Automating responses to integrity validation failures

Learn how to use a Cloud Functions trigger to automatically act on Shielded VM integrity monitoring (https://cloud.google.com/compute/docs/instances/integrity-monitoring) events.

Overview

Integrity monitoring collects measurements from Shielded VM instances and surfaces them in Stackdriver Logging. If integrity measurements change across boots of a Shielded VM instance, integrity validation fails. This failure is captured as a logged event, and is also raised in Stackdriver Monitoring.

Sometimes, Shielded VM integrity measurements change for a legitimate reason. For example, a system update might cause expected changes to the operating system kernel. Because of this, integrity monitoring lets you prompt a Shielded VM instance to learn a new integrity policy baseline in the case of an expected integrity validation failure.

In this tutorial, you'll first create a simple automated system that shuts down Shielded VM instances that fail integrity validation:

1. Export (https://cloud.google.com/logging/docs/export/configure_export_v2) all integrity monitoring events to a Pub/Sub topic.
2. Create a Cloud Functions trigger (https://cloud.google.com/functions/docs/concepts/events-triggers#triggers) that uses the events in that topic to identify and shut down Shielded VM instances that fail integrity validation.

Next, you can optionally expand the system so that it prompts Shielded VM instances that fail integrity validation to learn the new baseline if it matches a known good measurement, or to shut down otherwise:

1. Create a Firestore database to maintain a set of known good integrity baseline measurements.
2. Update the Cloud Functions trigger so that it prompts Shielded VM instances that fail integrity validation to learn the new baseline if it is in the database, or else to shut down.

If you choose to implement the expanded solution, use it in the following way:

1. Each time there is an update that is expected to cause validation failure for a legitimate reason, run that update on a single Shielded VM instance in the instance group.
2. Using the late boot event from the updated VM instance as a source, add the new policy baseline measurements to the database by creating a new document in the **known_good_measurements** collection. See Creating a database of known good baseline measurements (#create-database) for more information.
3. Update the remaining Shielded VM instances. The trigger prompts the remaining instances to learn the new baseline, because it can be verified as known good. See Updating the Cloud Functions trigger to learn known good baseline (#trigger-learn-baseline) for more information.

Prerequisites

- Use a project that has Firestore in Native mode selected as the database service. You make this selection when you create the project, and it can't be changed. If your project doesn't use Firestore in Native mode, you will see the message "This project uses another database service" when you open the Firestore console.
- Have a Compute Engine Shielded VM instance in that project to serve as the source of integrity baseline measurements. The Shielded VM instance must have been restarted at least once.
- Have the `gcloud` command-line tool installed (https://cloud.google.com/sdk/docs/#install_the_latest_cloud_sdk_version_cloudsdk_current_version)
.
- Enable the Stackdriver Logging and Cloud Functions APIs by following these steps:
  1. GO TO APIS & SERVICES (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/APIS)
  2. See if **Cloud Functions API** and **Stackdriver Logging API** appear on the **Enabled APIs and services** list.
  3. If either of the APIs don't appear, click **Add APIs and Services**.
  4. Search for and enable the APIs, as needed.

Exporting integrity monitoring log entries to a Pub/Sub topic

Use Logging to export all integrity monitoring log entries generated by Shielded VM instances to a Pub/Sub topic. You use this topic as a data source for a Cloud Functions trigger to automate responses to integrity monitoring events.

1. GO TO STACKDRIVER LOGGING (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/LOGS)
2. Click the drop-down arrow on the right side of **Filter by label or text search**, and then click **Convert to advanced filter**.
3. Type the following advanced filter:

   ```
   resource.type="gce_instance" AND logName:  "projects/YOUR_PROJECT_ID/logs/comp
   ```

   replacing `YOUR_PROJECT_ID` with the ID of your project. Note that there are two spaces after `logName:`.
4. Click **Submit Filter**.
5. Click on **Create Export**.
6. For **Sink Name**, type **integrity-monitoring**.
7. For **Sink Service**, select **Cloud Pub/Sub**.
8. Click the drop-down arrow on the right side of **Sink Destination**, and then click **Create new Cloud Pub/Sub topic**.
9. For **Name**, type **integrity-monitoring** and then click **Create**.
10. Click **Create Sink**.

Creating a Cloud Functions trigger to respond to integrity failures

Create a Cloud Functions trigger that reads the data in the Pub/Sub topic and that stops any Shielded VM instance that fails integrity validation.

