have defined Flavor Match Policies within a given Flavor Group.

```json
  "PLATFORM": { "any_of", "required" }
  "OS": { "all_of", "required_if_defined" }
  "HOST_UNIQUE": { "latest", "required_if_defined" }
  "ASSET_TAG": { "latest", "required_if_defined" }
  "SOFTWARE": { "all_of", "required_if_defined" }
```

The sample Policy above would require that a PLATFORM Flavor part be matched, but any PLATFORM Flavor part in the Flavor Group may be matched. The OS Flavor Part will only be required if there is an OS Flavor part in the Flavor Group; if there are no OS Flavor parts in the Group, the match will not be required. If more than one OS Flavor part exists in the Group, all of those OS parts will be required to match for a host to be Trusted.
7.2.5.1 Default Flavor Match Policy

The "automatic" Flavor Group, and any Flavor Group created without explicitly defining a Flavor Match Policy, will be created using the following Flavor Match Policy. This is the default behavior for Flavor Matching:

- "PLATFORM": { "any_of", "required" }
- "OS": { "any_of", "required" }
- "HOST_UNIQUE": {"latest","required_if_defined"}
- "ASSET_TAG": {"latest","required_if_defined"}
- "SOFTWARE": {"all_of","required_if_defined"}

7.2.5.2 ANY_OF

The ANY_OF Policy allows any Flavor of the specified Flavor part to be matched. If the Flavor Group contains OS Flavor 1 and OS Flavor 2, a host will be Trusted if it matches either OS Flavor 1 or OS Flavor 2.

7.2.5.3 ALL_OF

The ALL_OF Policy requires all Flavors of the specified Flavor Part in the Flavor Group to be matched. For example, if Flavor Group X contains PLATFORM Flavor Part 1 and PLATFORM Flavor Part 2, a host in Flavor Group X will need to match both PLATFORM Flavor 1 and PLATFORM Flavor 2 to attest as Trusted. If the host matches only one of the Flavors, or neither of them, the host will be attested as Untrusted.

7.2.5.4 LATEST

The LATEST Policy requires that the most recently created Flavor of the specified Flavor part be used when matching to a host. For example:

- "ASSET_TAG": { "latest", "required_if_defined" }

ASSET_TAG Flavor parts by default use the above Policy. This means that if Asset Tag Flavors are in the Flavor Group, the most recently created Asset Tag Flavor will be used. If no Asset Tag Flavors are present in the Flavor Group, then this Flavor part will be ignored.
7.2.5.5 REQUIRED

The REQUIRED Policy requires a Flavor of the specified part to be matched. For example:

```
"PLATFORM": { "any_of", "required" }
```

This policy means that a PLATFORM Flavor part must be used; if the Flavor Group contains no PLATFORM Flavor parts, hosts in this Flavor Group will always count as Untrusted.

7.2.5.6 REQUIRED_IF_DEFINED

The REQUIRED_IF_DEFINED Policy requires that a Flavor part be used if a Flavor of that part exists. If no Flavor part of this type exists in the Flavor Group, the Flavor part will not be required.

```
"ASSET_TAG": { "latest", "required_if_defined" }
```

ASSET_TAG Flavor parts by default use the above Policy. This means that if Asset Tag Flavors are in the Flavor Group, the most recently created Asset Tag Flavor will be used. If no Asset Tag Flavors are present in the Flavor Group, then this Flavor part will be ignored.

7.2.6 Flavor Match Event Triggers

Several events will cause the background queue service to attempt to re-match Flavors and hosts:

1. Host registration
   - This event is the first time a host will be attempted to be matched to appropriate Flavors in the same Flavor Group, and affects only the host that was added (other hosts will not be re-matched to Flavors when you add a new host).

2. Flavor creation
   - When a new Flavor is added to a Flavor Group, the queue system will repeat the Flavor match operation for all hosts in the same Flavor Group as the new Flavor.

3. Flavor deletion
   - When a Flavor is deleted, the queue system will repeat the Flavor match operation for all hosts in the same Flavor Group as the deleted Flavor.

4. Creation of a new Attestation Report
   - When a new Attestation Report is generated, if the host would attest as Untrusted with the currently-matched Flavors, the host being attested will be re-matched as part of the Report generation process. This ensures that Reports are always generated using the best possible Flavor matches available in the database.
7.2.7  Sample Flavorgroup API Calls

7.2.7.1  Create a New Flavorgroup

POST https://server.com:8443/mtwilson/v2/flavorgroups

Input:
{
    "flavorgroup_name": "firstTest",
    "flavor_match_policy_collection": {
        "flavor_match_policies": {
            "flavor_part": "PLATFORM",
            "match_policy": {
                "match_type": "ANY_OF",
                "required": "REQUIRED"
            }
        }
    }
}

Output:
{
    "id": "a0950923-596b-41f7-b9ad-09f525929ba1",
    "flavorgroup_name": "firstTest",
    "flavor_match_policy_collection": {
        "flavor_match_policies": {
            "flavor_part": "PLATFORM",
            "match_policy": {
                "match_type": "ANY_OF",
                "required": "REQUIRED"
            }
        }
    }
}

7.3  SOFTWARE Flavor Management

7.3.1  What is a SOFTWARE Flavor?

A SOFTWARE Flavor part defines the measurements expected for a specific application, or a specific set of files and folders on the physical host. SOFTWARE Flavors can be used to attest the boot-time integrity of any static files or folders on a physical server.

A single server can have multiple SOFTWARE Flavors associated. Intel® SecL-DC provides a "default" SOFTWARE Flavor that is deployed to each Trust Agent server during the provisioning step. This default Flavor includes the static files
and folders of the Trust Agent itself, so that the Trust Agent is measured during the server boot process, and its integrity is included in the attestation of the other server measurements.

Using SOFTWARE Flavors consists of two parts – creating the actual SOFTWARE Flavor, and deploying the SOFTWARE Flavor manifest to the host.
7.3.2 Creating a SOFTWARE Flavor part

Creating a new SOFTWARE Flavor requires creating a manifest of the files and folders that need to be measured.

There are three different types of entries for the manifest: Directories, Symlinks, and Files.

7.3.2.1 Directories

A Directory defines measurement rules for measuring a directory. Effectively this involves listing the contents of the directory and hashing the results; in this way, a Directory measurement can verify that no files have been added or removed from the directory specified, but will not measure the integrity of individual files (i.e., files can change within the directory, but cannot be renamed, added, or removed).

Directory entries can use regular expressions to define explicit Include and Exclude filters. For example, "Exclude=*.log" would exclude all files ending with .log from the measurement, meaning files with the .log extension can be added or removed from the directory.

```
<Dir Type="dir" Include=".*" Exclude="" Path="/opt/trustagent/hypertext/WEB-INF" />
```

7.3.2.2 Symlinks

A Symlink entry defines a symbolic link that will be measured. The actual symbolic link is hashed, not the file or folder the symlink points to. In this way, the measurement will detect the symbolic link being modified to point to a different location, but the actual file or folder pointed to can have its contents change.

```
<Symlink Path="/opt/trustagent/bin/tpm_nvinfo" />
```

7.3.2.3 Files

Individual files can be explicitly specified for measurement as well. Each file listed will be hashed and extended separately. This means that if any file explicitly listed this way changes its contents or is deleted or moved, the measurement will change, and the host will become Untrusted.

```
<File Path="/opt/trustagent/bin/module_analysis_da.sh" />
```

7.3.3 Sample SOFTWARE Flavor Creation Call

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 Connection String must point to the sample Trust Agent host. The Label defines the name of the new Flavor (ideally this should be the name of the application being measured for easier management).

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="Tomcat" 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>

7.3.4 Deploying a SOFTWARE Flavor Manifest to a Host

Once the SOFTWARE Flavor has been created, it can be deployed to any number of Trust Agent servers. This requires the Flavor ID (returned from Flavor creation) and the Host ID (returned from host registration). The Verification Service will send a request to the appropriate Trust Agent and create the manifest.

Note: After the SOFTWARE Flavor manifest is deployed to a host, the host must be rebooted. This will allow the measurements specified in the Flavor to be taken and extended to the TPM. Until the host is rebooted, the host will now appear Untrusted, as it now requires measurements from a SOFTWARE Flavor that have not yet been extended to the TPM.

POST https://server.com:8443/mtwilson/v2/rpc/deploy-software-manifest

Input:
{
  "flavor_id":"a6544ff4-6dc7-4c74-82be-578592e7e3ba",
  "host_id":"a6544ff4-6dc7-4c74-82be-578592e7e3ba"
}

7.3.5 SOFTWARE Flavor Matching

The default Flavor Match Policy for SOFTWARE Flavor parts is "ALL_OF","REQUIRED_IF_DEFINED". This means that all Software Flavors
defined in a Flavorgroup must match to all hosts in that Flavorgroup. If no SOFTWARE Flavors are in the Flavorgroup, then hosts can still be considered Trusted.

Because the default uses the "ALL_OF" Policy, it’s recommended to use Flavorgroups dedicated to specific software loadouts. For example, if a number of hosts will act as virtualization hosts and will have SOFTWARE Flavors for the hypervisor and VM management applications, those hosts should be placed in their own Flavorgroup as they will all run similar or identical application loadouts. If another group of servers in the datacenter will act as container hosts, these hosts might need SOFTWARE Flavors that include attestation of container runtimes and management applications, and will have a very different application loadout from the VM-based hosts. These should be placed in their own Flavorgroup, so that the VM hosts are attested using the hypervisor-related SOFTWARE Flavors, and the container hosts are attested using the container-related SOFTWARE Flavors.

As with other Flavor parts, hosts will be matched to Flavors in the same Flavorgroup that the host is added to, and will not be matched to Flavors in different Flavorgroups. Flavor matching will happen on the same events as for other Flavor parts.

### 7.3.6 Kernel Upgrades

Because the Application Integrity functionality involves adding a measurement agent (tbootXM) to initrd, an additional process must be followed when updating the OS kernel to ensure the new initrd also contains the measurement agent. This is not required if Application Integrity will not be used.

1) Update grub to have the boot menu-entry created for the new kernel version in grub.cfg.
   ```
   (grub2-mkconfig -o <path to grub file>)
   ```

2) Reboot the host and boot into new kernel menu-entry.

3) Generate a new initrd with tbootXM. (/opt/tbootxm/bin/generate_initrd.sh)

4) Copy the generated initrd to the boot directory. (cp /var/tbootxm/<generated initrd file name> /boot/)

5) Update the "TCB protection" menu-entry with the new kernel version. (edit /etc/grub.d/40_custom)

6) Update the default boot menu-entry to have new kernel version. (edit /etc/default/grub)

7) Update the grub to reflect the updates. (grub2-mkconfig -o <path to grub file>)

8) Reboot the host and boot into TCB protection menu-entry.

After updating the system with the new initrd, the Software Flavor should attest as Trusted. Note that changing grub and initrd does result in a new OS Flavor measurements, so an updated OS Flavor should be imported after updating the kernel and regenerating initrd.
8  Scalability and Sizing

8.1  Configuration Maximums

8.1.1  Registered Hosts

The Intel® SecL Verification Service can support a maximum of 2000 registered hosts with a single Verification Service instance with default settings.

8.1.2  HDD Space

The HDD space recommendations below represent expected log and database growth using default settings. Altering the database or log rotation settings, or the SAML expiration setting, may change the amount of disk space required. For default settings, 100 GB of disk space is recommended.

8.2  Database Rotation Settings

The Intel® SecL Verification Service database will automatically rotate the audit log table after one million records, and will retain up to ten total rotations. These settings are user-configurable if a longer retention period is needed.

