Fall | 2014

Technical Report - Intel Practicum

Team BlueChip
Carnegie Mellon University

This technical report is a summary of the Barbican-OAT proof-of-concept model designed and implemented by Team Bluechip as Fall Practicum project sponsored by Intel, and is intended to be used as a reference guideline for Openstack-based key management service integration with Intel attestation.

Table of Contents

1 | Logistics

Team

Deepti Sunder Prakash, dsunderp@andrew.cmu.edu
Divya Chandra Sekar, divyac@andrew.cmu.edu
Marcelo Domiguez, marcelod@andrew.cmu.edu
Rashmi Krishnan, rashmik@andrew.cmu.edu

Faculty Advisor

Patrick Tague, patrick.tague@west.cmu.edu

Sponsor - Intel Corp.

Jonathan Buhacoff, jonathan.buhacoff@intel.com
Eric Gee, eric.r.gee@intel.com

2 | Executive Summary

The practicum team consist of four CMU students in Silicon Valley working for satisfying Intel's requirements. The project with Intel has the objective to integrate Barbican with KMIP via a secret store plugin to generate, store and retrieve secrets, and at the same time extend Barbican functionality via plugin to verify client credentials through an attestation service.

3 | Introduction

Today users are increasingly relying on encryption to protect their data from accidental disclosure and with this there is an increasing need of managing hundreds to thousands of keys. Apart from disk encryption, passphrases for SSL keys and passwords also need to be managed. The problem gets worse when the cloud provider has been breached i.e. organization might lose all the keys and thus their data. Thus, it is required to have a solid credential storage solution to store all of the credentials and keys safely. Our project works towards the solution of providing secure storage of keys and credentials through Barbican with KMIP used as backend. Through this solution, it is also possible to leverage agent running on managed VMs to talk securely to the key management server.

Currently, most of the cloud providers do not have access control available i.e. they have no control over which client/agent can request for keys. Keystone which acts as an authentication and authorization server is integrated with Barbican which provides tokens to access/store keys to the secure storage. Whenever the

user wishes to use a platform, some kind of remote attestation protocol should be in place which ensures that the platform has not been tampered in any way. This involves reading the PCR(Platform configuration registers) measurements stored on the TPM chip of hardware. Thus, our open source solution called “BOAT” (Barbican with Open Attestation) solves the above security problems related to key management, trust attestation, authentication & authorization in cloud infrastructures.

This report explains in detail security problems that motivates the project, describes in detail the above mentioned system solutions, including the different components involved and their communication flows. The implementation details are explained and organized in three milestones. These sections are followed by conclusion and future work for our project.

4 | Motivation

Cloud computing exposes different security risks when the corporate information is hosted in a public cloud provider or when images are dynamically created and distributed to a commodity hardware in a private cloud:

- To maximise resources, cloud providers serve different virtual servers in the same physical machine. Same machine that potentially runs applications of direct competitors can be a source of threat for organization data. This requires storing data securely or obfuscating access to it.
- There is no guarantee that Hypervisors, firmware or Operating Systems have not been compromised.
- Organization data hosted by a third party in a public cloud gives third party unlimited access to organization assets deployed in the cloud infrastructure.

The team proposes a solution that use different components of OpenStack and at the same time takes advantages of Intel solutions to provide security in cloud infrastructures. The presented solution in this paper tries to solve the following security requirements:

- How to verify that a compute node is trusted?
- How to ensure that none but the trusted compute node can view the VM?
- How to manage/generate the secret keys used for encryption?
- How to ensure only the client can store secrets
- How to ensure only the client (on behalf of the trusted compute node) can retrieve secret keys?

5 | Project Scope

The scope of this project was limited to research on plugin support and KMIP support for Barbican and development of plugin to implement key wrapping functionality. The timeline for this project was six weeks.

6 | Assumptions

- The project is open source and does not require any Intel assets, but could involve other open source projects using either MIT, BSD, or Apache 2.0 licenses.
- Currently we are not emulating the movement and scheduling of the encrypted VM from client system to the compute node.
- The implementation expects and depends on proper functioning of the TCB. For our project, our TCP comprises of the client, KMS(key management server), KMIP(Key management interoperability protocol), agent and OAT(OpenAttestation).
- If client/agent is compromised, then any key retrieval/generation request can be made to barbican since they are part of the OAT CA trusted network. Denial of Service attacks are possible and confidentiality can be compromised.
- If Barbican system is compromised, then the secret keys are disclosed
- If OAT server is compromised, then the attacker can sign any certificate and pose an attacker machine as a legit one
- We are not simulating integrity measurements done by OAT server as we do not have necessary infrastructure. OAT server returns stub responses instead
- KMIP runs on the same machine as KMS because of H/W limitations.

7 | Related Work

In Divya's internship the following things were accomplished:

- Research on using Barbican (OpenStack's solution for key management) for generation and management of AES key used to encrypt the client's virtual machine that needs to be run on a trusted compute node in the cloud infrastructure.
- Barbican generated and stored the AES keys in a backend postgres database and the corresponding url generated was mapped against a pseudonym to provide some obfuscation.
- All communication with Barbican was over HTTP and was not secured using SSL as Barbican at that time didn't have support for SSL.
- There was no authentication and authorization and any machine could communicate with Barbican to store and retrieve secrets leading to the possibility of denial of service attacks and breach of confidentiality.