1. The following code defines the Cloud Functions trigger. Copy it into a file named `main.py`.

   ```
   import base64
   import json
   ```

```
import googleapiclient.discovery

def shutdown_vm(data, context):
    """A Cloud Function that shuts down a VM on failed integrity check."""
    log_entry = json.loads(base64.b64decode(data['data']).decode('utf-8'))
    payload = log_entry.get('jsonPayload', {})
    entry_type = payload.get('@type')
    if entry_type != 'type.googleapis.com/cloud_integrity.IntegrityEvent':
      raise TypeError("Unexpected log entry type: %s" % entry_type)

    report_event = (payload.get('earlyBootReportEvent')
        or payload.get('lateBootReportEvent'))

    if report_event is None:
      # We received a different event type, ignore.
      return

    policy_passed = report_event['policyEvaluationPassed']
    if not policy_passed:
      print('Integrity evaluation failed: %s' % report_event)
      print('Shutting down the VM')

      instance_id = log_entry['resource']['labels']['instance_id']
      project_id = log_entry['resource']['labels']['project_id']
      zone = log_entry['resource']['labels']['zone']

      # Shut down the instance.
      compute = googleapiclient.discovery.build(
          'compute', 'v1', cache_discovery=False)

      # Get the instance name from instance id.
      list_result = compute.instances().list(
          project=project_id,
          zone=zone,
              filter='id eq %s' % instance_id).execute()
      if len(list_result['items']) != 1:
        raise KeyError('unexpected number of items: %d'
            % len(list_result['items']))
      instance_name = list_result['items'][0]['name']

      result = compute.instances().stop(project=project_id,
          zone=zone,
          instance=instance_name).execute()
      print('Instance %s in project %s has been scheduled for shut down.'
          % (instance_name, project_id))
```

2. In the same location as `main.py`, create a file named `requirements.txt` and copy in the following dependencies:

```
google-api-python-client==1.6.6
google-auth==1.4.1
google-auth-httplib2==0.0.3
```

3. Open a terminal window and navigate to the directory containing `main.py` and `requirements.txt`.
4. Run the `gcloud beta functions deploy` command (https://cloud.google.com/sdk/gcloud/reference/beta/functions/deploy) to deploy the trigger:

```
gcloud beta functions deploy shutdown_vm --project YOUR_PROJECT_ID \
    --runtime python37 --trigger-resource integrity-monitoring \
    --trigger-event google.pubsub.topic.publish
```

replacing `YOUR_PROJECT_ID` with the ID of your project.

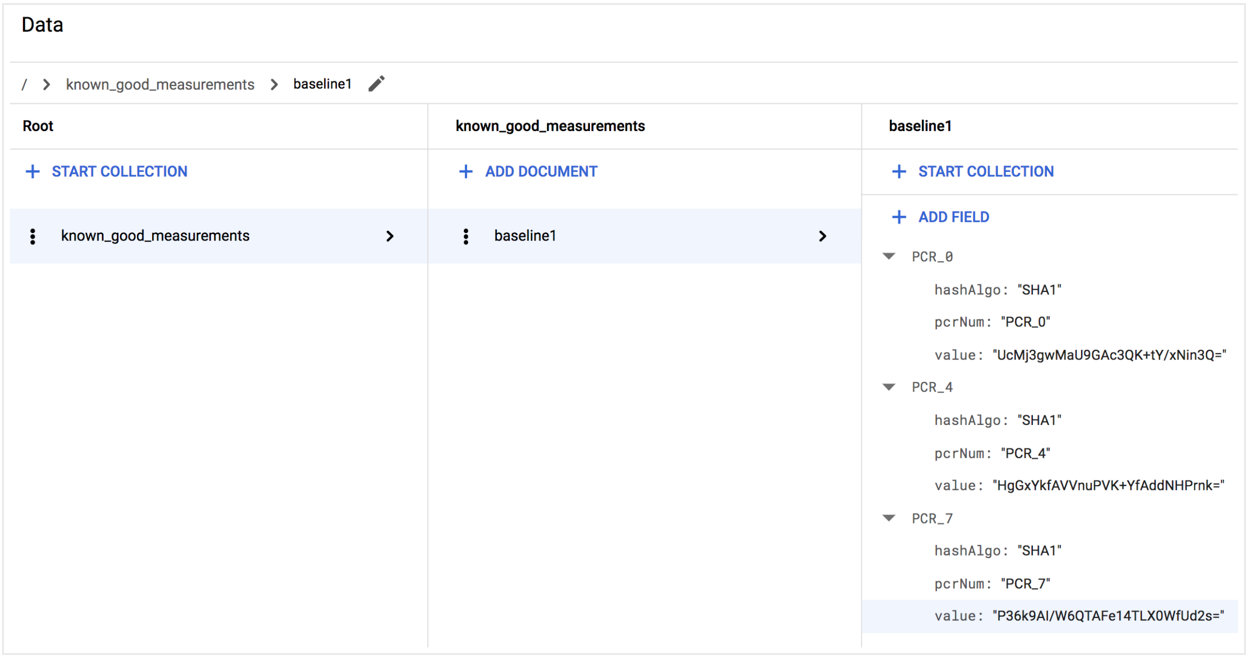
Creating a database of known good baseline measurements