`mtwilson.audit.log.num.rotations` - defines the maximum number of rotations before the oldest rotation is deleted to make space for a new rotation.

`mtwilson.audit.log.max.row.count` – defines the maximum number of rows in the audit log table before a rotation will occur.

8.3  Log Rotation

The Intel® SecL services (the Verification Service, Trust Agent, and Integration Hub) use Logrotate to rotate logs automatically during a daily cron job.

By default, logs are rotated once per month or when they exceed 1 GB in size, whichever comes first, and 12 total rotations will be retained.
## 9 Intel Security Libraries Configuration Settings

### 9.1 Verification Service

#### 9.1.1 Installation Answer File Options

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTWILSON_SERVER</td>
<td>192.168.1.1</td>
<td>Hostname or Ip address of the Verification Service</td>
</tr>
<tr>
<td>MTWILSON_API_BASEURL</td>
<td>https://&amp;{MTWILSON_SERVER}:8443/mtwilson/v1</td>
<td>v1 baseurl for the Verification Service. Generally this should not be changed.</td>
</tr>
<tr>
<td>MC_FIRST_USERNAME</td>
<td>administrator</td>
<td>Username for a new administrator-level user to be created during installation.</td>
</tr>
<tr>
<td>MC_FIRST_PASSWORD</td>
<td>mypassword123</td>
<td>Password for the new administrator-level user to be created during installation.</td>
</tr>
<tr>
<td>INSTALL_PKGS</td>
<td>&quot;logrotate&quot;</td>
<td>Defines the optional Verification Service components that will be installed. logrotate: the installer will automatically install and configure logrotate. This is optional.</td>
</tr>
<tr>
<td>LOG_SIZE</td>
<td>50M</td>
<td>Defines the log rotation size threshold for the Verification Service log. This is required if logrotate is installed.</td>
</tr>
<tr>
<td>LOG_OLD</td>
<td>3</td>
<td>Defines the number of rotated logs to be retained by logrotate before deletion. Required if logrotate is installed.</td>
</tr>
<tr>
<td>LOG_ROTATION_PERIOD</td>
<td>daily</td>
<td>Defines time interval. Log files are rotated when they grow bigger than size bytes, but not before the additionally specified time interval. Required if logrotate is installed.</td>
</tr>
<tr>
<td>LOG_COMPRESS</td>
<td>compress</td>
<td>Defines to compress log files. Old versions of log files are compressed with gzip(1) by default. Required if logrotate is installed.</td>
</tr>
<tr>
<td>LOG_DELAYCOMPRESS</td>
<td>delaycompress</td>
<td>Defines to postpone compression of the previous log file to the next rotation cycle. This is used with compress. Required if logrotate is installed.</td>
</tr>
<tr>
<td>LOG_COPYTRUNCATE</td>
<td>copytruncate</td>
<td>Defines to truncate the original log file in place after creating a copy, instead of moving the old log file and optionally creating a new one. Required if logrotate is installed.</td>
</tr>
<tr>
<td>MTWILSON_TLS_POLICY_ALLOW</td>
<td>certificate,certificate-digest,public-key,public-key-digest,TRUST_FIRST_CERTIFICATE,</td>
<td>Defines the TLS policies that will be allowed by the Verification Services. Policies not included in this list will not be created. See the TLS Policy Management section for details on TLS Policy types.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MTWILSON_DEFAULT_TLS_POLICY_ID</td>
<td>TRUST_FIRST_CERTIFICATE</td>
<td>Defines the default TLS policy to be used if no TLS policy is specified. Note that only TRUST_FIRST_CERTIFICATE can be specified here, because no other TLS policies exist at the time of installation. This setting is optional; if not used, there will be no default TLS Policy, and all API calls that require a TLS Policy will require the desired TLS Policy to be explicitly specified in the call.</td>
</tr>
<tr>
<td>DATABASE_HOSTNAME</td>
<td>127.0.0.1</td>
<td>Defines the database server IP address or hostname. This should be the loopback address for local database server installations, but should be the IP address or hostname of the database server if a remote database will be used.</td>
</tr>
<tr>
<td>DATABASE_PORTNUM</td>
<td>5432</td>
<td>Defines the port number for communication with the database server. By default with a local database server installation, this port will be set to 5432.</td>
</tr>
<tr>
<td>DATABASE_SCHEMA</td>
<td>mw_as</td>
<td>Defines the schema name of the database. If a remote database connection will be used, this schema must be created in the remote database before installing the Verification Service.</td>
</tr>
<tr>
<td>DATABASE_USERNAME</td>
<td>root</td>
<td>Username for accessing the database. If a remote database connection will be used, this user/password must be created and granted all permissions for the database schema before installing the Verification Service.</td>
</tr>
<tr>
<td>DATABASE_PASSWORD</td>
<td>dbpassword</td>
<td>Password for accessing the database. If a remote database connection will be used, this user/password must be created and granted all permissions for the database schema before installing the Verification Service.</td>
</tr>
<tr>
<td>MTWILSON_AUDIT_LOG_MAX_ROW_COUNT</td>
<td>1000000</td>
<td>Optional; uses a default of 1000000 if not specified during installation. Defines the maximum number of rows for a single rotation of the audit log table in the database. After reaching this number of records, the table will rotate.</td>
</tr>
<tr>
<td>MTWILSON_AUDIT_LOG_NUM_ROTATIONS</td>
<td>10</td>
<td>Optional; uses a default of 10 if not specified during installation. Defines the maximum number of rotations for the database audit table. After this number of rotations have occurred, subsequent rotations will result in the deletion of the oldest rotation to make room for the newest one.</td>
</tr>
<tr>
<td>POSTGRESQL_KEEP_PGP_PASS</td>
<td>TRUE</td>
<td>If set to &quot;TRUE,&quot; the Postgres database connectivity information will be stored in plaintext in the hidden file /opt/mtwilson/configuration/.pgpass. This prevents the user from being prompted for database credentials whenever the Hoist Verification Service starts.</td>
</tr>
<tr>
<td>ADD_POSTGRESQL_REPO</td>
<td>yes</td>
<td>This setting instructs the installer to add the Postgresql repository to the repository list.</td>
</tr>
<tr>
<td>TAG_VALIDITY_SECONDS</td>
<td>31536000</td>
<td>This value defines in seconds the length of time Asset Tag Certificates will remain valid.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>JETTY_TLS_CERT_DN</td>
<td>CN=Mt Wilson</td>
<td>Defines the Distinguished Name for the TLS Certificate</td>
</tr>
<tr>
<td>JETTY_PORT</td>
<td>8442</td>
<td>Insecure (http) Jetty port. This port must match the port defined in the endpoint.url value. Insecure (http) Jetty port. This port must match the port defined in the endpoint.url value.</td>
</tr>
<tr>
<td>JETTY_SECURE_PORT</td>
<td>8443</td>
<td>Secure (https) Jetty port. This port must match the port defined in the mtwilson.api.url value.</td>
</tr>
<tr>
<td>JAVAX_NET_SSL_KEYSTORE</td>
<td>/opt/mtwilson/configuration/keystore.jks</td>
<td>Defines the location of the Jetty webserver SSL keystore.</td>
</tr>
<tr>
<td>JETTY_TLS_CERT_DNS</td>
<td>devops5,localhost</td>
<td>Comma-separated list of hostnames to be added as Subject Alternative Names to the TLS certificate. Only connections to an IP address or hostname in the Subject Alternative Names list will be accepted; other connections will be rejected.</td>
</tr>
<tr>
<td>JETTY_TLS_CERT_IP</td>
<td>192.168.1.1,127.0.0.1</td>
<td>Comma-separated list of IP addresses to be added as Subject Alternative Names to the TLS certificate. Only connections to an IP address or hostname in the Subject Alternative Names list will be accepted; other connections will be rejected.</td>
</tr>
<tr>
<td>QUEUE_EXECUTION_INTERVAL</td>
<td>3</td>
<td>Defines the frequency in seconds at which the background queue process is executed</td>
</tr>
<tr>
<td>ESXIHOSTS_AUTOUPDATE_INTERVAL</td>
<td>120</td>
<td>Defines the frequency in seconds at which ESXi host information is retrieved from configured vCenters and updated in the database</td>
</tr>
<tr>
<td>MTWILSON_TELEMETRY_INTERVAL</td>
<td>86400</td>
<td>Defines the frequency in seconds at which telemetry data is collected. Telemetry data includes a simple count of all hosts currently registered.</td>
</tr>
</tbody>
</table>

### 9.1.2 Configuration Options

The Verification Service configuration is encrypted and stored in the file /opt/mtwilson/configuration/mtwilson.properties. To view or change any configuration settings, use the following commands:

**View Configuration:**

```bash
mtwilson export-config --stdout
```

To change the value of any configuration setting in mtwilson.properties, use the following command:

```bash
mtwilson config <key> <value>
```
<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mtwilson.queue.execution.interval</td>
<td>3</td>
<td>Defines in seconds the length of time between background queue operations. The background queue is the process that automatically matches Flavors to Hosts that share the same Flavorgroup.</td>
</tr>
<tr>
<td>mtwilson.queue.max.threads</td>
<td>32</td>
<td>The number of threads that the background queue will allocate for concurrent task execution.</td>
</tr>
<tr>
<td>mtwilson.queue.execution.timeout</td>
<td>60</td>
<td>The amount of time in seconds an individual background queue task is allowed to take before throwing a timeout exception.</td>
</tr>
<tr>
<td>mtwilson.telemetry.interval</td>
<td>86400</td>
<td>Defines in seconds the length of time between checks for the number of hosts registered in the Verification Service. By default this checks once every 24 hours. A report of the last 90 days of host counts can be retrieved through a REST API (see the Javadoc for details).</td>
</tr>
<tr>
<td>mtwilson.esxihosts.autoupdate.interval</td>
<td>120</td>
<td>Defines in seconds the length of time between checks to see if any changes have occurred in vCenter for any VMWare vCenter Cluster objects that have been registered with the Verification Service. If a new host has been added or removed from the Cluster in vCenter, the host will be added or removed respectively in the Verification Service as well.</td>
</tr>
<tr>
<td>saml.validity.seconds</td>
<td>3600</td>
<td>Defines in seconds the length of time a Report will remain valid. The Verification Service automatically checks for Reports that are nearing expiration and refreshes them; lowering this value will increase the frequency of automatic background Report generation.</td>
</tr>
<tr>
<td>mtwilson.tls.policy.allow</td>
<td>certificate,certificate,digest,public-key,public-key,digest,TRUST_FIRST_CERTIFICATE</td>
<td>Defines the TLS policies that will be allowed. TLS policies not in this list will be denied. This list is based off of the MTWILSON_TLS_POLICY_ALLOW list provided at installation time in the mtwilson.env answer file. See the TLS Policy section for information on TLS Policies.</td>
</tr>
<tr>
<td>mtwilson.default.tls.policy.id</td>
<td>TRUST_FIRST_CERTIFICATE</td>
<td>Defines the default TLS policy to be used if no TLS policy is specified. This value is based off of the MTWILSON_DEFAULT_TLS_POLICY_ID value provided at installation time in the mtwilson.env answer file. Note that only TRUST_FIRST_CERTIFICATE and INSECURE can be specified during installation, because no other TLS policies exist at the time of installation. This can be changed to a new TLS policy, however, by setting the ID of the TLS policy to be used as a default. This setting is optional.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>mtwilson.extensions.packageIncludeFilter.startsWith</td>
<td>com.intel.org.glassfish.jersey.media.multipart</td>
<td>Do not change this value. Java code package name string to include for classpath jar loading optimization.</td>
</tr>
<tr>
<td>mtwilson.extensions.fileIncludeFilter.contains</td>
<td>mtwilson, jersey-media-multipart</td>
<td>Do not change this value. Java jar filename string identifier to include for classpath jar loading optimization.</td>
</tr>
<tr>
<td>mtwilson.extensions.packageExcludeFilter.startsWith</td>
<td>java,javax</td>
<td>Do not change this value. Java code package name string to exclude for classpath jar loading optimization.</td>
</tr>
<tr>
<td>mtwilson.host</td>
<td>192.168.1.1</td>
<td>The IP address or hostname of the Verification Service. This is configured at installation time by the MTWILSON_SERVER value specified in the mtwilson.env answer file at installation time.</td>
</tr>
<tr>
<td>mtwilson.api.url</td>
<td><a href="https://192.168.1.1:8443/mtwilson/v1">https://192.168.1.1:8443/mtwilson/v1</a></td>
<td>Defines the baseurl the Verification Service v1 APIs. Note that this will need to be updated if the Jetty secure port is changed.</td>
</tr>
<tr>
<td>dbcp.validation.query</td>
<td>select 1</td>
<td>Query used to verify that the database is accessible</td>
</tr>
<tr>
<td>dbcp.validation.on.return</td>
<td>false</td>
<td>This property determines whether or not the pool will validate objects before they are borrowed from the pool.</td>
</tr>
<tr>
<td>dbcp.validation.on.borrow</td>
<td>true</td>
<td>This property determines whether or not the pool will validate objects before they are borrowed from the pool.</td>
</tr>
<tr>
<td>mtwilson.locale</td>
<td>en-US</td>
<td>Do not change this value.</td>
</tr>
<tr>
<td>mtwilson.db.user</td>
<td>root</td>
<td>Defines the database user</td>
</tr>
<tr>
<td>mtwilson.db.password</td>
<td>dbpassword</td>
<td>Defines the database password</td>
</tr>
<tr>
<td>mtwilson.db.driver</td>
<td>org.postgresql.Driver</td>
<td>Defines the database driver to be used. Do not change this value.</td>
</tr>
<tr>
<td>mtwilson.db.schema</td>
<td>mw_as</td>
<td>Defines the database schema name.</td>
</tr>
<tr>
<td>mtwilson.db.port</td>
<td>5432</td>
<td>Defines the database connection port. The default port for a locally installed Postgresql database server is 5432.</td>
</tr>
<tr>
<td>mtwilson.db.host</td>
<td>127.0.0.1</td>
<td>Defines the IP or hostname of the database server. By default for a locally installed Postgresql database server this will be 127.0.0.1.</td>
</tr>
<tr>
<td>mtwilson.audit.log.max.row.count</td>
<td>1000000</td>
<td>Defines the maximum number of rows for a single rotation of the audit log table in the database. After reaching this number of records, the table will rotate.</td>
</tr>
<tr>
<td>mtwilson.audit.log.num.rotations</td>
<td>10</td>
<td>Defines the maximum number of rotations for the database audit table. After this number of rotations have occurred, subsequent rotations will result in the deletion of the oldest rotation to make room for the newest one.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>--------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>mtwilson.privacyca.ek.p12.password</td>
<td></td>
<td>Defines Endorsement CA password</td>
</tr>
<tr>
<td>mtwilson.privacyca.aik.p12.password</td>
<td></td>
<td>Defines Privacy CA password</td>
</tr>
<tr>
<td>mtwilson.as.dek</td>
<td></td>
<td>Decryption key used for sensitive data encrypted in the database.</td>
</tr>
<tr>
<td>saml.key.alias</td>
<td>samlkey1</td>
<td>Alias for the SAML signing certificate. Do not change this value.</td>
</tr>
<tr>
<td>saml.keystore.file</td>
<td>SAML.jks</td>
<td>Keystore that contains the SAML signing key.</td>
</tr>
<tr>
<td>jetty.tls.cert.dn</td>
<td>CN=Mt Wilson</td>
<td>Defines the Distinguished Name for the TLS Certificate</td>
</tr>
<tr>
<td>jetty.port</td>
<td>8442</td>
<td>Insecure (http) Jetty port. This port must match the port defined in the endpoint.url value.</td>
</tr>
<tr>
<td>jetty.secure.port</td>
<td>8443</td>
<td>Secure (https) Jetty port. This port must match the port defined in the mtwilson.api.url value.</td>
</tr>
<tr>
<td>javax.net.ssl.keyStore</td>
<td>/opt/mtwilson/configuration/keystore.jks</td>
<td>Defines the location of the Jetty webserver SSL keystore.</td>
</tr>
<tr>
<td>jetty.tls.cert.dns</td>
<td>devops5,localhost</td>
<td>Comma-separated list of hostnames to be added as Subject Alternative Names to the TLS certificate. Only connections to an IP address or hostname in the Subject Alternative Names list will be accepted; other connections will be rejected.</td>
</tr>
<tr>
<td>jetty.tls.cert.ip</td>
<td>192.168.1.1,127.0.0.1</td>
<td>Comma-separated list of IP addresses to be added as Subject Alternative Names to the TLS certificate. Only connections to an IP address or hostname in the Subject Alternative Names list will be accepted; other connections will be rejected.</td>
</tr>
<tr>
<td>mtwilson.ca.dn</td>
<td>CN=mtwilson-ca,OU=mtwilson</td>
<td>CA distinguishable name</td>
</tr>
</tbody>
</table>

### 9.1.3 Command-Line Options

The Verification Service supports several command-line commands that can be executed only as the Root user:

**Syntax:**

mtwilson <command>

#### 9.1.3.1 Help

mtwilson help
Displays the list of available CLI commands.

9.1.3.2 Start
mtwilson start
Starts the services.

9.1.3.3 Stop
mtwilson stop
Stops the services.

9.1.3.4 Restart
mtwilson restart
Restarts the services.

9.1.3.5 Status
mtwilson status
Reports whether the service is currently running.

9.1.3.6 Uninstall
mtwilson uninstall
Uninstalls the service, including the deletion of all files and folders. Database content is not removed. See section 14.1 for additional details.

9.1.3.7 Version
mtwilson version
Reports the version of the service.

9.1.3.8 Fingerprint
mtwilson fingerprint
Displays the TLS certificate information.

9.1.3.9 Java-detect
mtwilson java-detect
Displays the detected path and installed version of Java.

9.1.3.10 **Erase-data**

mtwilson erase-data

Deletes all non-user information from the database.

9.1.3.11 **Erase-users**

mtwilson erase-users [--all]

Deletes all users from the database, except for the default administrative user. If the "--all" option is used, the administrative user will be deleted as well.

9.1.3.12 **Zeroize**

mtwilson zeroize

Shreds all secrets, keys, and configurations.

9.1.3.13 **Login-password**

mtwilson login-password [username] [password] [--permissions] [permission1] [permission2]...

Creates a new user with the specified username and password. If the --permissions option is used, the user can be directly assigned permissions.

**Note:** This command only allows the assignment of individual permissions, not roles. Permissions are defined in a domain:permission format. For example, hosts:create would allow the user to create new hosts, but not modify or delete existing hosts. Any number of permissions may be applied to a single user. Wildcards are also acceptable; the permissions *:* grants all permissions on all domains, effectively creating an administrative user.

9.1.3.14 **Export-config**

mtwilson export-config <infile|--in=infile|--out=outfile|--stdout>

Exports the current configuration. Configuration settings are stored in the encrypted file /opt/mtwilson/configuration/mtwilson.properties; this command allows the configuration to be decrypted or output to the console.

9.1.3.15 **Config**

mtwilson config [key] [value]
Configures a specified configuration setting to a specified value. Changing settings may require a service restart to take effect.

9.1.3.16 Setup

mtwilson setup [--force|--noexec] [task1 task2 ...]

Re-runs the installation setup tasks, or the specific tasks listed.

9.1.3.17 Replace-tls-key-pair

mtwilson replace-root-key-pair [--private-key=newprivatekey.pem] [--cert-chain=]

Replaces the TLS key pair and certificate. See the Certificate and Key Management section for more details.

9.1.4 Directory Layout

/etc/host-verification-service/
/var/log/host-verification-service/
/usr/share/java/host-verification-service
/usr/share/host-verification-service
/usr/sbin/

9.2 Trust Agent

9.2.1 Installation Answer File Options

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTWILSON_API_URL</td>
<td>https://&lt;Verification Service IP or hostname&gt;:8443/mtwilson/v2</td>
<td>Defines the base URL for the Verification Service API.</td>
</tr>
<tr>
<td>MTWILSON_TLS_CERT_SHA383</td>
<td>SHA384 hash of the Verification Service TLS certificate. This can be retrieved from the Verification Service in the file /opt/mtwilson/configuration/https.properties</td>
<td></td>
</tr>
<tr>
<td>MTWILSON_API_USERNAME</td>
<td>admin</td>
<td>Defines the username that will be used for authentication to the Verification Service API.</td>
</tr>
<tr>
<td>MTWILSON_API_PASSWORD</td>
<td>password</td>
<td>Defines the password that will be used for authentication to the Verification Service API.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>REGISTER_TPM_PASSWORD</td>
<td>y</td>
<td>If this is set to &quot;y,&quot; the Trust Agent will attempt to store the TPM owner secret with the Verification Service. This is used for &quot;Pull&quot; Asset Tag provisioning so that the TPM ownership does not need to be cleared (which reduces the number of needed reboots). Pull provisioning for Asset Tags is not recommended.</td>
</tr>
<tr>
<td>AUTOMATIC_REGISTRATION</td>
<td>no</td>
<td>If set to &quot;yes&quot; this will cause the Trust Agent installer to attempt to register the host with the Verification Service. Requires that &quot;PROVISION_ATTESTATION&quot; be set to &quot;y&quot;. If this is set to &quot;no&quot;, the registration will not happen after installation, and can be performed either using the trust agent command line commands, or using a REST API for the Verification Service.</td>
</tr>
<tr>
<td>TRUSTAGENT_LOGIN_REGISTER</td>
<td>true</td>
<td>If this is set to true,&quot; the Trust Agent will attempt to register the Trust Agent API credentials with the Verification Service. This will allow the Verification Service to accept connection strings to connect to this host without explicitly specifying the Trust Agent username and password. If this is set to &quot;false&quot; or not specified, the Verification Service will require the connection string to include the username and password. Setting this to &quot;true&quot; is recommended if the Trust Agent credentials will be randomly generated during installation.</td>
</tr>
<tr>
<td>PROVISION_ATTESTATION</td>
<td>y</td>
<td>If set to &quot;y,&quot; the Trust Agent will attempt to perform the Provisioning steps after installation completes. See the Trust Agent installation section of the Product Guide for details on Provisioning. If not specified or set to &quot;n,&quot; the Trust Agent will install but will not attempt to perform the Provisioning steps.</td>
</tr>
<tr>
<td>TRUSTAGENT_ADMIN_USERNAME</td>
<td>&lt;username&gt;</td>
<td>Defines the initial Trust Agent API user that will be created during Trust Agent installation. If this is not specified, the username and password will be randomly generated.</td>
</tr>
<tr>
<td>TRUSTAGENT_ADMIN_PASSWORD</td>
<td>&lt;password&gt;</td>
<td>Defines the initial Trust Agent API user that will be created during Trust Agent installation. If this is not specified, the username and password will be randomly generated.</td>
</tr>
<tr>
<td>CURRENT_IP</td>
<td>&lt;IP address&gt;</td>
<td>This IP is used for Attestation registration.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TPM_OWNER_SECRET</td>
<td>20 hex-encoded bytes</td>
<td>This can be used to preserve the TPM Owner Secret between installations so that TPM ownership does not need to be reset for a re-installation. Note: If the OS is re-installed, the file /var/lib/tpm/system.data contains sealed persistent TPM secrets unique to the TPM owner and needs to be preserved from just before the old OS is wiped and replaced immediately after trousers is re-installed in the new OS.</td>
</tr>
<tr>
<td>AIK_SECRET</td>
<td>20 hex-encoded bytes</td>
<td>Secret to generate AIK</td>
</tr>
<tr>
<td>JAVA_REQUIRED_VERSION</td>
<td>1.8</td>
<td>Defines the required Java version. Do not change this value.</td>
</tr>
<tr>
<td>TPMQUOTE_IPV4</td>
<td>Default: true</td>
<td>When enabled (not set or set to true) causes the challenger’s nonce to be extended with the IP Address of the trusted host for quoting to prevent quote relay attacks.</td>
</tr>
<tr>
<td>TPM_SRK_SECRET</td>
<td>Example: 00000000000000000000</td>
<td>Many tools assume this well-known SRK comprised of 20 zero bytes.</td>
</tr>
<tr>
<td>TRUSTAGENT_KEYSTORE_PASSWORD</td>
<td>Generated automatically. Example: 9JF7+HhpMUM_</td>
<td>The password used to access the trustagent.jks file with the keytool.</td>
</tr>
<tr>
<td>TRUSTAGENT_PASSWORD</td>
<td>No default value.</td>
<td>The password used to encrypt and decrypt the trustagent.properties file. This password must be exported in an environment variable for the Trust Agent to use it.</td>
</tr>
<tr>
<td>TRUSTAGENT_TLS_CERT_DN</td>
<td>CN=trustagent,OU=DCG, O=Intel, L=Folsom,C=US</td>
<td>Determines the subject name of the trust agent’s TLS certificate. These names are added as subject alternative names on the Trust Agent’s TLS certificate. By default, all names in /etc/hosts corresponding to local IP Addresses are used. If not specified, the installer performs a reverse DNS lookup for all IP Addresses found in the ifconfig output. In some environments this can cause a delay during installation. Manually specifying the subject names can eliminate this delay.</td>
</tr>
<tr>
<td>TRUSTAGENT_TLS_CERT_DNS</td>
<td>Generated automatically. Comma-separated list of all hostnames used by the host</td>
<td>Comma-separated list of DNS names to be added as Subject Alternative Names to the TLS certificate. Only connections to an IP address or hostname in the Subject Alternative Names list will be accepted; other connections will be rejected.</td>
</tr>
<tr>
<td>TRUSTAGENT_TLS_CERT_IP</td>
<td>Possible values: Comma-separated list of IP Addresses.</td>
<td>Comma-separated list of IP addresses to be added as Subject Alternative Names to the TLS certificate. Only connections to an IP address or hostname in the Subject Alternative Names list will be accepted; other connections will be rejected.</td>
</tr>
</tbody>
</table>
9.2.2 **Configuration Options**