8 | System Design

8.1 BOAT Architecture Overview

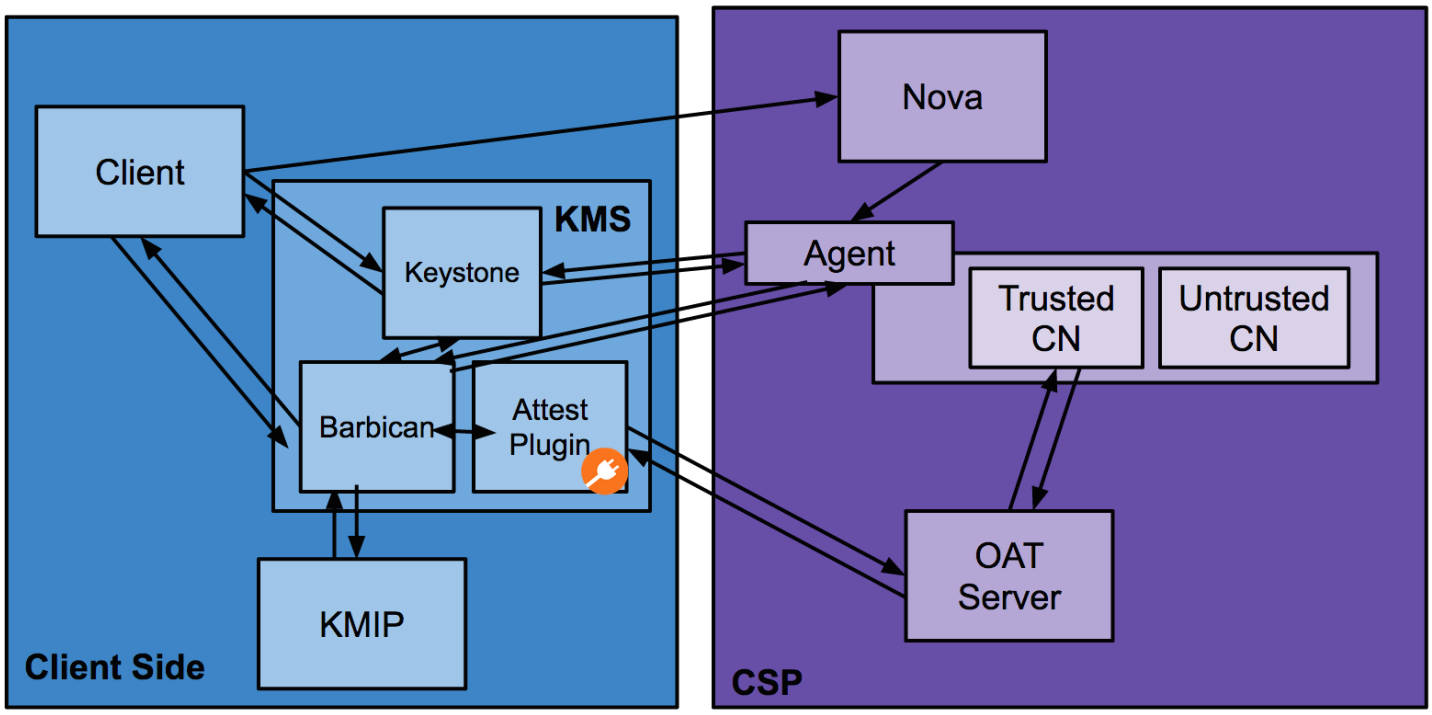

Figure 8.1 - System Design of BOAT

The complete system design of Project BOAT can be seen in Figure 4.1. On the Client side we have 3 systems:

1. Client machine: The system that intends to run a virtual machine on a trusted compute node.
2. Key Management System (KMS): The system that manages authenticating and authorizing the client & agent to store and retrieve secrets, attestation and necessary key wrapping, and management of secrets that are created at the KMIP Server.
    a. Keystone: The component performs username-password authentication to give authorization tokens to the client and agent. This authorization token is necessary to perform an further actions involving Barbican (the key management component).
    b. Barbican: Manages secrets that are created by KMIP server and facilitates retrieval of secrets based on generated ID that maps to the secret stored in KMIP.
    c. Attest Plugin: This component intercepts key retrieval requests from an agent and verifies if the compute node to where the client's virtual machine has been scheduled to execute is trusted or not by making appropriate calls to the OAT Server (Attestation Server). It also performs key wrapping based on the trust status & Bind public key (of compute node) returned by the OAT server.
3. KMIP Server: This component creates and stores secrets/key based on the requirements of the client. It can only be accessed by the KMIP client that resides in Barbican. Together, Barbican and KMIP

perform key generation according to client specifications and key retrieval based on external ID generated.

On the Cloud Service Provider side (CSP):

1. Nova: Once the client creates a key, it encrypts the virtual machine with this key and sends it to the Openstack scheduler-Nova with the keyID of the encryption key. Nova schedules decides which compute node the received computation request must be run on and accordingly sends the request with the appropriate information the the Agent.
2. Agent: This component is responsible for retrieving the wrapped encryption key from the KMS, requesting for decryption of the key from the compute node, decrypting the encrypted virtual machine and forwarding the decrypted virtual machine to the compute node for computation.
3. OAT server: This component attests if a compute node is trusted or not based on its signed PCR values. It also attests if the compute node is compliant to other required policies like location specified by the client.
4. Compute Nodes: These are the actual components that perform the required computation for the client. The nodes can be trusted or untrusted based on the PCR hashes stored in it's TPM. The trusted nodes perform a trusted boot while they startup (i.e. components in the chain are hashed [and appended to the PCRs in the TPM] and only loaded if the hash is present in the whitelist). These measurements are sent to the OAT server upon request for attestation.

