Managing User Logins for Amazon EC2 Linux Instances

September 2018

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

Public key and private key pairs are used to log in to Amazon Elastic Compute Cloud (Amazon EC2) Linux instances and provide robust security. The process to manage user logins can be manually intensive if you have many EC2 Linux instances and many users. Simplified management of user logins is natively available for EC2 Windows instances, but not yet for EC2 Linux instances.

This whitepaper describes a method to automate the process to grant and revoke login access to users across multiple EC2 Linux instances. The description is based on Amazon EC2 Linux, but can applied, with minor modifications, to other types of Amazon EC2 Linux instances. The required steps and commands are described in this whitepaper and can be captured in a script or program. You can then use the script or program as a tool to automate and simplify login management on other Amazon EC2 Linux instances.

The target audience for this whitepaper includes Solutions Architects, Technical Account Managers, Product Engineers, and System Administrators. All references in this whitepaper to EC2 instances refer to Amazon EC2 Linux instances, unless otherwise stated.

Introduction

Amazon Web Services (AWS) generates a public key and private key (key pair) for logging in to each Amazon Elastic Compute Cloud (Amazon EC2) Linux instance, which is an extremely robust security design. The key pair is used for the Secure Sockets Layer (SSL) handshake. It enables a user to log in to an Amazon EC2 Linux host with an SSH client, without having to enter a password.

Use Key Pairs for Amazon EC2 Linux Logins

For Amazon EC2 Linux instances, the default user name is *ec2-user*. The public key is stored on the target instance (the instance that the user is requesting access to) in the *ec2-user* home directory (`~ec2-user/.ssh/authorized_keys`). The private key is stored locally on the client device from which the user logs in, for example: a PC, desktop computer, tablet, Linux host, or Unix host.

Typically, the private key for an Amazon EC2 Linux instance is downloaded by the users who are authorized to log in to that host. For login access to a new EC2 Linux instance, you can either generate a new key pair or use an existing key pair. Key pairs can either be generated on the AWS console or created locally. The public key of a locally generated key pair can be given a unique name and uploaded to AWS from the AWS Command Line Interface (CLI). Thereafter, that key pair can be used to log in to newly created EC2 instances, but only as *ec2-user*.

The Challenge

Although using key pairs to log in to EC2 instances is very robust, efficiently managing access to multiple instances for many users with key pairs can be manually intensive and difficult to automate. To simplify the process to manage access to your Amazon EC2 *Windows* instances, you can integrate your EC2 Windows instance with Active Directory. You can grant or remove the login access of one user or a group of users to a Windows instance or a group of Windows instances.

Currently, however, AWS does not support integration of EC2 *Linux* instance logins with Active Directory or a Security Assertion Markup Language (SAML) compliant authentication repository, such as LDAP.

Imagine a scenario where ten users have access to one Linux instance, and each user logs in to the server as *ec2-user* with the same private key. In a situation where one user's access to this instance must be removed, you would typically have to complete these steps:

1. Generate a new key pair.
2. Log into the instance and replace the old public key with the new public key.
3. Distribute the new private key to the remaining nine users.

This process is manual and must be repeated each time you must remove access to the instance for any of the ten users. This can be tedious if there are many EC2 Linux instances, if you need to temporarily grant user access, or quickly revoke user access, without impacting other users, for example: in a production environment.

Next, imagine a scenario in which there are ten EC2 Linux instances that share a single key pair. In this case, a user who has access to one instance automatically has access to all ten instances. One method to provide more granular login access control is to create ten different key pairs—one for each instance—so that a user only gets the private keys to the specific instances to which that user needs access. Although this provides granularity, it makes private key management difficult. For example, if the user needs to log in to a hundred different EC2 instances, he will need 100 different private keys.

Furthermore, even with a unique key pair for every instance, if a user's access to an instance must be removed, you still face the problem of recreating, reinstalling, and redistributing new key pairs to all the users that have login access to that instance. To remove user access to a large fleet, for example 100 EC2 Linux instances each with a unique key pair, you must create and distribute 100 new key pairs to each user that already has login access to those instances. For medium to large environments, login access management, key distribution, and tracking can become complicated and time consuming.