The Trust Agent configuration .properties file is encrypted during installation. To view the contents of the `trustagent.properties` file, use the following commands:

**View Configuration:**

`tagent export-config --stdout`

**Change a Setting:**

`tagent config <property_name> <new_value>`

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>trustagent.keystore.password</code></td>
<td>Generated automatically</td>
<td>Password for the <code>trustagent.jks</code> keystore</td>
</tr>
<tr>
<td><code>trustagent.tls.cert.ip</code></td>
<td>Comma-separated list of IP addresses</td>
<td>These addresses are added as subject alternative names on the Trust Agent's TLS certificate. By default, all IP Addresses shown in the <code>ifconfig</code> output are used. Connections to the Trust Agent using an IP or hostname not in this list will be rejected.</td>
</tr>
<tr>
<td><code>trustagent.tls.cert.dns</code></td>
<td>Comma-separated list of hostnames</td>
<td>These names are added as subject alternative names on the Trust Agent's TLS certificate. By default, all names in <code>/etc/hosts</code> corresponding to local IP Addresses are used. If not specified, the installer performs a reverse DNS lookup for all IP Addresses found in the <code>ifconfig</code> output. In some environments this can cause a delay during installation. Manually specifying the subject names can eliminate this delay. Connections to the Trust Agent using an IP or hostname not in this list will be rejected.</td>
</tr>
<tr>
<td><code>mtwilson.tls.cert.sha384</code></td>
<td>SHA384 hash of the Verification Service TLS certificate. Obtain this from the Verification Service server, in <code>/opt.mtwilson/configuration/https.properties</code></td>
<td></td>
</tr>
<tr>
<td><code>mtwilson.api.username</code></td>
<td>User specified</td>
<td>Verification Service username used for any REST calls from the Trust Agent to the VS. This includes Trust Agent Provisioning tasks, and optionally may include host registration and/or importing the HOST_ UNIQUE flavor part.</td>
</tr>
<tr>
<td><code>hardware.uuid</code></td>
<td>Hardware UUID of the host</td>
<td></td>
</tr>
<tr>
<td><code>tpm.srk.secret</code></td>
<td><code>00000000000000000000000000000000</code></td>
<td>Storage Root Key secret. Many tools assume the SRK secret to be a &quot;well-known&quot; secret of twenty bytes of zero.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>trustagent.admin.username</td>
<td>User specified, or randomly generated</td>
<td>Username of the Trust Agent administrative user created during installation. This is inherited from the TRUSTAGENT_ADMIN_USERNAME value in trustagent.env during installation if specified. If not specified, this value will be generated randomly during installation. This user is used by external services (including the Verification Service) when making REST API calls to the Trust Agent.</td>
</tr>
<tr>
<td>mtwilson.api.password</td>
<td>User specified</td>
<td>Verification Service password used for any REST calls from the Trust Agent to the VS. This includes Trust Agent Provisioning tasks, and optionally may include host registration and/or importing the HOST_UNIQUE flavor part.</td>
</tr>
<tr>
<td>aik.secret</td>
<td>Generated automatically as 20 hex-encoded bytes.</td>
<td>20 hex-encoded bytes, Secret to generate AIK</td>
</tr>
<tr>
<td>current.ip</td>
<td>10.105.167.121</td>
<td>This IP address or hostname value is used as the hostname field when automatically registering the host when using the “tagent attestation-registration” command.</td>
</tr>
<tr>
<td>mtwilson.extensions.packag eExcludeFilter.startsWith</td>
<td>java, javax</td>
<td>Do not change this value.</td>
</tr>
<tr>
<td>mtwilson.api.url</td>
<td>https://&lt;IP or hostname&gt;:8443/mtwilson/v2</td>
<td>Baseurl for the Verification Service API.</td>
</tr>
<tr>
<td>mtwilson.extensions.packag eIncludeFilter.startsWith</td>
<td>com.intel</td>
<td>Do not change this value.</td>
</tr>
<tr>
<td>tpm.owner.secret</td>
<td>User specified or randomly generated, 20 hex-encoded bytes.</td>
<td>Password used to assert ownership of the host TPM. Inherited from TPM_OWNER_SECRET in trustagent.env if specified during installation of the Trust Agent. Randomly generated if not specified.</td>
</tr>
<tr>
<td>trustagent.tls.cert.dn</td>
<td>CN=trustagent</td>
<td>Defines the subject name of the Trust Agent's TLS certificate.</td>
</tr>
<tr>
<td>trustagent.tls.cert.sha384</td>
<td>SHA384 hash of the TLS certificate. Do not change this value.</td>
<td></td>
</tr>
<tr>
<td>aik.secret</td>
<td>Generated automatically.</td>
<td>20 hex-encoded bytes, Secret to generate Attestation Identity Key</td>
</tr>
<tr>
<td>binding.key.secret</td>
<td>20 hex-encoded bytes, Secret to generate Binding Key</td>
<td></td>
</tr>
<tr>
<td>hardware.uuid</td>
<td>Host hardware UUID. Do not change this value.</td>
<td></td>
</tr>
<tr>
<td>signing.key.secret</td>
<td>20 hex-encoded bytes, Secret to generate Signing key</td>
<td></td>
</tr>
<tr>
<td>tpm.srk.secret</td>
<td>0</td>
<td>Many tools assume this well-known SRK comprised of 20 zero bytes.</td>
</tr>
<tr>
<td>trustagent.keystore.password</td>
<td>Generated automatically. Example: 9JF7+HhpMUM_</td>
<td>The password used to access the trustagent.jks file with the keytool.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>trustagent.tls.cert.dns</td>
<td>Generated automatically Comma-separated list of all hostnames used by the host.</td>
<td>These names are added as subject alternative names on the Trust Agent's TLS certificate. By default, all names in /etc/hosts corresponding to local IP Addresses are used. If not specified, the installer performs a reverse DNS lookup for all IP Addresses found in the ifconfig output. In some environments this can cause a delay during installation. Manually specifying the subject names can eliminate this delay.</td>
</tr>
<tr>
<td>trustagent.tls.cert.ip</td>
<td>Generated automatically. Comma-separated list of IP Addresses.</td>
<td>These addresses are added as subject alternative names on the Trust Agent's TLS certificate. By default, all IP Addresses shown in the ifconfig output are used.</td>
</tr>
</tbody>
</table>

### 9.2.3 Command-Line Options

#### 9.2.3.1 Available Commands

**9.2.3.1.1 Help**

tagent help
Displays the list of available CLI commands.

**9.2.3.1.2 Start**

tagent start
Starts the services.

**9.2.3.1.3 Stop**

tagent stop
Stops the services.

**9.2.3.1.4 Restart**

tagent restart
Restarts the services.

**9.2.3.1.5 Status**

tagent status
Reports whether the service is currently running.
9.2.3.1.6 **Uninstall**

tagent uninstall

Uninstalls the service, including the deletion of all files and folders. See section 14.2 for additional information.

9.2.3.1.7 **Version**

tagent version

Reports the version of the service.

9.2.3.1.8 **Fingerprint**

tagent fingerprint

Displays the TLS certificate information.

9.2.3.1.9 **Java-detect**

tagent java-detect

Displays the detected path and installed version of Java.

9.2.3.1.10 **Zeroize**

tagent zeroize

Shreds all secrets, keys, and configurations.

9.2.3.1.11 **Password**

tagent password [username] [password] --permissions *:**

Creates a new user with the specified username and password. Because the Trust Agent does not have granular user permissions, the --permissions *:* is necessary and assigns all permissions to the created user.

9.2.3.1.12 **Export-config**

tagent export-config <outfile|--in=infle|--out=outfile|--stdout>

Exports the current configuration. Configuration settings are stored in the encrypted file /opt/trustagent/configuration/trustagent.properties; this command allows the configuration to be decrypted or output to the console.

9.2.3.1.13 **Config**

tagent config [key] [value]
9.2.3.14 Setup

tagent setup [--force|--noexec] [task1 task2 ...]
Re-runs the installation setup tasks, or the specific tasks listed.