Create a Firestore database to provide a source of known good integrity policy baseline measurements. You must manually add baseline measurements to keep this database up to date.

1. GO TO THE VM INSTANCES PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/COMPUTE/INSTANCES
2. Click the Shielded VM instance ID to open the **VM instance details** page.
3. Under **Logs**, click on **Stackdriver Logging**.
4. Locate the most recent `lateBootReportEvent` log entry.
5. Expand the log entry > `jsonPayload` > `lateBootReportEvent` > `policyMeasurements`.
6. Note the values for the elements contained in `lateBootReportEvent` > `policyMeasurements`.
7. GO TO THE FIRESTORE CONSOLE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FIRESTORE/DATA)
8. Choose **Start collection**.

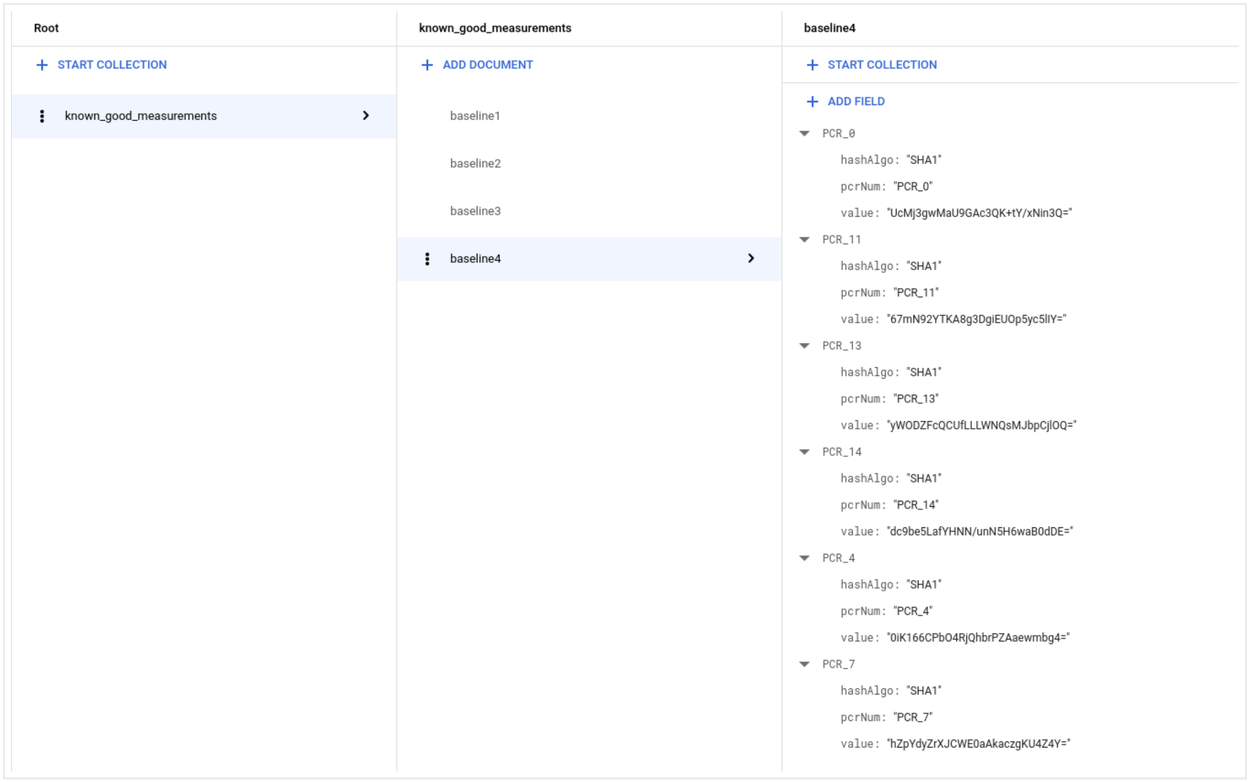
9. For **Collection ID**, type **known_good_measurements**.
10. For **Document ID**, type **baseline1**.
11. For **Field name**, type the **pcrNum** field value from element `0` in `lateBootReportEvent` > `policyMeasurements`.
12. For **Field type**, select **map**.
13. Add three string fields to the map field, named **hashAlgo**, **pcrNum**, and **value**, respectively. Make their values the values of the element `0` fields in `lateBootReportEvent` > `policyMeasurements`.
14. Create more map fields, one for each additional element in `lateBootReportEvent` > `policyMeasurements`. Give them the same subfields as the first map field. The values for those subfields should map to those in each of the additional elements.