8.2 BOAT Processes

8.2.1 Key Generation

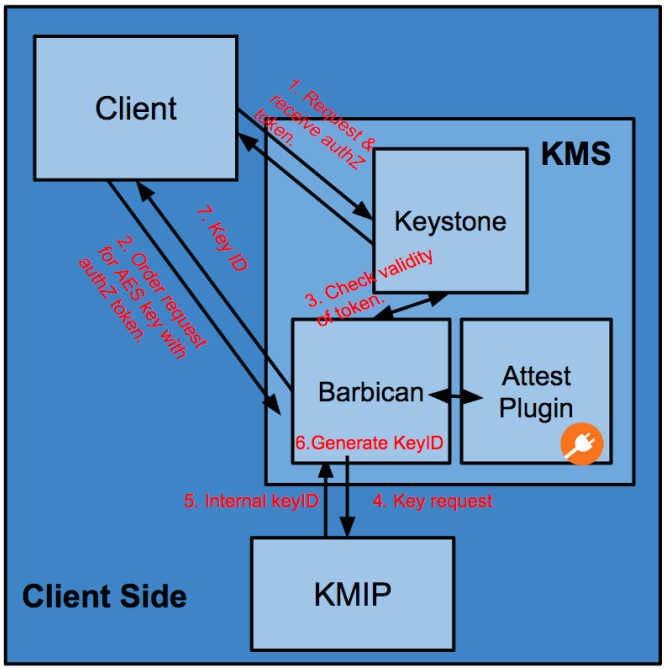

Figure 8.2 - Key generation process in BOAT

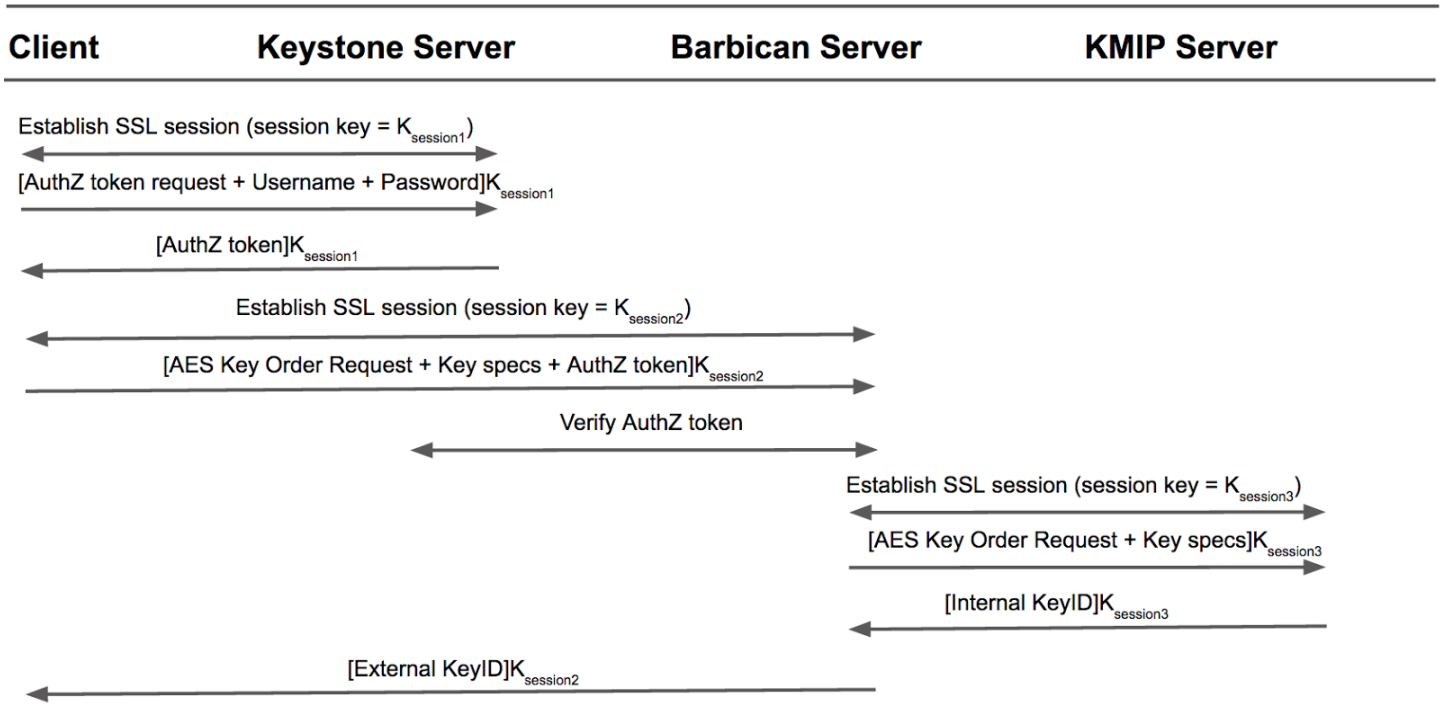

Figure 8.3 - Key generation process in BOAT (Dataflow)

The key generation process in BOAT is illustrated in figures 4.2 and 4.3. The client requires a key to encrypt the virtual machine that it sends to the compute node so as to ensure none but the compute node can view the computation. It requests the KMS to create a 256 bit AES key with which it can encrypt the virtual machine. The details of the key generation process are as follows:

1. The client authenticates to the keystone component by providing username and password to get an authorization token. The communication with keystone is over SSL so as to ensure that the username and password and authorization token are sent over a secure encrypted channel and no one other than the client and the keystone component can see this information. Authentication and authorization is done so as to ensure that only a legit client communicates with the KMS.
2. The client now sends an order request for a 256 bit AES key to the Barbican component with the authorization token got from keystone. All communication with Barbican is also over SSL so as to ensure that all sensitive information exchanged is encrypted. We have also enabled mutual authentication with SSL so as to authenticate the client and ensure only legit clients make key generation or retrieval requests.
3. Barbican checks the validity of the authorization token by communicating with keystone. Once the check passes it forwards the order request to the KMIP server. The communication between Barbican and KMIP is also over SSL.
4. The KMIP server creates a key according to the required specification and sends back an internal keyID that maps to the key generated, to Barbican.
5. Barbican generates an external keyID for the internal keyID returned from KMIP and sends it back to the client.

8.2.2 Key Retrieval (Client)

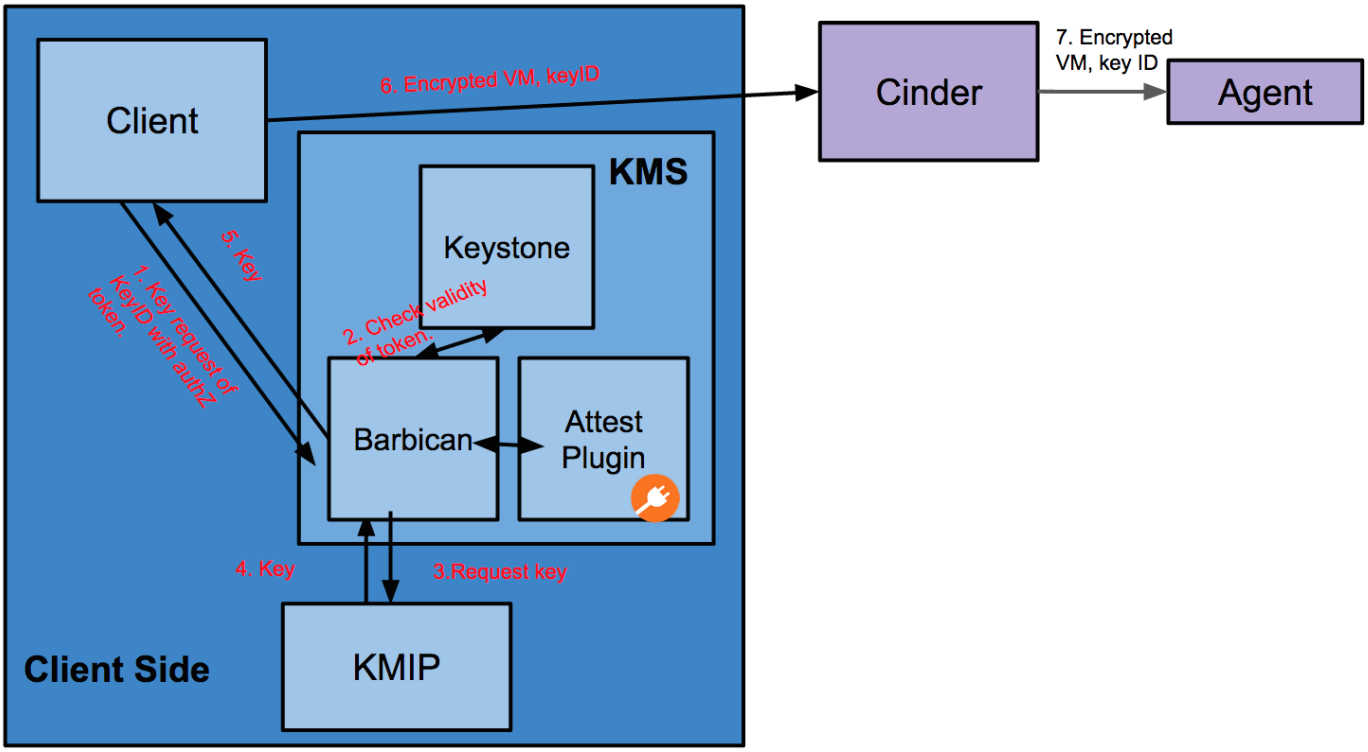

Figure 8.4 - Key retrieval process by client in BOAT

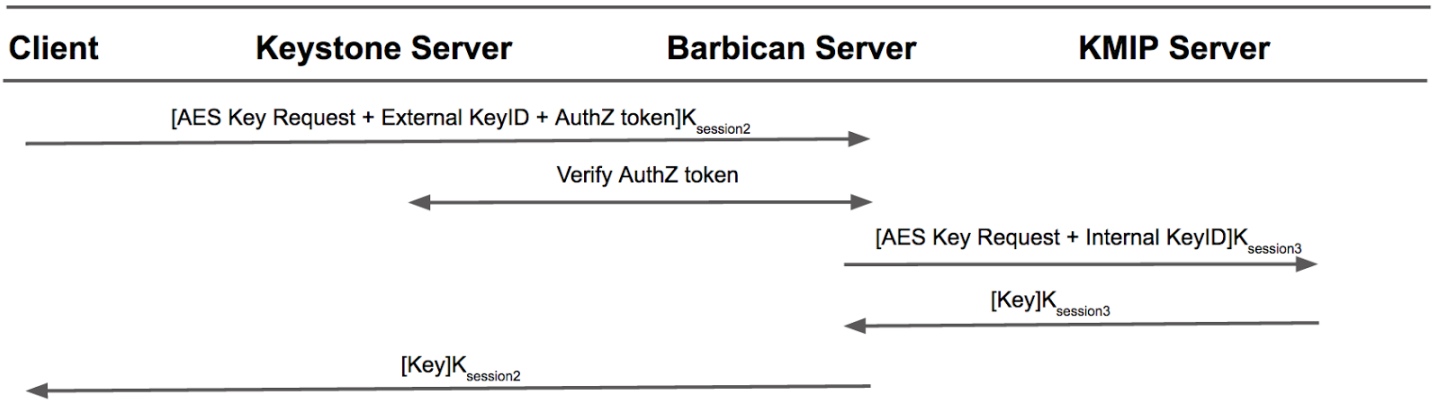

Figure 8.5 - Key retrieval process by client in BOAT (Dataflow)