9.2.4 Directory Layout

9.2.4.1 Windows

9.2.4.2 Linux

The Linux rust Agent installs by default to /opt/trustagent, with the following subfolders:
Bin
Configuration
Env.d
Features
Hypertext
Logs
Repository
Share
Var

9.3 Integration Hub

9.3.1 Installation Answer File

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTESTATION_HUB_PORT_HTTP</td>
<td>19082</td>
</tr>
<tr>
<td>ATTESTATION_HUB_PORT_HTTPS</td>
<td>19445</td>
</tr>
<tr>
<td>ATTESTATION_HUB_DB_NAME</td>
<td>attestation_hub_db</td>
</tr>
<tr>
<td>ATTESTATION_HUB_DB_HOSTNAME</td>
<td>localhost</td>
</tr>
<tr>
<td>ATTESTATION_HUB_DB_PORTNUM</td>
<td>5432</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>ATTESTATION_HUB_DB_DRIVER</td>
<td>org.postgresql.Driver</td>
</tr>
<tr>
<td>MTWILSON_SERVER</td>
<td>10.105.168.145</td>
</tr>
<tr>
<td>MTWILSON_SERVER_PORT</td>
<td>8443</td>
</tr>
<tr>
<td>MTWILSON_USERNAME</td>
<td>hubadmin</td>
</tr>
<tr>
<td>MTWILSON_PASSWORD</td>
<td>HubPassword</td>
</tr>
<tr>
<td>MTWILSON_TLS</td>
<td>30fa58947d64373ec2180d77d492c6d1f3c144368ea08d115bc2d3d8f9c2e</td>
</tr>
<tr>
<td>ATTESTATION_HUB_TENANT_CONFIGURATIONS_PATH</td>
<td>/opt/tenantconfig</td>
</tr>
<tr>
<td>ATTESTATION_HUB_POLL_INTERVAL</td>
<td>2</td>
</tr>
<tr>
<td>ATTESTATION_HUB_DB_USERNAME</td>
<td>root</td>
</tr>
<tr>
<td>ATTESTATION_HUB_DB_PASSWORD</td>
<td>dbpassword</td>
</tr>
</tbody>
</table>

### 9.3.2 Configuration Options

<table>
<thead>
<tr>
<th>Key</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jetty.tls.cert.dns</td>
<td>server.com,localhost</td>
<td>Subject Alternative Names</td>
</tr>
<tr>
<td>attestation-hub.db.schema</td>
<td>attestation_hub_pu</td>
<td>Defines the database schema name.</td>
</tr>
<tr>
<td>attestation-hub.poll.interval</td>
<td>2</td>
<td>Defines in minutes how long the Hub will wait before sending a new polling request to the Verification Service to retrieve new reports.</td>
</tr>
<tr>
<td>mtwilson.api.password</td>
<td>HubPassword</td>
<td>Defines the Verification Service user that will be used by the Integration Hub for API requests. This user must be created with at minimum the reports:search permission or the reports_manager or auditor roles.</td>
</tr>
<tr>
<td>mtwilson.api.url</td>
<td><a href="https://server.com:8443/mtwilson/v2">https://server.com:8443/mtwilson/v2</a></td>
<td>Defines the base URL for the Verification Service API.</td>
</tr>
<tr>
<td>javax.net.ssl.keyStore</td>
<td>/opt/attestation-hub/configuration/keystore.jks</td>
<td>Defines the SSL keystore file path.</td>
</tr>
<tr>
<td>mtwilson.extensions.packageExcludeFilter.startsWith</td>
<td>java,javax</td>
<td>Do not change this value.</td>
</tr>
<tr>
<td>mtwilson.username</td>
<td>hubadmin</td>
<td>Defines the Verification Service user that will be used by the Integration Hub for API requests. This user must be created with at minimum the reports:search permission or the reports_manager or auditor roles.</td>
</tr>
<tr>
<td>Key</td>
<td>Sample Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jetty.tls.cert.ip</td>
<td>192.168.1.1,127.0.0.1</td>
<td>Defines the IP addresses that are listed in the Integration Hub TLS Certificate as Subject Alternative Names. These addresses are used to validate connections to the Hub.</td>
</tr>
<tr>
<td>attestation-hub.db.password</td>
<td>dbpassword</td>
<td>Defines the database connection password.</td>
</tr>
<tr>
<td>attestation-hub.db.driver</td>
<td>org.postgresql.Driver</td>
<td>Defines the database connection driver. Do not change this value.</td>
</tr>
<tr>
<td>mtwilson.extensions.fileIncludeFilter.contains</td>
<td>mtwilson,attestation-hub</td>
<td>Do not change this value.</td>
</tr>
<tr>
<td>mtwilson.api.tls.policy.certificated.</td>
<td></td>
<td>20 hex-encoded bytes. Obtain this from the Verification Service server, in /opt/mtwilson/configuration/https.properties.</td>
</tr>
<tr>
<td>jetty.port</td>
<td>19082</td>
<td>Defines the webserver insecure (http) port.</td>
</tr>
<tr>
<td>mtwilson.server</td>
<td>hvs.server.com</td>
<td>Defines the IP address or hostname of the Verification Service from which the Hub will poll new attestation reports.</td>
</tr>
<tr>
<td>jetty.secure.port</td>
<td>19445</td>
<td>Defines the webserver secure (https) port.</td>
</tr>
<tr>
<td>attestation-hub.db.name</td>
<td>attestation_hub_pu</td>
<td>Defines the database schema name.</td>
</tr>
<tr>
<td>password.vault.file</td>
<td>/opt/attestation-hub/configuration/password-vault.jck</td>
<td>Information about keystore path, default would be configuration.</td>
</tr>
<tr>
<td>attestation-hub.db.username</td>
<td>root</td>
<td>Defines the database connection credentials.</td>
</tr>
<tr>
<td>mtwilson.extensions.packageIncludeFilter.startsWith</td>
<td>com.intel.org.glassfish.jersey.media.multipart</td>
<td>Do not change this value.</td>
</tr>
<tr>
<td>tenant.configuration.path</td>
<td>/opt/tenantconfig</td>
<td>Defines the path where tenant configurations will be stored. These are also stored in the database.</td>
</tr>
<tr>
<td>attestation-hub.db.portnum</td>
<td>5432</td>
<td>Defines the database connection port.</td>
</tr>
<tr>
<td>mtwilson.api.username</td>
<td>hubadmin</td>
<td>Defines the Verification Service credentials that the Hub will use to access the VS API.</td>
</tr>
<tr>
<td>jetty.tls.cert.dn</td>
<td>CN=Attestation Hub</td>
<td>Defines the Distinguished Name of the Hub TLS certificate.</td>
</tr>
<tr>
<td>mtwilson.password</td>
<td>HubPassword</td>
<td>Defines the Verification Service credentials that the Hub will use to access the VS API.</td>
</tr>
<tr>
<td>attestation-hub.db.hostname</td>
<td>localhost</td>
<td>Defines the database connection IP address or hostname.</td>
</tr>
<tr>
<td>password.vault.type</td>
<td>JCEKS</td>
<td>Defines password for keystore vault.</td>
</tr>
<tr>
<td>attestation-hub.db.url</td>
<td>jdbc:\postgresql:\localhost:\5432\attestation_hub_pu</td>
<td>Defines the complete database connection URL.</td>
</tr>
<tr>
<td>mtwilson.server.port</td>
<td>8443</td>
<td>Defines the Verification Service API port.</td>
</tr>
</tbody>
</table>
9.3.3 Command-Line Options

9.3.3.1 Available Commands

9.3.3.1.1 Help

attestation-hub help
Displays the list of available CLI commands.

9.3.3.1.2 Start

attestation-hub start
Starts the services.

9.3.3.1.3 Stop

mtwilson stop
Stops the services.

9.3.3.1.4 Restart

attestation-hub restart
Restarts the services.

9.3.3.1.5 Status

attestation-hub status
Reports whether the service is currently running.

9.3.3.1.6 Uninstall

attestation-hub uninstall [--purge]
Uninstalls the service, including the deletion of all files and folders. Database content is not removed. If the --purge option is used, database content will be removed during the uninstallation.

9.3.3.1.7 Version

attestation-hub version
Reports the version of the service.
9.3.3.1.8 Password

attestation-hub password [username] [password] --permissions *:*  
Creates a new user with the specified username and password. Because the Hub does not have granular user permissions, the --permissions *:* is necessary and assigns all permissions to the created user.

9.3.3.1.9 Export-config

attestation-hub export-config <outfile|--in=infile|--out=outfile|--stdout>

Exports the current configuration. Configuration settings are stored in the encrypted file /opt/mtwilson/configuration/mtwilson.properties; this command allows the configuration to be decrypted or output to the console.

9.3.3.1.10 Config

attestation-hub config [key] [value]

Configures a specified configuration setting to a specified value. Changing settings may require a service restart to take effect.

9.3.3.1.11 Setup

attestation-hub setup [--force|--noexec] [task1 task2 ...]

Re-runs the installation setup tasks, or the specific tasks listed.

9.3.4 Directory Layout

9.3.4.1 Logs
10 Certificate and Key Management

10.1 Verification Service Certificates and Keys

Each of the certificates and keys used by the Verification Service can be automatically replaced by deleting the certificate or key and running "mtwilson setup". This will re-run the installation setup tasks, which will detect that needed certificates are missing and create new ones. Restart the VS after replacing certificates in this way.

10.1.1 Root Certificate

The VS Root Certificate is used to sign the TLS, SAML and Asset tag certificates, and is generated during installation.

/opt/mtwilson/configuration/cacerts.pem
/opt/mtwilson/configuration/cakey.pem
/opt/mtwilson/configuration/MtWilsonRootCA.crt.pem

The Root Certificate can be replaced using the following command:

mtwilson replace-root-key-pair --private-key=key.pem --cert-chain=cert-chain.pem

This will:
- Replace key pair in /opt/mtwilson/configuration/cakey.pem
- Update /opt/mtwilson/configuration/cacerts.pem with cert chain
- Update /opt/mtwilson/configuration/MtWilsonRootCA.crt.pem with cert chain

This does not require a service restart. However, since the Root Certificate is used to sign the TLS, SAML, and Asset Tag certificates, changing the Root Certificate requires recreation of those certificates as well.

10.1.2 TLS Certificate

The TLS Certificate is created at installation time, and the key pair is stored in a keystore:

/opt/mtwilson/configuration/keystore.jks
When generating a new TLS key pair for the Verification Service, it is recommended to use the RSA algorithm with no restriction on size. The TLS certificate must contain Subject Alternative name entries for any resolvable hostname or IP address for hostname verification.

The TLS Certificate can be replaced with a user-specified keypair and certificate chain using the following command:

```
mtwilson replace-tls-key-pair --private-key=new.key.pem --cert-chain=new.cert-chain.pem
```

This will:

- Replace the key pair in /opt/mtwilson/configuration/keystore.jks, alias jetty.
- Updates the values of the following properties in mtwilson.properties:
  - jetty.tls.cert.dn
  - jetty.tls.cert.ip
  - jetty.tls.cert.dns

After replacing the Verification Service TLS certificate, the VS service will need to be restarted:

```
mtwilson restart
```

Additionally, all Trust Agent hosts will need to be updated to trust the new TLS certificate. On each Trust Agent host, update the value of the configuration setting `mtwilson.tls.cert.sha384` with the SHA384 hash of the new TLS certificate.

To retrieve the new SHA384 value, run "mtwilson fingerprint" on the VS.

To reconfigure the `mtwilson.tls.cert.sha384` value on a Trust Agent host, run the following commands:

```bash
tagent config mtwilson.tls.cert.sha384 <HVS TLS SHA384>
tagent setup download-mtwilson-tls-certificate
tagent restart
```

No action is required after changing the VS TLS certificate for VMWare hosts.

### 10.1.3 SAML

The SAML Certificate is used to sigh SAML attestation reports, and is itself signed by the Root Certificate.
The SAML Certificate can be replaced with a user-specified keypair and certificate chain using the following command:

```
mtwilson replace-saml-key-pair --private-key=new.key.pem --cert-chain=new.cert-chain.pem
```

This will:
- Replace key pair in `/opt/mtwilson/configuration/SAML.jks`, alias samlkey1
- Update `/opt/mtwilson/configuration/saml.crt` with saml DER public key cert
- Update `/opt/mtwilson/configuration/saml.crt.pem` with saml PEM public key cert
- Update configuration properties:
  - saml.key.password to null
  - saml.certificate.dn
  - saml.issuer

When the SAML certificate is replaced, all hosts will immediately be added to a queue to generate a new attestation report, since the old signing certificate is no longer valid. No service restart is necessary.

