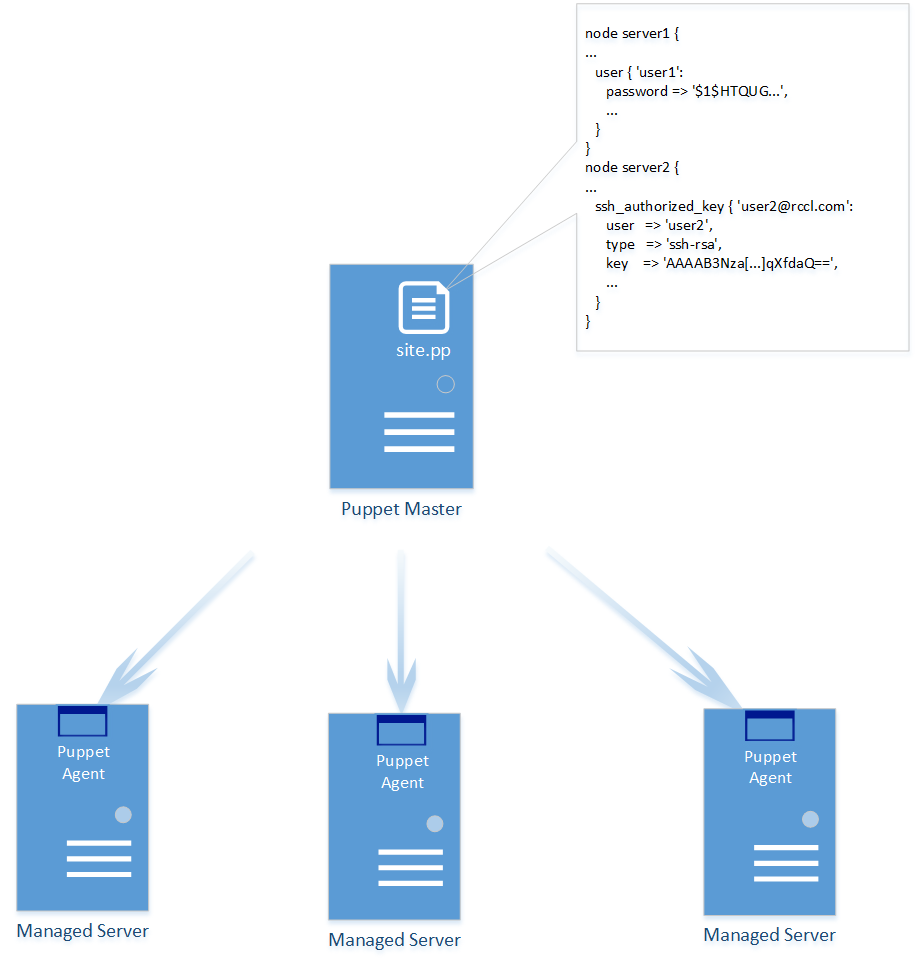
****

Manage MSH Account

1. Current Password/Key Management with Puppet

The passwords and keys on Linux platform are managed by Puppet, agent on each server periodically pulls the latest configuration from central server and apply the changes.

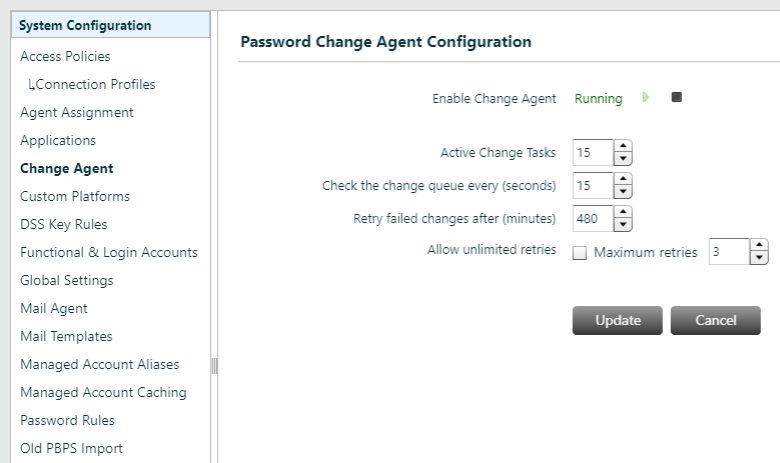


Linux team agreed to take out the password/keys from puppet manifest, and let PS fully take control of password/key management.