Now that the client has the keyID it needs to retrieve the key from the KMS. Figure 4.4 and 4.5 illustrate the client's key retrieval process in BOAT. The details of the process are as follows:

1. The client sends a request for the key created with the keyID and the authorization token it received from keystone.
2. Barbican verifies the authorization token by communicating with keystone and forwards the key retrieval request to KMIP with the internal keyID.
3. KMIP server returns the corresponding key to Barbican which then sends this to the client.

Now that the client has retrieved the key it encrypts the virtual machine and sends it to Nova with the keyID of the encryption key.

8.2.3 Attestation & Key Retrieval (Agent)

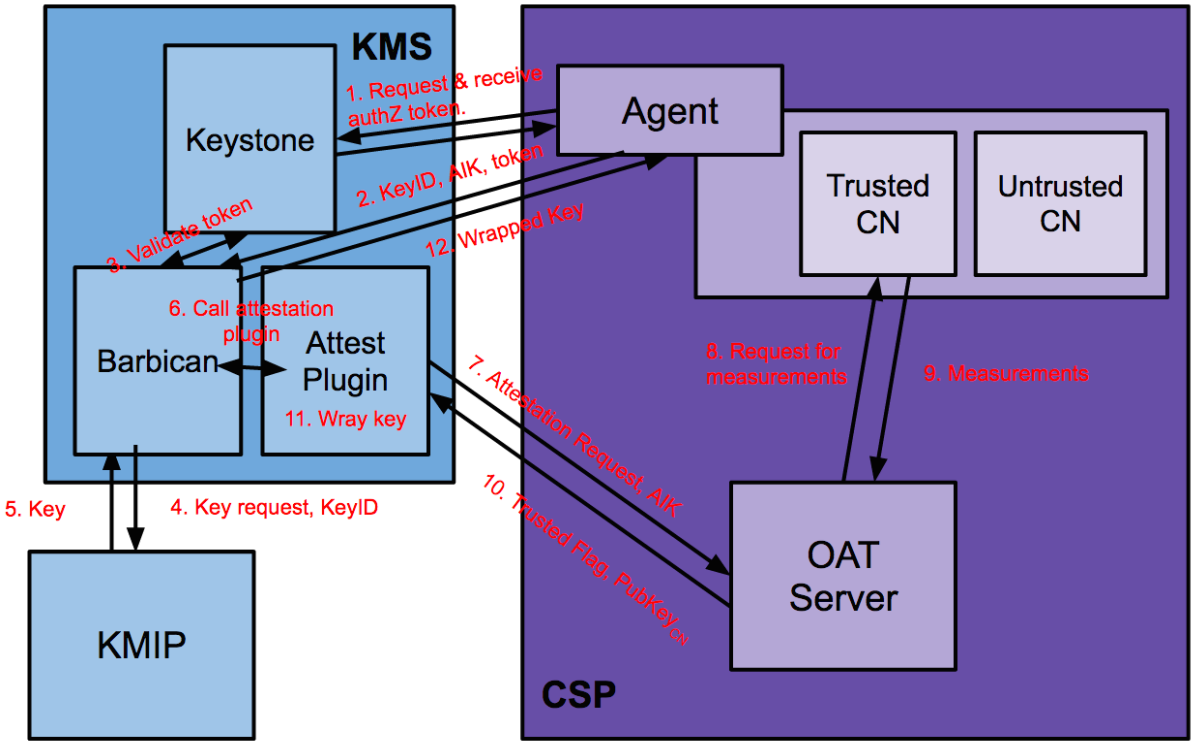

Figure 8.6 - Attestation & Key retrieval process by agent in BOAT

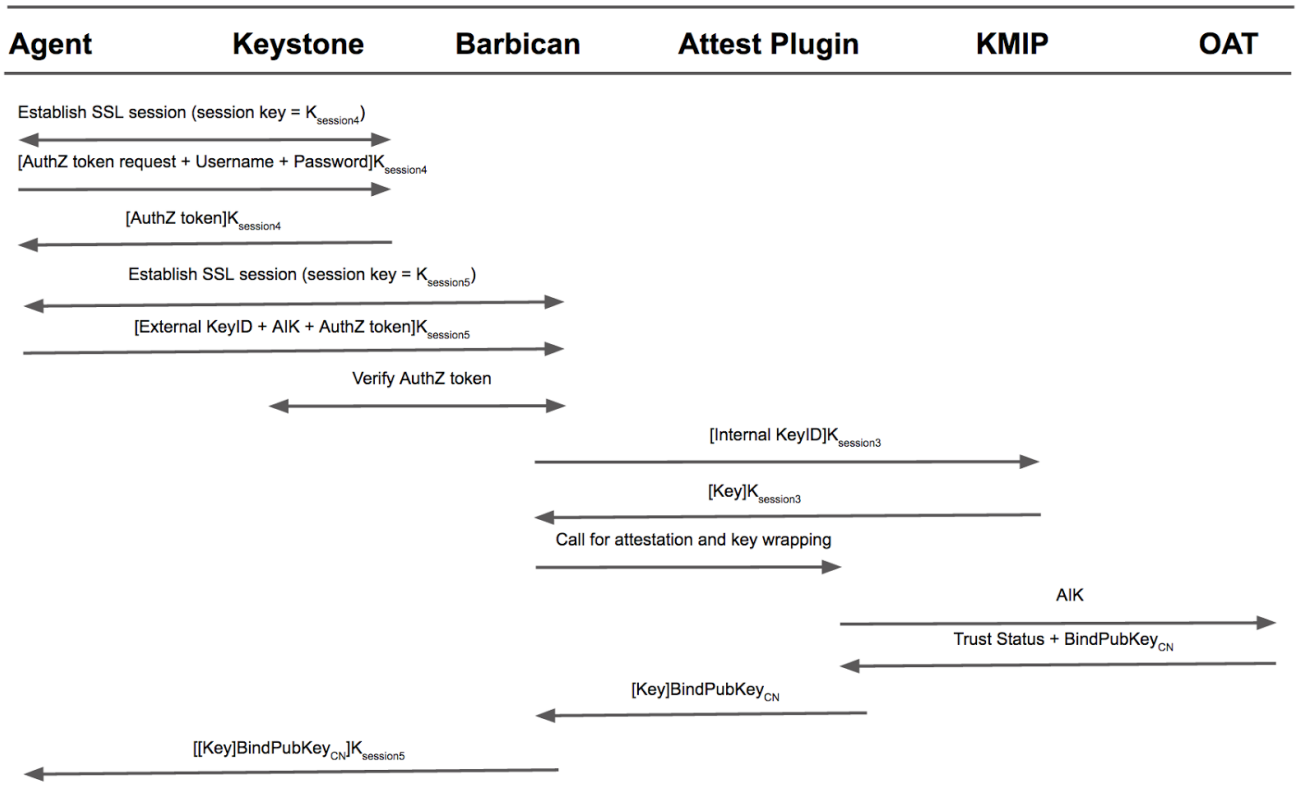

Figure 8.7 - Attestation & Key retrieval process by agent in BOAT (Dataflow)