    For example, if you are using a Linux VM, the collection should look similar to the following when you are done:



```
PCR_0
    hashAlgo: "SHA1"
    pcrNum: "PCR_0"
    value: "UcMj3gwMaU9GAc3QK+tY/xNin3Q="
PCR_4
    hashAlgo: "SHA1"
    pcrNum: "PCR_4"
    value: "HgGxYkfAVVnuPVK+YfAddNHPrnk="
PCR_7
    hashAlgo: "SHA1"
    pcrNum: "PCR_7"
    value: "P36k9AI/W6QTAFe14TLX0WfUd2s="
```

If you are using a Windows VM, you will see more measurements thus the collection should look similar to the following:



```
PCR_0
    hashAlgo: "SHA1"
    pcrNum: "PCR_0"
    value: "UcMj3gwMaU9GAc3QK+tY/xNin3Q="
PCR_11
    hashAlgo: "SHA1"
    pcrNum: "PCR_11"
    value: "67mN92YTKA8g3DgiEUOp5yc5lIY="
PCR_13
    hashAlgo: "SHA1"
    pcrNum: "PCR_13"
    value: "yWODZFcQCUfLLLWNQsMJbpCjlOQ="
PCR_14
    hashAlgo: "SHA1"
    pcrNum: "PCR_14"
    value: "dc9be5LafYHNN/unN5H6waB0dDE="
PCR_4
    hashAlgo: "SHA1"
    pcrNum: "PCR_4"
    value: "0iK166CPbO4RjQhbrPZAaewmbg4="
PCR_7
    hashAlgo: "SHA1"
    pcrNum: "PCR_7"
    value: "hZpYdyZrXJCWE0aAkaczgKU4Z4Y="
```