1. MSH Account & the Challenges

Linux team currently runs script with MSH account on a jump box to do server management. All 5K + servers share the same SSH key. We’ll use the PS synched accounts feature to set the same keys for the MSH account, the challenge here is how to handle so many servers, and even worse, servers on ship may not be reachable all the time.

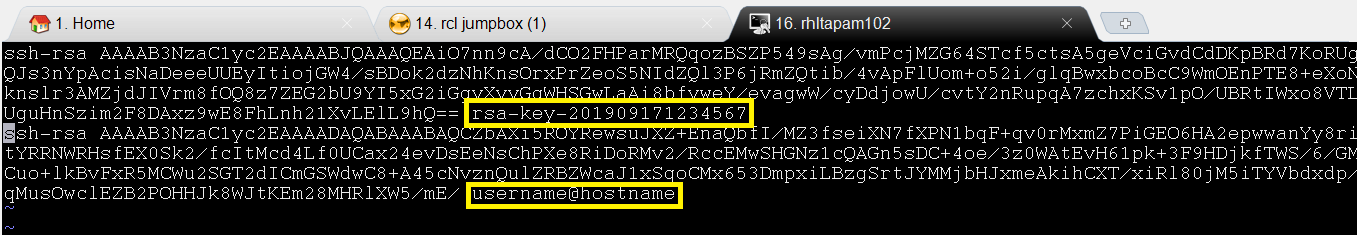
We did a performance testing in AWS environment to sync 3000 keys, it took about 1 ~ 2 hours to sync 1000 keys, and we also verified PS did retry later when host was not reachable. We can set the max retries = (rotation period / retry after minutes)



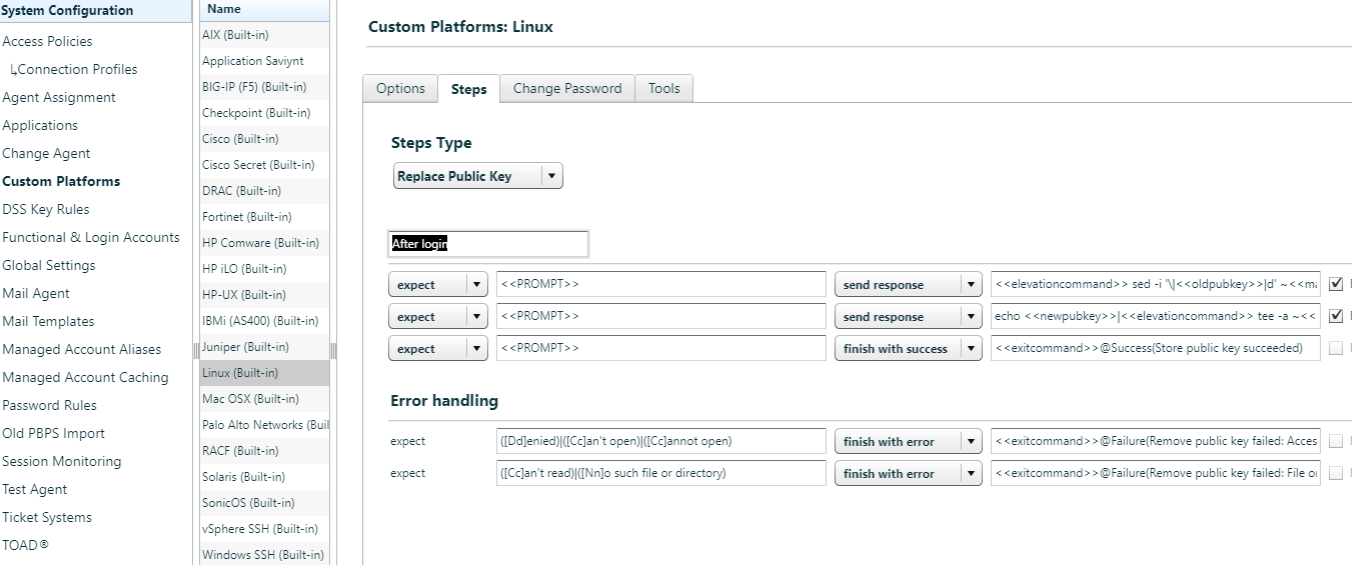
1. The Solution

To make sure no interruption when Linux team run their scrips, we plan to use a last-two-keys solution.