If the Integration Hub is being used, the new SAML certificate will need to be imported to the Hub.

### 10.1.4 Asset Tag

The Asset Tag Certificate is used to sign all Asset Tag Certificates.

```
/opt/mtwilson/configuration/tag-cacerts.pem
```

The Asset Tag Certificate can be replaced with a user-specified keypair and certificate chain using the following command:

```
mtwilson replace-tag-key-pair --private-key=new.key.pem --cert-chain=new.cert-chain.pem
```

This will:
- Replace key pair in database table `mw_file` (cakey is private and public key pem formatted, cacerts is cert chain)
- Update `/opt/mtwilson/configuration/tag-cacerts.pem` with cert chain
- Update configuration properties:
No service restart is needed. However, all existing Asset Tags will be considered invalid, and will need to be recreated. It is recommended to delete any existing Asset Tag certificates and Flavors, and then recreate and deploy new Tags.

10.1.5 Privacy CA

The Privacy CA certificate is used as part of the certificate chain for creating the Attestation Identity Key (AIK) during Trust Agent provisioning. The Privacy CA must be a self-signed certificate.

The Privacy CA certificate is used by Trust Agent nodes during Trust Agent provisioning; if the Privacy CA certificate is changed, all Trust Agent nodes will need to be re-provisioned.

```
/opt/mtwilson/configuration/PrivacyCA.p12
```
```
/opt/mtwilson/configuration/PrivacyCA.pem
```

The Privacy CA Certificate can be replaced with a user-specified keypair and certificate chain using the following command:

```
mtwilson replace-pca-key-pair --private-key=new.key.pem --cert-chain=new.cert-chain.pem
```

This will:

- Replace key pair in `/opt/mtwilson/configuration/PrivacyCA.p12`, alias 1
- Update `/opt/mtwilson/configuration/PrivacyCA.pem` with cert
- Update configuration properties:
  - `mtwilson.privacyca.aik.issuer`
  - `mtwilson.privacyca.aik.validity.days`

After the Privacy CA certificate is replaced, all Trust Agent hosts will need to be re-provisioned with a new AIK:

```
tagent setup download-mtwilson-privacy-ca-certificate --force
```
```
tagent setup request-aik-certificate --force
```
```
tagent restart
```

10.1.6 Endorsement CA

The Endorsement CA is a self-signed certificate used during Trust Agent provisioning.
The Endorsement CA Certificate can be replaced with a user-specified keypair and certificate chain using the following command:

```
mtwilson replace-eca-key-pair --private-key=new.key.pem --cert-chain=new.cert-chain.pem
```

This will:
- Replace key pair in `/opt/mtwilson/configuration/EndorsementCA.p12`, alias 1
- Update `/opt/mtwilson/configuration/EndorsementCA.pem` with accepted ECs
- Update configuration properties:
  - `mtwilson.privacyca.ek.issuer`
  - `mtwilson.privacyca.ek.validity.days`

After the Endorsement CA certificate is replaced, all Trust Agent hosts will need to be re-provisioned with a new Endorsement Certificate:

```
tagent setup request-endorsement-certificate --force
tagent restart
```

10.2 Trust Agent Certificates and Keys

10.2.1 TLS Certificate

The TLS Certificate is created at installation time, and the key pair is stored in a keystore:

```
/opt/trustagent/configuration/trustagent.jks
```

When generating a new TLS key pair for the Trust Agent, it is recommended to use the RSA algorithm with no restriction on size. The TLS certificate must contain Subject Alternative name entries for any resolvable hostname or IP address for hostname verification.

The TLS Certificate can be replaced with a user-specified keypair and certificate chain using the following command:

```
tagent replace-tls-key-pair --private-key=new.key.pem --cert-chain=new.cert-chain.pem
```

This will:
- Replace key pair in `/opt/mtwilson/configuration/trustagent.jks` alias tls
• Update configuration properties:
  — trustagent.tls.cert.sha384
  — trustagent.tls.cert.dn
  — trustagent.tls.cert.ip
  — trustagent.tls.cert.dns

The Trust Agent service will need to be restarted after replacing the TLS certificate.

10.2.2 Trust Agent Provisioning Process

aik.blob
aik.pem
endorsement.pem
trustagent.jks

10.3 Integration Hub Certificates and Keys

10.3.1 TLS Certificate

The TLS Certificate is created at installation time, and the key pair is stored in a keystore:

/opt/attestation-hub/configuration/keystore.jks

When generating a new TLS key pair for the Hub, it is recommended to use the RSA algorithm with no restriction on size. The TLS certificate must contain Subject Alternative name entries for any resolvable hostname or IP address for hostname verification.

The TLS Certificate can be replaced with a user-specified keypair and certificate chain using the following command:

attestation-hub replace-tls-key-pair --private-key=new.key.pem --cert-chain=new.cert-chain.pem

This will:
• Replace key pair in /opt/attestation-hub/configuration/keystore.jks, alias jetty
• Update configuration properties:
  — jetty.tls.cert.dn
— jetty.tls.cert.ip
— jetty.tls.cert.dns

The Integration Hub service will need to be restarted after replacing the TLS certificate.
11 High Availability

To maximize service availability in an environment where hypervisor-level high availability or fault tolerance solutions are impractical, unavailable, or insufficient, Intel recommends using a redundancy model featuring two (or more) Intel Verification Service instances running on separate hosts accessing the same external database with traffic directed by a reverse-proxy server.

Note: The shared database server should be configured for redundancy as well. However, database server configuration is beyond the scope of this document.

In this configuration, if one of the Verification Services becomes unavailable for any reason, the reverse-proxy automatically fails over to another remaining Verification Service. Since both instances are accessing a shared database, there is no loss of data in the case of a failover, and there is no manual intervention required after the failed server is brought back on-line.

The instructions below define a sample configuration using the NginX reverse proxy and two Verification Service instances. These instructions can be adapted to similar alternative solutions.

The key requirements for redundant Verification Services are:

- A shared database (whether this is external, or a replicated database across the Verification Service servers).
- Some form of reverse proxy or other method of sharing a single IP Address or hostname across all Verification Service instances.
- Identical keys, certificates, and other secrets on all Verification Service instances so that they can all use the same credentials to access the database, and so that the TLS certificate used by the web services remains identical. These include passwords and secrets defined in the various .properties files in the Verification Service configuration directory, all of the certificates and keys in that directory, the TLS certificate for the web server, and the configuration files for the web server.

11.1 Prerequisites

Following are the prerequisites:

- Separate database server (PostgreSQL is supported).
- An additional server running the NginX reverse-proxy service (other solutions are also possible).
11.2 Deployment Instructions

Following are the deployment instructions:

1. Deploy two (or more) separate Attestation Server instances on separate host servers.
   - Use a shared IP Address or hostname for the Attestation Server IP when setting up all Attestation Servers. The first installation performed configures the secrets, certificates, keys, and users that are later copied to all of the other instances. Subsequent installations throws errors during user creation, as they attempt to create users that already exist; this is normal and expected.

2. In the mtwilson.env installation answer file, configure the IP Address, port, and logon credentials for the external database server as well as the port and logon credentials and the database name.
   - The database schema and a valid user with rights over this database must be created on the database server prior to running the Verification Service installation, and must be configured to enable connections from all redundant servers. The Verification Service installer automatically creates the required database tables on the remote server.

3. Copy the entire /opt/mtwilson/configuration directory from the first/Primary VS to all other Verification Services.

4. Restart the Verification Service an all servers.

5. On the reverse-proxy server, run the following command:
   - $ apt-get install nginx

6. Create or modify the default.conf configuration file:
   - $ nano /etc/nginx/conf.d/default.conf
   - Edit the file as follows:

```bash
upstream mtwilson {
    server <Verification Service 1 IP>:<port> max_fails=1 fail_timeout=30s; server <Verification Service 2 IP>:<port> backup;
}

server {
listen <IP Address of NginX Server>:<port>; server_name <DNS Name of NginX Server>;

    access_log /var/log/nginx/ISecL-VS-HA.log; error_log /var/log/nginx/ISecL-VS-HA.error.log;
    ## send request back to mtwilson ##
    location / {
        proxy_pass http://mtwilson/;
        proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_504;
    }
```

The upstream section allows for the declaration of the Verification Services to be used with the reverse proxy, as well as configuration of when each of the servers is used. In this configuration, one server is set to be the primary, and a second server is set to be the backup.

**Note:** All traffic goes to the primary server, and the server labeled backup is only used if the other server has been flagged as failed. This is done to preserve session information.

A failure is triggered when an HTTP request times out. In this case, the `max_fails=1` variable sets a single timeout to trigger a failure. This can be set higher, but doing so increases the amount of time before a failure is detected and remediated. The `fail_timeout=30s` variable tells NginX to leave a server flagged as failed for 30 seconds before trying again. Increasing this value can improve performance slightly when a failover has occurred (as every new request more than 30 seconds since the last failed flag re-attempts the primary server and thus has to wait for a timeout), but also delay the automatic fail-back to the primary server once it is recovered.

As an alternative to the active/passive configuration, the ip_hash line can be added to the beginning of the upstream section. Note that if ip_hash is used, remove the max_fails, fail_timeout, and backup arguments. This causes NginX to direct incoming traffic to a Verification Service in the upstream list based on a hash of the requesting machine’s IP Address.

In this way, NginX can perform load balancing. Requests are divided evenly across all servers in the list, or the weight argument can be added to determine load ratios.

The `listen` and `server_name` variables tell NginX which specific IP and port to listen on and forward. Requests sent to other ports are not forwarded and receives a default NginX page. This can be configured to be a different page if desired. For more information, refer to the documentation for NginX at http://nginx.org/en/docs/.

The `proxy_pass` setting tells NginX to forward all requests matching the syntax `http://<listening IP/ name and port>/` to the Verification Service setting declared in the upstream section.

7. Restart the NginX service.

   `$ service nginx restart`

At this point, it should be possible to point a browser to the Attestation Server portal URLs, but substituting the Attestation Server IP/DNS name with the NginX server’s IP/DNS name. NginX automatically forwards the URLs to the appropriate Attestation Server, and the services can be used normally.
11.3 Failover

If a failover to the backup server should occur, there is a brief waiting period (approximately 5-10 seconds) as NginX waits for the timeout and flags the primary as failed.
12 TLS Policies

The Intel Security Libraries Verification Service validates the authenticity of connections through the use of various TLS verification policies.

12.1 TLS Policy Types

Intel Security Libraries Verification Service uses six types of TLS policies.

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Behavior</th>
<th>Shared</th>
<th>Per-Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate</td>
<td>The certificate policy requires one or more trusted certificates or CA certificates and only connects to a peer whose certificate either is a trusted certificate or is signed by a CA that is trusted. This policy type also performs hostname verification. <strong>Note:</strong> The remote server's hostname must be resolvable from the Mt. Wilson server.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Certificate Fingerprint</td>
<td>This policy stores the SHA384 hash of the certificate for validation rather than the entire certificate itself.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Public Key</td>
<td>The public key policy requires one public key parameter and only connects to a peer using that key. This is similar to SSH public key authentication of clients and hosts. Hostname verification is NOT performed when using Public Key TLS policies.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Public Key Fingerprint</td>
<td>This policy stores the SHA384 hash of the public key for validation rather than the public key itself.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TRUST_FIRST_CERTIFICATE</td>
<td>This policy stores the first certificate encountered when connecting to a host, and uses that certificate for all future TLS validation with that host.</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>INSECURE</td>
<td>This policy disables all TLS validation. All connections are accepted regardless of TLS certificates. This policy should be used for troubleshooting and development only, and should never be used in a production environment.</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

12.2 Policy Scope

TLS policies can be per-host or shared across multiple hosts.
12.2.1 Per-Host

A per-host TLS policy is an individual, per-host TLS policy. When the host is deleted, its per-host TLS policy is automatically deleted as well.