Now the encrypted virtual machine that has reached the agent, needs to be decrypted and executed on a trusted compute node. The agent requests for the encryption key by passing the AIK (Attestation Identity Key) of the compute node as a header value in the key retrieval request. Figure 4.6 & 4.7 illustrate the agent's key retrieval process. The details of the process is as follows:

1. The Agent authenticates itself with keystone and requests for an authorization token.
2. It then sends the key retrieval request to Barbican with the AIK of the compute node and authorization token from keystone.
3. Barbican verifies the authorization token by communicating with keystone and then requests for the key from KMIP with the internal keyID.
4. KMIP returns the key to Barbican.
5. Barbican now calls the attest plugin to verify if the compute node is trusted and compliant with the policies specified by the client.
6. The attest plugin calls the OAT server for the verification and wraps the key with bind public key of the compute node on positive trust status of attestation.
7. The wrapped key is returned to Barbican, which returns it to the agent.

Agent now has the wrapped key and the encrypted virtual machine. It sends the wrapped key to the compute node for decryption and then decrypts the virtual machine with decrypted key. It forwards the virtual machine to the compute node for computation.

9 | System Implementation

9.1 Milestone 1

9.1.1 Objective Overview

The first milestone involved research on Openstack Barbican support for external plugins and support for KMIP server backend.

9.1.2 Technology Selection

Analysis of existing internal plugins incorporated in Barbican, such as crypto, secret_store, and others, showed that stevedore is used to manage plugins and use them dynamically at runtime. Since it implements an efficient method to load plugins and has already integrated well with Barbican project, we decided to implement a stevedore-type plugin in Python for remote attestation that can be installed as a library in Barbican python environment (pyenv) using setuptools.

Through our research of KMIP server, we came across KMIP4J which is an open source implementation of KMIP in Java which has server, client and has its own database implementation. KMIP4J runs its server over tomcat apache server and requires ANT configuration to run database. We realized that we do not need a heavy-weight implementation of KMIP server which required starting DB and running apache tomcat server.

So we chose PyKMIP which is a local python implementation of KMIP server. Running KMIP server is as simple as running KMIP server.py file.

9.2 Milestone 2

9.2.1 Objective Overview

The second milestone was a working implementation of attestation plugin, which intercepts key retrieval requests and redirects the Agent's request (along with AIK) for integrity verification and attestation by OAT Server.

9.2.2 Setup and Installation Guidelines

Stevedore uses naming convention and namespaces to define an entry point for the plugin to be called by the main application at runtime. Using stevedore plugin guidelines, we have built a plugin of namespace **boat.attestation.plugin**.

The following steps must be done to use the attestation plugin in barbican setup: -

a. Start your barbican python environment
   pyenv shell barbican27
b. Ensure you have stevedore and setuptools installed in your python environment.
c. Copy boat folder and contents from github repository branch **bluechip** (location barbican/bluechip/attest-plugin/boat) and add to site-packages in python environment of Barbican (location ~/.pyenv/versions/barbican27/lib/python2.7/site-packages/).
d. Install the plugin as a library using the command:
   python setup.py install
e. Append the following change in /etc/barbican/barbican-api.conf

```
# ================= Attestation plugin ===================
[attester]
namespace = boat.attestation.attester
enabled_attester_plugins = simple_attest
```

f. start your barbican instance:
   bin/barbican.sh start

Note: - The GET method in Barbican API calls attest plugin (barbican/api/controllers/secrets.py)

9.2.3 Working of Attest Plugin

The purpose of the attest plugin is to intercept key retrieval requests from Agent to Barbican and ensure that only trusted compute node AIK can request for the key. The dynamic call to attest plugin is implemented in secrets.py (the main REST API for Barbican functionality).

9.3 Milestone 3

9.3.1 Objective Overview

The objective of the third milestone was to implement a secret store plugin which uses an open source KMIP server to generate, store and retrieve secrets and a Barbican KMIP client which makes calls to the KMIP server on behalf of client to store and retrieve secrets.

9.3.2 Setup and Installation Guidelines

Barbican by default uses store_crypto plugin as a custom cryptographic plugin.

Setting up KMIP secret store plugin has the following steps:

1. **Configure Barbican to run our secret store plugin**
    a. Go to <barbican-installation-folder>/setup.cfg
    b. Change store_crypto = barbican.plugin.store_crypto:**StoreCryptoAdapterPlugin** to store_crypto = barbican.plugin.kmip_secret_store:**KMIPSecretStore**
    KMIPSecretStore: this class in defined in kmip_secret_store.py which is in "<barbican-installation-folder>/barbican/plugin" directory
    c. Add **pyKMIP>=0.2.0** to <barbican-installation-folder>/requirements.txt file to install PyKMIP
    d. Run <barbican-installation-folder>/pip install -e . to install barbican again
    e. Run Barbican:
        i. Go to <barbican-installation-folder>
        ii. Run bin/barbican.sh start

2. **Run KeyStone**
    a. Start keystone by running the following command: */opt/stack/keystone/bin/keystone-all --verbose --debug*

3. **Run KMIP server**
    a. Download the kmipconfig.ini file from "https://github.com/OpenKMIP/PyKMIP/blob/master/kmip/kmipconfig.ini"
    b. Store this file in "/usr/local/lib/<python folder>/dist-packages/kmip"
    c. Make following changes to kmipconfig.ini file:
        i. Change keyfile and certfile under [server] section to KMIP server's private key and certificate
        ii. If mutual authentication is required, change cert_reqs under [server] section to CERT_REQUIRED which means client certificate is required
        iii. Change ca_certs both in client and server section to Certificate Authority's certificate
        iv. Set username and password to <admin> and <passwordxyz> under client section
    d. Download server.py from "https://github.com/OpenKMIP/PyKMIP/tree/master/kmip/demos" and store this file in some other customized folder or use the existing server.py under "/usr/local/lib/<python folder>/dist-packages/kmip/demos"
        i. Start the KMIP server by running the command:

python server.py --host <hostname> --port <port_number> --keyfile </path/to/keyfile> --certfile </path/to/servercertfile> --ca_certs </path/to/CAcertfile>

9.3.3 Working of KMIP Server

Barbican uses secret store plugin kmip_secret_store.py to generate, store and retrieval secrets from KMIP server. Before starting Barbican server, make changes to the set up file of Barbican server to use kmip_secret_store plugin. Currently we do not have a way to load custom plugin because we ran into errors where Barbican was always using the default plugin. As a workaround to this, we changes to the set up file such that the default plugin points to the kmip_secret_store plugin mentioned in the above "Set up and installation guidelines" section under 'Run Barbican'.