Updating the Cloud Functions trigger to learn known good baseline

1. The following code creates a Cloud Functions trigger that causes any Shielded VM instance that fails integrity validation to learn the new baseline if it is in the database of known good measurements, or else shut down. Copy this code and use it to overwrite the existing code in `main.py`.

```
import base64
import json
import googleapiclient.discovery

import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore

PROJECT_ID = 'YOUR_PROJECT_ID'

firebase_admin.initialize_app(credentials.ApplicationDefault(), {
    'projectId': PROJECT_ID,
})
```

```
def pcr_values_to_dict(pcr_values):
  """Converts a list of PCR values to a dict, keyed by PCR num"""
  result = {}
  for value in pcr_values:
    result[value['pcrNum']] = value
  return result

def instance_id_to_instance_name(compute, zone, project_id, instance_id):
  list_result = compute.instances().list(
      project=project_id,
      zone=zone,
      filter='id eq %s' % instance_id).execute()
  if len(list_result['items']) != 1:
    raise KeyError('unexpected number of items: %d'
        % len(list_result['items']))
  return list_result['items'][0]['name']

def relearn_if_known_good(data, context):
    """A Cloud Function that shuts down a VM on failed integrity check.
    """
    log_entry = json.loads(base64.b64decode(data['data']).decode('utf-8'))
    payload = log_entry.get('jsonPayload', {})
    entry_type = payload.get('@type')
    if entry_type != 'type.googleapis.com/cloud_integrity.IntegrityEvent':
      raise TypeError("Unexpected log entry type: %s" % entry_type)

    # We only send relearn signal upon receiving late boot report event: if
    # early boot measurements are in a known good database, but late boot
    # measurements aren't, and we send relearn signal upon receiving early boo
    # report event, the VM will also relearn late boot policy baseline, which
    # don't want, because they aren't known good.
    report_event = payload.get('lateBootReportEvent')
    if report_event is None:
      return

    evaluation_passed = report_event['policyEvaluationPassed']
    if evaluation_passed:
      # Policy evaluation passed, nothing to do.
      return

    # See if the new measurement is known good, and if it is, relearn.
    measurements = pcr_values_to_dict(report_event['actualMeasurements'])

    db = firestore.Client()
    kg_ref = db.collection('known_good_measurements')
```

```
# Check current measurements against known good database.
relearn = False
for kg in kg_ref.get():

  kg_map = kg.to_dict()

  # Check PCR values for lateBootReportEvent measurements against the know
  # measurements stored in the Firestore table

  if ('PCR_0' in kg_map and kg_map['PCR_0'] == measurements['PCR_0'] and
      'PCR_4' in kg_map and kg_map['PCR_4'] == measurements['PCR_4'] and
      'PCR_7' in kg_map and kg_map['PCR_7'] == measurements['PCR_7']):

    # Linux VM (3 measurements), only need to check above 3 measurements
    if len(kg_map) == 3:
      relearn = True

    # Windows VM (6 measurements), need to check 3 additional measurements
    elif len(kg_map) == 6:
      if ('PCR_11' in kg_map and kg_map['PCR_11'] == measurements['PCR_11'
          'PCR_13' in kg_map and kg_map['PCR_13'] == measurements['PCR_13'
          'PCR_14' in kg_map and kg_map['PCR_14'] == measurements['PCR_14'
        relearn = True

compute = googleapiclient.discovery.build('compute', 'beta',
    cache_discovery=False)

instance_id = log_entry['resource']['labels']['instance_id']
project_id = log_entry['resource']['labels']['project_id']
zone = log_entry['resource']['labels']['zone']

instance_name = instance_id_to_instance_name(compute, zone, project_id, ir

if not relearn:
  # Issue shutdown API call.
  print('New measurement is not known good. Shutting down a VM.')

  result = compute.instances().stop(project=project_id,
      zone=zone,
      instance=instance_name).execute()

  print('Instance %s in project %s has been scheduled for shut down.'
        % (instance_name, project_id))
else:
  # Issue relearn API call.
  print('New measurement is known good. Relearning...')

  result = compute.instances().setShieldedInstanceIntegrityPolicy(
      project=project_id,
      zone=zone,
      instance=instance_name,
      body={'updateAutoLearnPolicy':True}).execute()

  print('Instance %s in project %s has been scheduled for relearning.'
    % (instance_name, project_id))
```

2. Copy the following dependencies and use them to overwrite the existing code in `requirements.txt`:

```
google-api-python-client==1.6.6
google-auth==1.4.1
google-auth-httplib2==0.0.3
google-cloud-firestore==0.29.0
firebase-admin==2.13.0
```

3. Open a terminal window and navigate to the directory containing `main.py` and `requirements.txt`.
4. Run the `gcloud beta functions deploy` command (https://cloud.google.com/sdk/gcloud/reference/beta/functions/deploy) to deploy the trigger:

```
gcloud beta functions deploy relearn_if_known_good --project YOUR_PROJECT_ID \
    --runtime python37 --trigger-resource integrity-monitoring \
    --trigger-event google.pubsub.topic.publish
```

   replacing `YOUR_PROJECT_ID` with the ID of your project.
5. Manually delete the previous `shutdown_vm` function in the cloud function console.
6. GO TO CLOUD FUNCTIONS (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FUNCTIONS)
7. Select the **shutdown_vm** function and click delete.

Verify the automated responses to integrity validation failures

1. First, check if you have a running instance with **Secure Boot** turned on as a Shielded VM option. If not, you can create a new instance with Shielded VM image (Ubuntu 18.04LTS) and turn on the **Secure Boot** option. You may be charged a few cents for the instance (this step can be finished within an hour).
2. Now, assume for some reason, you want to manually upgrade the kernel.
3. SSH into the instance, and use the following command to check the current kernel.