12.2.2 Shared

A shared TLS policy may be referenced by multiple host records. When a host that referenced a shared TLS policy is deleted, the shared policy continues to exist regardless if there are any remaining hosts that are referencing it. Shared policies must be explicitly deleted by the user.

The Verification Service requires a TLS policy to be defined for any remote host to which it connects. If no TLS policy is defined, or if the TLS information does not match the TLS policy, the connection fails.

12.3 Default Policy Selection

Any shared-scope policy can be defined as the "default" TLS policy for a given Verification Service environment. For example, if all TLS certificates for all hosts in the attestation environment have been signed by the same CA certificate, that CA certificate can be used to create a shared-scope certificate policy, and this same policy could be used to validate all TLS connections with all attested hosts. By configuring this policy as the default TLS policy, the Verification Service uses this specific policy for all hosts unless another policy is specified.

In the Verification Service UI, this mostly means that the default policy is automatically selected from the drop-down when registering hosts. From an API perspective, it means that, when calling a registration API, if no TLS policy is specifically defined in the call, the default TLS policy is used. Using a shared default policy that is valid across all hosts in the attestation environment can greatly simplify TLS policy and host management.

**Note:** During installation, the only two shared-scope policies that might be available are TRUST_FIRST_CERTIFICATE and INSECURE, and these only if they have actually been enabled. All other policies must be user-created after installation. To define a default TLS policy, edit the mtwilson.properties file and set the value of mtwilson.default.tls.policy.id to either the UUID or the name of the shared-scope TLS policy to be set as the default. Restart Mt. Wilson to affect the change.

12.4 Default TLS Policies

At the time the Verification Service is installed, two TLS policies are created.
12.4.1 **TRUST_FIRST_CERTIFICATE**

This policy creates a new TLS policy the first time that a new host is registered to the Verification Service, and uses that policy for all future interactions with that host.

12.4.2 **INSECURE**

This policy turned off all TLS certificate validation entirely (all connections were trusted, regardless of TLS certificates). This policy should only be used for development or troubleshooting, and should never be used in a production environment.

To configure the Verification Service to use **TRUST_FIRST_CERTIFICATE** as the default TLS Policy (and disallow the use of INSECURE), use the following settings:

- `mtwilson.tls.policy.allow=TRUST_FIRST_CERTIFICATE`
- `mtwilson.default.tls.policy.id=TRUST_FIRST_CERTIFICATE`

This can be done automatically during installation by setting the following variables in `mtwilson.env`:

- `export MTW_TLS_POLICY_ALLOW=TRUST_FIRST_CERTIFICATE`
- `export MTW_DEFAULT_TLS_POLICY_ID=TRUST_FIRST_CERTIFICATE`
13 User Management

13.1 Permissions

Each REST API in ISecL has specific permission requirements; these are defined for each API resource in the Javadoc. Permissions are defined as a domain:permission key/value pair.

For example, "hosts:create" defines a permission that would allow a user to create host objects. This would correspond to host registration.

Wildcards are also acceptable. The permissions "hosts:*" would allow all permissions on the hosts resource, allowing the user to create, update, and delete hosts.

Permissions can be used for users on the Verification Service, the Trust Agent, and the Integration Hub. However, permissions and users are not shared across these services. A user with permissions on the Verification Service will not have any permissions on the Integration Hub, for example.

13.2 Roles

Roles are collections of permissions that can be applied to a user. Roles are applicable only for the Verification Service.

Several roles are included by default.

<table>
<thead>
<tr>
<th>Role</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>trustagent_provisioner</td>
<td>host_aiks:certify</td>
</tr>
<tr>
<td></td>
<td>tpm_endorsements:create</td>
</tr>
<tr>
<td></td>
<td>tpm_endorsements:search</td>
</tr>
<tr>
<td></td>
<td>tpm_passwords:create</td>
</tr>
<tr>
<td></td>
<td>tpm_passwords:retrieve</td>
</tr>
<tr>
<td></td>
<td>tpm_passwords:search</td>
</tr>
<tr>
<td></td>
<td>tpm_passwords:store</td>
</tr>
<tr>
<td></td>
<td>tpm:endorse</td>
</tr>
<tr>
<td></td>
<td>host_signing_key_certificates:create</td>
</tr>
<tr>
<td></td>
<td>store_host_pre_registration_details:create</td>
</tr>
<tr>
<td>administrator</td>
<td><em>:</em></td>
</tr>
<tr>
<td>auditor</td>
<td>*:search,retrieve</td>
</tr>
<tr>
<td>asset_tag_manager</td>
<td>tag_certificate_requests:*</td>
</tr>
<tr>
<td></td>
<td>tag_selection_kv_attributes:*</td>
</tr>
<tr>
<td></td>
<td>tag_certificates:*</td>
</tr>
<tr>
<td></td>
<td>tag_kv_attributes:*</td>
</tr>
<tr>
<td></td>
<td>tag_selections:*</td>
</tr>
<tr>
<td>Role</td>
<td>Permissions</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>flavor_manager</td>
<td>host_tls_policies:create, search, retrieve</td>
</tr>
<tr>
<td></td>
<td>flavorgroup:*</td>
</tr>
<tr>
<td></td>
<td>flavors:*</td>
</tr>
<tr>
<td>host_manager</td>
<td>hosts:*</td>
</tr>
<tr>
<td></td>
<td>host_attestations:*</td>
</tr>
<tr>
<td></td>
<td>host_tls_policies:search, retrieve</td>
</tr>
<tr>
<td></td>
<td>host_status:search, retrieve</td>
</tr>
<tr>
<td>reports_manager</td>
<td>reports:*</td>
</tr>
<tr>
<td>host_unique_flavor_creator</td>
<td>host_unique_flavors:create</td>
</tr>
<tr>
<td>host_based_flavor_creator</td>
<td>host_based_flavors:create</td>
</tr>
</tbody>
</table>

Custom roles can be created using REST APIs:

1. Create a role:

   POST https://host-verification-server.com:8443/mtwilson/v2/roles

   Input: {"role_name":"role_sample"}

2. Assign permissions to the role:

   https://host-verification-server.com:8443/mtwilson/v2/roles/<role ID>/permissions

   Input: {"permit_domain":"hosts", "permit_action":"create,delete", "permit_selection":"*"}

Roles can now be assigned using the new custom role.

13.3 Creating a User for HTTP Basic Authentication

User creation can be performed via the command line (for all ISeCL services),
or via REST API (Verification Service only).

**Note:** ISeCL does not currently integrate with other authentication providers like Keystone or LDAP.

ISecL passwords do not expire automatically, and cannot be changed. To change a password, delete the existing password for the user, and create a new password.

**Note:** Integration Hub and Trust Agent users cannot be deleted without reinstalling.
13.3.1 Creating a User – CLI

13.3.1.1 Verification Service

mtwilson login-password <username> <password> --permissions <permission1>,<permission2>,...

13.3.1.2 Trust Agent

tagent password <username> <password> --permissions *:*

13.3.1.3 Integration Hub

attestation-hub password <username> <password> --permissions *:*

13.3.2 Creating a User – REST API

Creating a user via the REST API involves three steps:
1. Creating a new user request
2. Creating a new user password
3. Approving the new user password and applying permissions

Note: A single username can have more than one password. Permissions and roles are attached to the password, so a given user could have one password that allows only "hosts:search", but another password that allows "hosts:*".

1. Creating a new user request

   POST https://host-verification-server.com:8443/mtwilson/v2/users
   
   Input: {"username": "User1", "locale": "en-US", "comment": "Access needed for Project1"}

2. Creating a new user password

   The "password_hash" value should be the SHA384 digest of the password in base64 encoded format.
   The "salt" should be a random 8 bytes in base64 encoded format.
   The "iterations" value determines the number of iterations used to generate the password_hash from the original password. If the password was hashed only once, this value should be set to "1".
   Currently only the SHA256 algorithm is supported.

   POST https://host-verification-server.com:8443/mtwilson/v2/users/<userID>/login-password
   
   Input: {"password_hash": "RZMrrSt/PvKvdsiXpR01dBd8d6vF4KbKv7sF+oDg=", "salt": "agHma0UHF+=", "iterations": 1, "algorithm": "SHA256", "comment": "Access needed for development"}
3. Approving the new user password and applying permissions

New user passwords cannot be used until approved by an administrator. The same call used to approve a new user/password request is used to assign permissions to that user by applying one or more roles. Permissions cannot be directly applied via this API. To use a set of permissions not provided by an existing role, create a new custom role with the needed permissions, and use that role for the user password.

https://host-verification-server.com:8443/mtwilson/v1/users/<User ID>/login-passwords/<Password ID>

Input: `{"status":"APPROVED","enabled":true,"roles": ["<Role 1>", "<Role 2>"]}

13.4 Certificate Authentication

As an alternative to HTTP Basic authentication, the ISeCL Verification Service also supports certificate-based authentication when using the ISeCL Verification Service API Client.

Certificate authentication requires a user-generated keypair. The public key is registered to the Verification Service in x509 certificate form and associated to a user. The login certificate can then be activated and assigned roles by an administrator.

**Note:** A single user may have any number of login passwords and login certificates, each with their own permissions. For example, a user may have additional permissions when logging on using a secure API client using certificate authentication, but have read-only access to fewer resources when logging on with a password via a browser.

ISeCL certificate authentication is a custom implementation. Requests using certificate authentication must include the following header:

`authorization: x509 <string of signed data>`

The signed data includes informative request data and the base64 encoded signature of the request body and all headers. The body of the request and all headers are concatenated into a string and then signed using the user’s private key. The following must be included in the signed data string:

- HTTP method (GET, POST, etc.)
- Request URL
- Shiro authentication realm
- The Base64 encoded sha-384 fingerprint of the login certificate containing the RSA public key
- Full request body (if any)
- Signature algorithm (Currently SHA256withRSA or RSA-SHA256 are supported)
- HTTP headers
- A 24-byte X-Nonce consisting of 8 bytes of the current time in milliseconds and 16 bytes of random data
- The current date in the following format: “EEE, d MMM yyyy HH:mm:ss z”

**Note:** The ISeCL Verification Service API Client is required for using certificate authentication due to the custom implementation.

### 13.4.1 Creating a Login Certificate

1. Creating a new user request

   ```
   POST https://host-verification-server.com:8443/mtwilson/v2/users
   ```

   Input: `{"username":"User1","locale":"en-US","comment":"Access needed for Project1"}`

2. Creating a new user login certificate

   The user should create a new RSA keypair, and include the public key in x509 format in the “certificate” field of the request body.

   ```
   POST https://host-verification-server.com:8443/mtwilson/v2/users/<user ID>/login-certificates
   ```

   Input: `{ "certificate":"MIICrzCCAZegAwIB.....LX+ukpXKQ0dgiSfK+Yw==", "comment":"Need to manage user accounts."`}

3. Approving the new user password and applying permissions

   New user passwords cannot be used until approved by an administrator.

   The same call used to approve a new user/certificate request is used to assign permissions to that user by applying one or more roles. Permissions cannot be directly applied via this API. To use a set of permissions not provided by an existing role, create a new custom role with the needed permissions, and use that role for the user password.