KMIP python server runs in background. When client makes request for key generation, secret store plugin calls **generate_symmetric_key** which in turn calls Barbican KMIP client which opens connection to KMIP server to create secret. KMIP server returns an order_pref which is then used to generate keyID. In order to store secret to backend, **store_secret** is called by the secret store plugin. Every call made to KMIP server is authorized by Keystone which uses X-Auth-Token for authorization. When client requests for key, it requires a "secret_ref" to do so which is obtained during previous stage of key storage. This is again authorized by Keystone, Barbican KMIP client calls **get_secret** along with keyID to KMIP server to retrieve the encryption key.

9.4 Milestones timeline

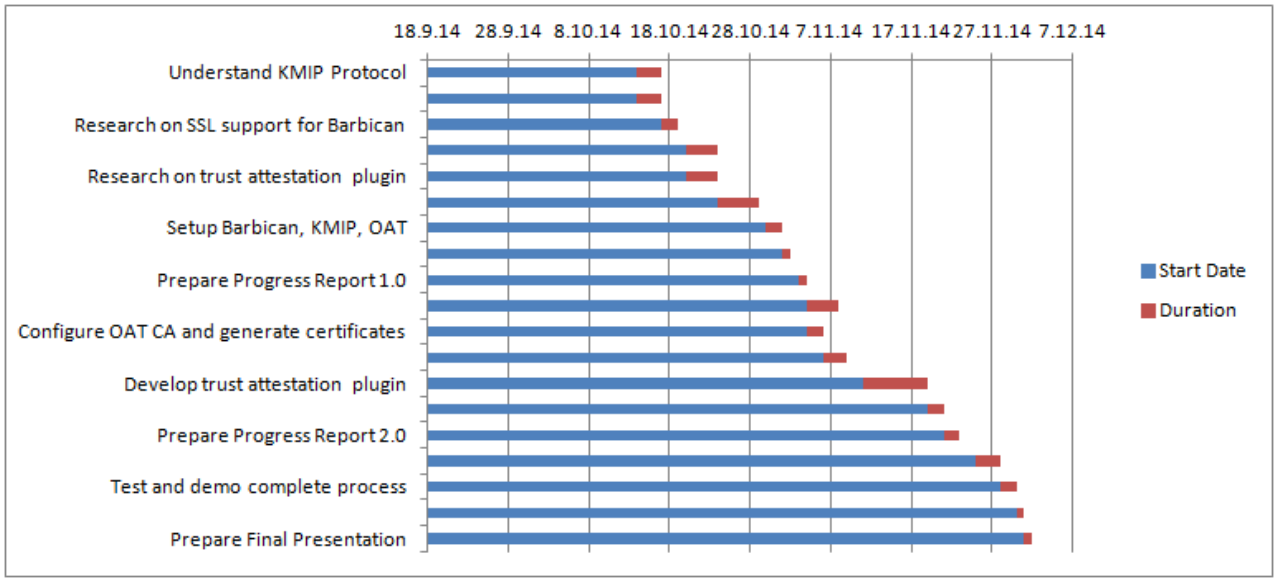

Figure 9.1 - Milestones Timeline for BOAT

10 | Experiments and Analysis

10.1 Key generation and retrieval (Client)

1. **Request token from keystone:**

   Syntax:
   curl -k -X 'POST' -v https://<keystone_server>:<keystone_server_port_number>/v2.0/tokens -d '{"auth":{"tenantName":"<tenant_name>", "passwordCredentials":{"username": "<username>", "password":"<password>"}}}' -H 'Content-type: application/json'

   Request:
   curl -k -X 'POST' -v https://10.0.16.144:5000/v2.0/tokens -d '{"auth":{"tenantName":"demo", "passwordCredentials":{"username": "admin", "password":"bluechip"}}}' -H 'Content-type: application/json'

   Response: Authorization token

{"access": {"token": {"issued_at": "2014-12-06T23:06:44.398704", "expires": "2014-12-07T00:06:44Z", "id": "**5f382852028f4fbc9c5d584941bca0d1**", "tenant": {"description": null, "enabled": true, "id": "ec7f7425fd9e4e01be632f9cf35bebfd", "name": "demo"}, "audit_ids": ["xwOM_KE9TjWwuTcyyktJKA"]}, "serviceCatalog": [{"endpoints": [{"adminURL": "http://10.0.16.144:9312/v1", "region": "RegionOne", "internalURL": "http://10.0.16.144:9313/v1", "id": "83b42c94b4284b69ac7f51bb22d6672b", "publicURL": "http://10.0.16.144:9311/v1"}], "endpoints_links": [], "type": "keystore", "name": "barbican"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8774/v2/ec7f7425fd9e4e01be632f9cf35bebfd", "region": "RegionOne", "internalURL": "http://10.0.16.144:8774/v2/ec7f7425fd9e4e01be632f9cf35bebfd","id": "157bdc9680964f7891b2c6f98820752c", "publicURL": "http://10.0.16.144:8774/v2/ec7f7425fd9e4e01be632f9cf35bebfd"}], "endpoints_links": [], "type": "compute", "name": "nova"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8774/v2.1/ec7f7425fd9e4e01be632f9cf35bebfd", "region": "RegionOne", "internalURL": "http://10.0.16.144:8774/v2.1/ec7f7425fd9e4e01be632f9cf35bebfd", "id": "1ea6c13b933b453f82796355eb1fdb34", "publicURL": "http://10.0.16.144:8774/v2.1/ec7f7425fd9e4e01be632f9cf35bebfd"}], "endpoints_links": [], "type": "computev21", "name": "novav21"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8776/v2/ec7f7425fd9e4e01be632f9cf35bebfd", "region": "RegionOne", "internalURL": "http://10.0.16.144:8776/v2/ec7f7425fd9e4e01be632f9cf35bebfd", "id": "005752199c3b48138b98ef39019ad1ff", "publicURL": "http://10.0.16.144:8776/v2/ec7f7425fd9e4e01be632f9cf35bebfd"}], "endpoints_links": [], "type": "volumev2", "name": "cinderv2"}, {"endpoints": [{"adminURL": "http://10.0.16.144:3333", "region": "RegionOne", "internalURL": "http://10.0.16.144:3333", "id": "480476deed504f8499fdc29245256b5c", "publicURL": "http://10.0.16.144:3333"}], "endpoints_links": [], "type": "s3", "name": "s3"}, {"endpoints": [{"adminURL": "http://10.0.16.144:9292", "region": "RegionOne", "internalURL": "http://10.0.16.144:9292", "id": "7b5fa7b2899d4577b7ee660c9427ef38", "publicURL": "http://10.0.16.144:9292"}], "endpoints_links": [], "type": "image", "name": "glance"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8000/v1", "region": "RegionOne", "internalURL": "http://10.0.16.144:8000/v1", "id": "3315199b4b2b42138dfa0fd4b2b0e060", "publicURL": "http://10.0.16.144:8000/v1"}], "endpoints_links": [], "type": "cloudformation", "name": "heat-cfn"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8776/v1/ec7f7425fd9e4e01be632f9cf35bebfd", "region": "RegionOne", "internalURL": "http://10.0.16.144:8776/v1/ec7f7425fd9e4e01be632f9cf35bebfd", "id": "0c33742907934b1e96b0b699c435b542", "publicURL": "http://10.0.16.144:8776/v1/ec7f7425fd9e4e01be632f9cf35bebfd"}], "endpoints_links": [], "type": "volume", "name": "cinder"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8773/services/Admin", "region": "RegionOne", "internalURL": "http://10.0.16.144:8773/services/Cloud", "id": "1af17d9f68664907a24b9ff6a487c479", "publicURL": "http://10.0.16.144:8773/services/Cloud"}], "endpoints_links": [], "type": "ec2", "name": "ec2"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8004/v1/ec7f7425fd9e4e01be632f9cf35bebfd", "region": "RegionOne", "internalURL": "http://10.0.16.144:8004/v1/ec7f7425fd9e4e01be632f9cf35bebfd", "id": "745a51454b1945cd84cf6e0f2721076b", "publicURL": "http://10.0.16.144:8004/v1/ec7f7425fd9e4e01be632f9cf35bebfd"}], "endpoints_links": [], "type": "orchestration", "name": "heat"}, {"endpoints": [{"adminURL": "http://10.0.16.144:35357/v2.0", "region": "RegionOne", "internalURL": "https://10.0.16.144:5000/v2.0", "id": "2b8c3e44549847d38e238e191b8b634f", "publicURL": "https://10.0.16.144:5000/v2.0"}], "endpoints_links": [], "type": "identity", "name": "keystone"}], "user": {"username": "admin", "roles_links": [], "id": "0033335d527e4f1ba1659ef5593fb462", "roles": [{"name": "admin"}, {"name": "heat_stack_owner"}], "name": "admin"}, "metadata": {"is_admin": 0, "roles": ["2338e32580c948d1ab386c9d9e04e04d", "635ae65c43ac45fda218e68e9068467e"]}}}