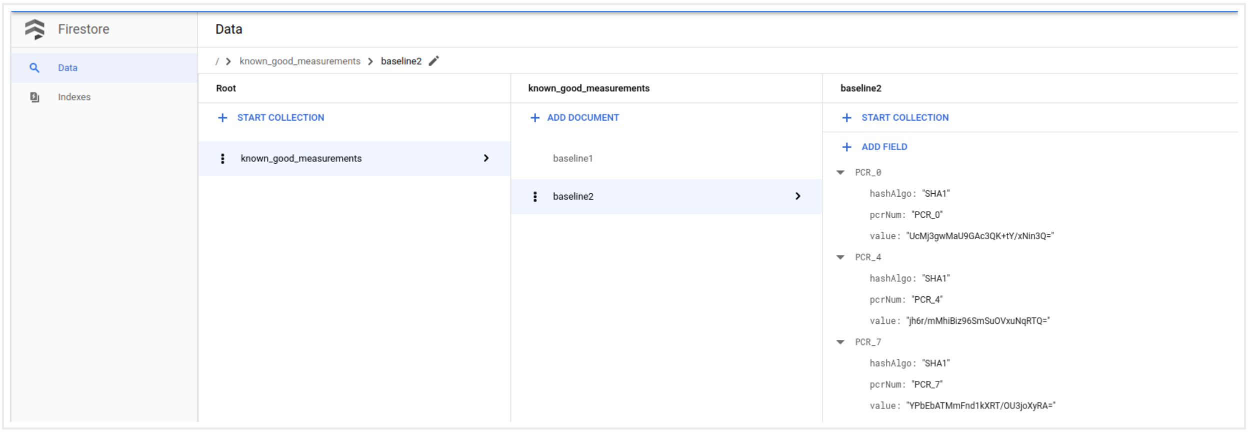
   ```
   uname -sr
   ```

   You should see something like `Linux 4.15.0-1028-gcp`.
4. Download a generic kernel from https://kernel.ubuntu.com/~kernel-ppa/mainline/
5. Use the command to install.

   ```
   sudo dpkg -i *.deb
   ```

6. Reboot the VM.
7. You should notice the VM is not booting up (cannot SSH into the machine). This is what we expect, because the signature of the new kernel is not in our **Secure Boot** whitelist. This also demonstrates how **Secure Boot** can prevent an unauthorized/malicious kernel modification.
8. But because we know this time the kernel upgrading is not malicious and is indeed done by ourself, we can turn off **Secure Boot** in order to boot the new kernel.
9. Shutdown the VM and untick the **Secure Boot** option, then restart the VM.
10. The boot of the machine should fail again! But this time it is being shutdown automatically by the cloud function we created as the **Secure Boot** option has been altered (also because of the new kernel image), and they caused the measurement to be different than the baseline. (We can check that in the cloud function's **Stackdriver** log.)
11. Because we know this is not a malicious modification and we know the root cause, we can add the current measurement in `lateBootReportEvent` to the known good measurement Firebase table. (Remember there are two things being changed: 1. **Secure Boot** option 2. Kernel Image.)

    Follow the previous step **Creating a database of known good baseline measurements** to append a new baseline to the Firestore database using the actual measurement in the latest `lateBootReportEvent`.



```
PCR_0
    hashAlgo: "SHA1"
    pcrNum: "PCR_0"
    value: "UcMj3gwMaU9GAc3QK+tY/xNin3Q="
PCR_4
    hashAlgo: "SHA1"
    pcrNum: "PCR_4"
    value: "jh6r/mMhiBiz96SmSuOVxuNqRTQ="
PCR_7
    hashAlgo: "SHA1"
    pcrNum: "PCR_7"
    value: "YPbEbATMmFnd1kXRT/OU3joXyRA="
```

12. Now reboot the machine. When you check the **Stackdriver log**, you will see the `lateBootReportEvent` still showing false, but the machine should now boot successfully, because the cloud function trusted and relearned the new measurement. We can verify it by checking the **Stackdriver** of the cloud function.
13. With **Secure Boot** being disabled, we can now boot into the kernel. SSH into the machine and check the kernel again, you will see the new kernel version.

```
uname -sr
```

14. Finally, let's clean up the resources and the data used in this step.
15. Shutdown the VM if you created one for this step to avoid additional charge.
16. GO TO THE VM INSTANCES PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/COMPUTE/INSTANCES
17. Remove the known good measurements you added in this step.
18. GO TO THE FIRESTORE CONSOLE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FIRESTORE/DATA)

Last updated December 26, 2019.