   ```
   ```

   Input: `{"status":"APPROVED","enabled":true,"roles":["<Role 1>","<Role 2>"],}`
14 Uninstallation

This section describes steps used for uninstalling Intel SecL-DC services.

**Note:** This section does not apply for containerized deployments. To uninstall a containerized deployment, simply shut down the container and delete the persistence volumes.

14.1 Verification Service

To uninstall the Verification Service, run the following command:

```bash
mtwilson uninstall
```

Removes following directories:
1. `usr/local/bin/mtwilson`
2. `$MTWILSON_HOME/bin`
3. `$MTWILSON_HOME/java`
4. `$MTWILSON_HOME/features`

```bash
mtwilson uninstall --purge
```

Removes following directories:
1. `$MTWILSON_HOME /opt/mtwilson`
2. `$MTWILSON_CONFIGURATION $MTWILSON_HOME/configuration (/opt/mtwilson/configuration)`
3. `$MTWILSON_LOGS $MTWILSON_HOME/logs (/opt/mtwilson/logs)`

The `mtwilson uninstall` command will not delete any database content. To completely uninstall and delete all database content and user data, run the following:

```bash
mtwilson erase-data

mtwilson erase-users --all

mtwilson uninstall --purge
```

**Note:** The uninstall command must be issued last, because the uninstall process removes the scripts that execute the other commands, along with all database connectivity info.
### 14.2 Trust Agent

To uninstall the Trust Agent, run the following command:

tagent uninstall

Backs up the configuration directory and removes all Trust Agent files, except for configuration files which are saved and restored.

Removes following directories:
1. `/usr/local/bin/tagent`
2. `TRUSTAGENT_HOME : /opt/trustagent`
3. `/opt/tbootxm`
4. `/var/log/trustagent/measurement.*`

**Note:** TPM ownership can be preserved by retaining the TPM owner secret. If the Operating System will also be cleared, Linux systems will also require the `/usr/local/var/lib/tpm/system.data` file to be preserved. This file must be preserved from after ownership is taken, and then replaced after the OS reload before the Trust Agent attempts to reassert ownership.

If the ownership secret and/or `system.data` file are not preserved, reinstallation will require clearing TPM ownership.

### 14.3 Integration Hub

To uninstall the Integration Hub, run the following command:

`attestation hub uninstall`

Removes the following directories:
1. `/usr/local/bin/attestation-hub`
2. `/usr/bin/attestation-hub`
3. `/opt/attestation-hub`
4. `/etc/logrotate.d/attestation-hub`

`attestation-hub uninstall --purge`

Removes the following directories (in addition to directories removed without the --purge option):
1. Drops the database
2. Drops the user
3. Removes attestation hub tenant configuration path
## 15.1 PCR Definitions

### 15.1.1 Microsoft Windows Server 2016 Datacenter

#### 15.1.1.1 TPM 1.2 and 2.0

<table>
<thead>
<tr>
<th>PCR</th>
<th>Measurement Parameters</th>
<th>Description</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>BIOS ROM and Flash Image</td>
<td>This PCR is based solely on the BIOS version, and remains identical across all hosts using the same BIOS. This PCR is used as the PLATFORM Flavor.</td>
<td>All</td>
</tr>
<tr>
<td>12</td>
<td>Data events and highly volatile events</td>
<td>This PCR measures some of the modules which has boot counters in it. It changes on every boot and resume (Microsoft Windows ONLY; do not use for attestation as the values change on reboot)</td>
<td>Microsoft Windows Server</td>
</tr>
<tr>
<td>13</td>
<td>Boot Module Details</td>
<td>This PCR remains static except major changes such as kernel module update, different device driver for different OEM servers, etc. (Microsoft Windows ONLY)</td>
<td>Microsoft Windows Server</td>
</tr>
<tr>
<td>14</td>
<td>Boot Authorities</td>
<td>Used to record the Public keys of authorities that sign OS components. Expected not to change often. (Microsoft Windows ONLY)</td>
<td>Microsoft Windows Server</td>
</tr>
</tbody>
</table>

### 15.1.2 Red Hat Enterprise Linux

#### 15.1.2.1 TPM 2.0

<table>
<thead>
<tr>
<th>PCR</th>
<th>Measurement Parameters</th>
<th>Description</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>BIOS ROM and Flash Image Initial Boot Block (Intel® BootGuard only)</td>
<td>This PCR is based solely on the BIOS version, and remains identical across all hosts using the same BIOS. This PCR is used as the PLATFORM Flavor. Intel® BootGuard only): Extends measurements based on the Intel® BootGuard profile configuration and production vs non-production ACM flags; ACM signature; BootGuard key manifest hash; Boot Policy Manifest Signature</td>
<td>All</td>
</tr>
<tr>
<td>7</td>
<td>Intel® BootGuard configuration and profiles</td>
<td>Describes the success of the IBB measurement event.</td>
<td>All (Intel® BootGuard only)</td>
</tr>
<tr>
<td>PCR</td>
<td>Measurement Parameters</td>
<td>Description</td>
<td>Operating System</td>
</tr>
<tr>
<td>-----</td>
<td>------------------------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>PCR 0</td>
<td>BIOS ROM and Flash Image</td>
<td>This PCR is based solely on the BIOS version, and remains identical across all hosts using the same BIOS. This PCR is used as the PLATFORM Flavor.</td>
<td>• All</td>
</tr>
</tbody>
</table>
| PCR 17 | ACM | This PCR measures the SINIT ACM, and is hardware platform-specific. This PCR is part of the PLATFORM Flavor. | • VMware ESXi  
• Red Hat Enterprise Linux |
| PCR 18 | MLE [Tboot + VMM] | This PCR measures the tboot and hypervisor version. In ESXi hosts, only the tboot version is measured. | • VMware ESXi  
• Red Hat Enterprise Linux |
### 15.1.3.2 TPM 2.0

VMware supports TPM 2.0 with Intel TXT starting in vSphere 6.7 Update 1. Earlier versions will support TPM 1.2 only.

<table>
<thead>
<tr>
<th>PCR</th>
<th>Measurement Parameters</th>
<th>Description</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCR 0</td>
<td>BIOS ROM and Flash Image</td>
<td>This PCR is based solely on the BIOS version, and remains identical across all hosts using the same BIOS. This PCR is used as part of the PLATFORM flavor.</td>
<td>All</td>
</tr>
<tr>
<td>PCR 17</td>
<td>ACM</td>
<td>This PCR measures the SINIT ACM, and is hardware platform-specific. This PCR is part of the PLATFORM Flavor.</td>
<td>VMware ESXi • Red Hat Enterprise Linux</td>
</tr>
<tr>
<td>PCR 18</td>
<td>MLE [Tboot +VMM]</td>
<td>This PCR measures the tboot and hypervisor version. In ESXi hosts, only the tboot version is measured. This PCR is part of the PLATFORM Flavor.</td>
<td>VMware ESXi • Red Hat Enterprise Linux</td>
</tr>
<tr>
<td>PCR 19</td>
<td>OS Specific. • ESX and Trust Agent — non Kernel modules • Citrix Xen — OS • + Init RD + UUID</td>
<td>For ESXi this PCR contains individual measurements of all of the non-Kernel modules – this includes all of the VIBs installed on the ESXi host. This is part of the OS flavor. Note that two ESXi hosts with the same version of ESXi installed may require different OS flavors if different VIBs are installed.</td>
<td>VMware ESXi • Red Hat Enterprise Linux</td>
</tr>
<tr>
<td>PCR 20</td>
<td>For ESXi only, VM Kernel and VMK Boot</td>
<td>This PCR is used only by ESXi hosts for some host-specific measurements, and is part of the host-unique flavor.</td>
<td>VMware ESXi</td>
</tr>
<tr>
<td>PCR</td>
<td>Measurement Parameters</td>
<td>Description</td>
<td>Operating System</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>PCR 22 Asset Tag</td>
<td>Asset Tag is not currently supported for TPM 2.0 with ESXi.</td>
<td></td>
<td>VMware ESXi</td>
</tr>
</tbody>
</table>

### A.1 Attestation Rules

<table>
<thead>
<tr>
<th>Platform</th>
<th>TPM</th>
<th>Flavor Type</th>
<th>Rules to be verified</th>
<th>Comments</th>
</tr>
</thead>
</table>
| RHEL     | 1.2 | HARDWARE   | PcrMatchesConstant rule for PCR 0  
PcrMatchesConstant rule for PCR 17 | For all flavor evaluations, verification of the AIK certificate is needed. |
|          |     | OS         | PcrMatchesConstant rule for PCR 18  
PcrEventLogIncludes rule for PCR 19  
PcrEventLogIntegrity rule for PCR 19 | Evaluation of PcrEventLogIncludes would not include the host specific module verification.  
Even though PCR 19 only contains host specific modules, it is still needed in OS flavor for integrity check. |
|          |     | ASSET_TAG  | AssetTagMatches rule | AssetTagMatches rule needs to be updated to verify the key-value pairs after verifying the tag certificate. |
|          |     | HOST_SPECIFIC | PcrEventLogIncludes rule for PCR 19 | Only for the host specific modules. Only PCR 19 data is needed in this flavor. |
| RHEL     | 2.0 | HARDWARE   | PcrMatchesConstant rule for PCR 0  
PcrEventLogIncludes rule for PCR 17  
(LCP_DETAILS_HASH, BIOSAC_REG_DATA, OSSINITDATA_CAP_HASH, STM_HASH, MLE_HASH, NV_INFO_HASH, tb_policy, CPU_SCRMT_STAT, HASH_START, LCP_CONTROL_HASH)  
PcrEventLogIntegrity rule for PCR 17 | Evaluation of PcrEventLogIncludes would not include initrd and vmlinuz modules. They would be handled in host_specific flavor.  
Evaluation of PcrEventLogIntegrity rule would also include OS modules (initrd & vmlinuz) |
|          |     | OS         | PcrEventLogIntegrity rule for PCR 17 | |
|          |     | ASSET_TAG  | AssetTagMatches rule | |
|          |     | HOST_SPECIFIC | PcrEventLogIncludes rule for PCR 17  
(initrd & vmlinuz) | |
<table>
<thead>
<tr>
<th>Platform</th>
<th>TPM</th>
<th>Flavor Type</th>
<th>Rules to be verified</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMware ESXi</td>
<td>1.2</td>
<td>PLATFORM</td>
<td>PcrMatchesConstant rule for PCR 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrMatchesConstant rule for PCR 17</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
<td>PcrMatchesConstant rule for PCR 18</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrMatchesConstant rule for PCR 20</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrEventLogEqualsExcluding rule for PCR 19 (excludes dynamic modules based on component name)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrEventLogIntegrity rule for PCR 19</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASSET_TAG</td>
<td>PcrMatchesConstant rule for PCR 22</td>
<td></td>
</tr>
<tr>
<td>VMware ESXi</td>
<td>2.0</td>
<td>NOT SUPPORTED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>1.2</td>
<td>PLATFORM</td>
<td>PcrMatchesConstant rule for PCR 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
<td>PcrMatchesConstant rule for PCR 13</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrMatchesConstant rule for PCR 14</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASSET_TAG</td>
<td>AssetTagMatches rule</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>2.0</td>
<td>PLATFORM</td>
<td>PcrMatchesConstant rule for PCR 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
<td>PcrMatchesConstant rule for PCR 13</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PcrMatchesConstant rule for PCR 14</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASSET_TAG</td>
<td>AssetTagMatches rule</td>
<td>AssetTagMatches rule needs to be updated to verify the key-value pairs after verifying the tag certificate.</td>
</tr>
</tbody>
</table>

**A.2 Intel TXT and the Trusted Boot Process**