2. **Key generation request:**

   Syntax:

curl -k --key <client_key> --cert <client_cert> -X POST -H 'content-type:application/json' -H 'X-Project-Id: <project_id> -H 'X-Auth-Token:<token id>' -d '{ "type":"key","meta": {"name": "<secret_name>", "algorithm": "<crypto_algo>", "bit_length": <bit_length>, "mode": "<mode>", "payload_content_type": "application/octet-stream"}}' https://<nginx_server>:443/v1/orders

Request:
curl -k --key client.key --cert client.crt -X POST -H 'content-type:application/json' -H 'X-Project-Id: 12345' -H 'X-Auth-Token:5f382852028f4fbc9c5d584941bca0d1' -d '{ "type":"key","meta": {"name": "secretname", "algorithm": "aes", "bit_length": 256, "mode": "cbc", "payload_content_type": "application/octet-stream"}}' https://10.0.16.144:443/v1/orders

Response: order-ref
{"order_ref": "**http://10.0.16.144:9311/v1/orders/64520e77-5527-40b6-9a95-e9230cc44a8d**"}

3. **Get secret ref:**

Syntax:
curl -k --key <client_key> --cert <client_cert> -H 'X-Project-Id: <project_id>‘ -H 'X-Auth-Token:<token id>' https://<nginx_server>:443/v1/orders/<order_ref>

Request:
curl -k --key client.key --cert client.crt -H 'X-Project-Id: 12345' -H 'X-Auth-Token:5f382852028f4fbc9c5d584941bca0d1'
https://10.0.16.144:443/v1/orders/64520e77-5527-40b6-9a95-e9230cc44a8d

Response: secret-ref (this is same as keyID referenced in this report.)
{"status": "ACTIVE", "secret_ref": "**http://10.0.16.144:9311/v1/secrets/c0448f53-9e11-426d-86de-8983f726acfe**", "updated": "2014-12-06T23:17:48.893926", "meta": {"name": "secretname", "algorithm": "aes", "payload_content_type": "application/octet-stream", "mode": "cbc", "bit_length": 256, "expiration": null}, "created": "2014-12-06T23:17:48.685550", "type": "key", "order_ref": "http://10.0.16.144:9311/v1/orders/64520e77-5527-40b6-9a95-e9230cc44a8d"}

4. **Get secret:**

Syntax:
curl -k --key <client_key> --cert <client_cert> -H 'Accept: application/octet-stream' -H 'X-Project-Id: <project_id> -H 'X-Auth-Token:<token id>' https://<nginx_server>:443/v1/secrets/<secret_ref>

Request:
curl -k --key client.key --cert client.crt -H 'Accept: application/octet-stream' -H 'X-Project-Id: 12345' -H 'X-Auth-Token:5f382852028f4fbc9c5d584941bca0d1'
https://10.0.16.144:443/v1/secrets/2df8d196-76b6-4f89-a6d2-c9e764900791

Response: secret (secret i.e. key to encrypt/decrypt VM)
x8f7YXZh12XBg0b/6oFk2NrpPk16+cjp4nI/7V6rx+c=t

10.2 Attestation and key retrieval (Agent)

1. **Request token from keystone:**

   Syntax:
   curl -k -X 'POST' -v https://<keystone_server>:<keystone_server_port_number>/v2.0/tokens -d '{"auth":{"tenantName":"<tenant_name>", "passwordCredentials":{"username": "<username>", "password":"<password>"}}}' -H 'Content-type: application/json'

   Request:
   curl -k -X 'POST' -v https://10.0.16.144:5000/v2.0/tokens -d '{"auth":{"tenantName":"demo", "passwordCredentials":{"username": "admin", "password":"bluechip"}}}' -H 'Content-type: application/json'