DSS allows multi key entries in .ssh/authorized\_keys, user can use corresponding private key of any public key listed in this file to authenticate, below is a sample, the one with alias = username@hostname is the one set by PS.



When PS rotates key, it finds the last key in this file, deletes the line if exists. Then append the new key to the file. All other keys are not touched. This is controlled by the platform settings.



To make the key rotation smooth for Linux team, we can modify these commands settings so that both the last key and new key are kept in the authorized\_keys file.

<<elevationcommand>> sed -i '\|rclpamlast2key|d' ~<<manacctname>>/.ssh/authorized\_keys

echo <<newpubkey>>rclpamlast2key|<<elevationcommand>> tee -a ~<<manacctname>>/.ssh/authorized\_keys

echo <<oldpubkey>>rclpamlast2key|<<elevationcommand>> tee -a ~<<manacctname>>/.ssh/authorized\_keys

Linux team uses a script “getkey.sh” provided by us to retrieve the private key from PS, the script maintains two keys (older key and newer key), and always gives the older key to client. When the key retrieved from PS is different from the newer key, it copies the newer key to older key, and replace the newer key with the one just returned from PS.

With this approach, the PS server has enough time to sync all the child keys to the parent key. There’s a monitoring script running to make sure all children keys are synced before parent key rotates to next one.

1. The Deployment Plan

Pick one Linux server as the asset for the parent key, and mange it. Run the MSH onboarding script to set up all the children keys.

Note: Linux team will provide an excel file for the server list, the onboarding script makes sure all the servers are managed by PS.

After some time, all children keys are synced to parent.

Note: at this time, all old keys managed by puppet still exist so no interruption for Linux team’s daily work.

Force the parent key change (from UI), and verify all keys can be in synced within one rotation cycle.

At the end, ask Linux team use Puppet to take out the old key, and start using the script provided by use to retrieve key from PS.

1. Deploy getkey.sh

Copy getkey.sh and script.properties into a folder, also create an empty file named “empty”, this is used to provide passphrase for newly retrieved key file, if empty then no password protection of the key file.

Check if puttygen exists, we use it for converting key format. If not exist, run this command to install (or Linux team can install their version)

sudo rpm -ivh http://download-ib01.fedoraproject.org/pub/epel/7/x86\_64/Packages/p/putty-0.71-1.el7.x86\_64.rpm

Below is the configurations in script.properties

#==============================================================================

# Server properties

#==============================================================================

pbps.server.api.uri=https://devbtuvm01/BeyondTrust/api/public/v3/

#==============================================================================

# Authentication properties

#==============================================================================

pbps.auth.key=41ec32ea2b9946d880e2ad5a72d27f3b67d0ea50b88ca3821e3590b64760f33a47370013af893f480af573ee2737ff29038672022385af07c1b20e13f72f64a4

pbps.auth.user=api\_onboard

#==============================================================================

# Managed Account details

# Change the Managed Server and Managed Account information below to

# retrieve the credentials. Also ensure the DSS Auth variable is correctly

# configured (0 for password and 1 for DSS Key)

#==============================================================================

pbps.api.managed.server=rhltapam102

pbps.api.managed.account=user990

#Set DSS Auth to 1 to enable retreiving DSS Key isntead of password

pbps.api.account.dssauth=1

#==============================================================================

# REST API Request Body

#==============================================================================

pbps.api.content.type=Content-Type: application/json

# OUTPUT FILE

ppk.out=/apps/pamonboard/msh/key.ppk

pem.out=/apps/pamonboard/msh/key.pem

Set the correct host, API key, api user etc.

pbps.api.managed.server

Set this value to the asset name of the parent account.

pbps.api.managed.account

Set this value to the mshdaemon account name.

The script can output two types of key format, putty and open SSH, set the output file name in the script.properties, if the output file is not set, no key will be generated for that format.

# OUTPUT FILE

ppk.out=/apps/pamonboard/msh/key.ppk

pem.out=/apps/pamonboard/msh/key.pem