   Response: Token
   {"access": {"token": {"issued_at": "2014-12-06T23:22:36.191379", "expires": "2014-12-07T00:22:36Z", "id": "**fbff72ac990a4dcaa449d5efa8cda67d**", "tenant": {"description": null, "enabled": true, "id": "ec7f7425fd9e4e01be632f9cf35bebfd", "name": "demo"}, "audit_ids": ["j6Q4yFilSSiCCNm1urMnEg"]}, "serviceCatalog": [{"endpoints": [{"adminURL": "http://10.0.16.144:9312/v1", "region": "RegionOne", "internalURL": "http://10.0.16.144:9313/v1", "id": "83b42c94b4284b69ac7f51bb22d6672b", "publicURL": "http://10.0.16.144:9311/v1"}], "endpoints_links": [], "type": "keystore", "name": "barbican"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8774/v2/ec7f7425fd9e4e01be632f9cf35bebfd", "region": "RegionOne", "internalURL": "http://10.0.16.144:8774/v2/ec7f7425fd9e4e01be632f9cf35bebfd", "id": "157bdc9680964f7891b2c6f98820752c", "publicURL": "http://10.0.16.144:8774/v2/ec7f7425fd9e4e01be632f9cf35bebfd"}], "endpoints_links": [], "type": "compute", "name": "nova"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8774/v2.1/ec7f7425fd9e4e01be632f9cf35bebfd", "region": "RegionOne", "internalURL": "http://10.0.16.144:8774/v2.1/ec7f7425fd9e4e01be632f9cf35bebfd", "id": "1ea6c13b933b453f82796355eb1fdb34", "publicURL": "http://10.0.16.144:8774/v2.1/ec7f7425fd9e4e01be632f9cf35bebfd"}], "endpoints_links": [], "type": "computev21", "name": "novav21"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8776/v2/ec7f7425fd9e4e01be632f9cf35bebfd", "region": "RegionOne", "internalURL": "http://10.0.16.144:8776/v2/ec7f7425fd9e4e01be632f9cf35bebfd", "id": "005752199c3b48138b98ef39019ad1ff", "publicURL": "http://10.0.16.144:8776/v2/ec7f7425fd9e4e01be632f9cf35bebfd"}], "endpoints_links": [], "type": "volumev2", "name": "cinderv2"}, {"endpoints": [{"adminURL": "http://10.0.16.144:3333", "region": "RegionOne", "internalURL": "http://10.0.2.15:3333", "id": "480476deed504f8499fdc29245256b5c", "publicURL": "http://10.0.16.144:3333"}], "endpoints_links": [], "type": "s3", "name": "s3"}, {"endpoints": [{"adminURL": "http://10.0.16.144:9292", "region": "RegionOne", "internalURL": "http://10.0.16.144:9292", "id": "7b5fa7b2899d4577b7ee660c9427ef38", "publicURL": "http://10.0.16.144:9292"}], "endpoints_links": [], "type": "image", "name": "glance"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8000/v1", "region": "RegionOne", "internalURL": "http://10.0.16.144:8000/v1", "id": "3315199b4b2b42138dfa0fd4b2b0e060", "publicURL": "http://10.0.16.144:8000/v1"}], "endpoints_links": [], "type": "cloudformation", "name": "heat-cfn"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8776/v1/ec7f7425fd9e4e01be632f9cf35bebfd", "region": "RegionOne", "internalURL": "http://10.0.16.144:8776/v1/ec7f7425fd9e4e01be632f9cf35bebfd", "id": "0c33742907934b1e96b0b699c435b542", "publicURL": "http://10.0.16.144:8776/v1/ec7f7425fd9e4e01be632f9cf35bebfd"}], "endpoints_links": [], "type": "volume", "name": "cinder"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8773/services/Admin", "region": "RegionOne", "internalURL": "http://10.0.16.144:8773/services/Cloud", "id": "1af17d9f68664907a24b9ff6a487c479", "publicURL": "http://10.0.16.144:8773/services/Cloud"}], "endpoints_links": [], "type": "ec2", "name": "ec2"}, {"endpoints": [{"adminURL": "http://10.0.16.144:8004/v1/ec7f7425fd9e4e01be632f9cf35bebfd", "region": "RegionOne", "internalURL": "http://10.0.16.144:8004/v1/ec7f7425fd9e4e01be632f9cf35bebfd", "id": "745a51454b1945cd84cf6e0f2721076b", "publicURL": "http://10.0.16.144:8004/v1/ec7f7425fd9e4e01be632f9cf35bebfd"}], "endpoints_links": [], "type": "orchestration", "name": "heat"}, {"endpoints": [{"adminURL": "http://10.0.16.144:35357/v2.0", "region": "RegionOne", "internalURL": "https://10.0.16.144:5000/v2.0", "id": "2b8c3e44549847d38e238e191b8b634f", "publicURL": "https://10.0.16.144:5000/v2.0"}], "endpoints_links": [], "type": "identity", "name": "keystone"}], "user": {"username": "admin", "roles_links": [], "id": "0033335d527e4f1ba1659ef5593fb462", "roles": [{"name": "admin"}, {"name":

"heat_stack_owner"}], "name": "admin"}, "metadata": {"is_admin": 0, "roles": ["2338e32580c948d1ab386c9d9e04e04d", "635ae65c43ac45fda218e68e9068467e"]}}}

2. **Agent key retrieval and decryption:**

   Syntax:
   python agent.py <secret-ref> <compute_node_private_key> <keystone_token>

   Request:
   python agent.py https://10.0.16.144:443/v1/secrets/2df8d196-76b6-4f89-a6d2-c9e764900791 private-key fbff72ac990a4dcaa449d5efa8cda67d

   Response: Decrypted AES key
   x8f7YXZh12XBg0b/6oFk2NrpPk16+cjp4nI/7V6rx+c=t

10.3 Exception handling

BOAT handles the following exceptions:

- BindPublicKey of compute node returned from OAT server is null: This means that the compute node is untrusted and so BOAT sends back a http 401 (Unauthorized) error code to the user.
- Invalid BindPublicKey returned from OAT server: If an invalid BindPublicKey has been returned from the OAT server, encryption will fail so BOAT throws a http 500 (internal server error) error code back to the user.
- OAT not reachable: If the OAT server is unreachable then BOAT throws back a http 500 (internal server error) error code back to the user.
- Invalid AIK: If AIK isn't according to specifications listed in the checkAttribute method, BOAT throws http 400 (Bad request) error code back to the user.

11 | Conclusion and Future Work

11.1 Conclusion

We were able to achieve all the goals designed for this project which included implementing attestation plugin which intercepts requests to Barbican for key retrieval and check with OpenAttestation server if the compute node is trusted; and a secret store plugin which facilitates creating, storing and retrieving secrets. We have also added in authentication and authorization and secured all communication channels by using SSL. The design of BOAT takes certain factors of scalability like having KMIP on a different machine and securing the communication between KMS and Barbican into consideration as well.

11.2 Future Work

Due to lack of infrastructure, we were not able to simulate actual Openstack compute nodes, Nova scheduler and Agent monitoring compute nodes. This project is completely open source but we need to check feasibility

of integration of this project with Intel's corresponding components. Scope for future work includes the following areas:

1. Integration this open source solution with Intel's proprietary components.
2. Emulate and test actual attestation of compute nodes.
3. Develop end to end model of the solution by including transfer and scheduling of encrypted virtual machine from client to agent, decryption of AES key and virtual machine by the agent and passing the decrypted machine to the trusted compute node to execute.

12 | References

- http://enstratius.typepad.com/blog/2014/11/top-4-cloud-security-issues-part-4.html
- http://en.community.dell.com/techcenter/cloud/b/dell-cloud-blog/archive/2014/10/06/the-top-4-cloud-security-issues-part-one-identity-and-authentication
- http://en.community.dell.com/techcenter/cloud/b/dell-cloud-blog/archive/2014/10/17/the-top-4-cloud-security-issues-part-two-access-control-and-authorization
- https://github.com/cloudkeep/barbican/wiki/Barbican-Quick-Start-Guide
- https://github.com/openstack/barbican-specs/blob/master/specs/juno/api-orders-add-more-types.rst
- http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
- https://github.com/cloudkeep/barbican/wiki/Deploy-OpenStack-Barbican-with-Nginx-web-server
- http://nginx.org/en/docs/http/configuring_https_servers.html
- http://docs.openstack.org/admin-guide-cloud/content/keystone-ssl-config.html
- http://docs.openstack.org/developer/keystone/configuration.html
- https://github.com/kjtanaka/deploy_havana/wiki/How-to-enable-ssl-on-keystone
- https://github.com/cloudkeep/barbican/wiki/Barbican-Options:-authentication-with-Keystone