In addition, every user authorized to log in to a Linux instance does so with the *ec2-user* account, which is root by default. That means that *ec2-user* can run *any* command with sudo. This might not always be desirable. You might want to grant a user root login access to EC2 Linux instances in development, but limit the commands that the user can run on production

instances. For example, you might want to prevent a user from performing mount or unmount operations on Amazon Elastic File System (EFS) in production, but permit it in development. (Although mount privileges for Amazon EFS can be limited through root squashing, it is preferable to have finer control by attaching sudo privileges to the user, especially when granting access to a production environment).

The Solution

The solution to the preceding challenges is to give each user a unique login name, and a single key pair with which to log in to every EC2 Linux instance to which that user is granted access. The user gets a unique home directory on every EC2 instance to which that user has login access. This directory has the same name as the user's login name.

This design greatly simplifies login management: Granting a user login access to an EC2 Linux instance simply requires creating a home directory for that user on that EC2 instance and placing the user's public key in that home directory. No other users with login access to that instance are affected.

Conversely, when you have to remove a user's login access to a specific EC2 instance, you can simply delete that user's public key from that user's home directory on that EC2 Linux instance. Again, no other users with login access to that instance are affected. If you want to temporarily grant login access to a user, you can generate a new key pair, place the public key in the user's home directory, and securely send the private key to the user. This significantly reduces the overhead associated with key distribution to many users. To *purge* a user from an instance, delete that user's home directory (which should be backed up if it contains files, scripts, or data) which also removes the user's login access.

Lastly, each user does not need sudo root privileges on every instance, but can have different sudo privilege levels on different instances. For example, sudo can be set to default to root (unlimited permissions) but can be modified to allow only a limited set of commands on a specific instance or group of instances. This is controlled entirely through the sudo configuration file on the EC2 instance, which is typically `/etc/sudoers` or `/etc/sudoers.d/cloud-init` for Amazon Linux. This file can be modified by a root user to set sudo privileges for any user.

To give a user access to an EC2 instance, complete these steps:

1. Log-in to the target EC2 instance as root (*ec2-user*).
2. Create the user's login and home directory.
3. Generate a key pair and place the public key in the user's home directory.

   If the user already has a key pair, copy the public key of that key pair to the user's home directory on the target EC2 instance.
4. Modify the configuration file `/etc/ssh/ssh_config` to disable password login, and allow only ssh login by key pair.

5. Modify the `/etc/sudoers.d/cloud-init` file to grant the required sudo permissions to the user.
6. Securely send the private key to the user.

The user will then be able to log in to the instance with a unique login name and private key.

To simplify the process so you can easily repeat it for each user, you can complete these steps manually and capture the steps in a Bash or Python script:

1. Log in to the target EC2 instance and run the commands to create the user account.
2. Set sudo permissions for the user account.
3. Grant login access with a key pair.

The script takes a user's login name as input so, when you run the script on any target EC2 instance, it grants login access to that user for that specific instance. The process to login to the target instance and run the script is usually manual and interactive (over SSH), but can be automated with a wrapper script written in *Expect*.

When you run the Expect wrapper script, you don't have to manually log in to each target EC2 instance to run the commands that create the user and enable the user to log in. By automating this process, an administrator can grant or revoke access to any number of users for any number of instances.

Expect is public domain software that enables you to automate control of interactive applications, such as SSH, SFTP, SCP, FTP, passwd, etc. These applications interactively prompt and expect a user to enter keystrokes in response. This is the case with SSH (secure socket shell), the protocol typically used to securely log in to EC2 Linux instances. When you use Expect, you can write simple scripts to automate SSH interactions. This makes it ideal for automating interactive logins to Linux, which does not have a login API. Several languages either have ports of Expect, for example: Perl (expect.pm), Python (pexpect), and Java (expect4j), or have projects implementing Expect-like functionality. All Linux versions come installed with Expect, which is also available on Windows.

An Expect Example

One example of how you can use Expect to automate your actions with scripts, is a connection to an anonymous FTP server.

To make a manual FTP connection to an anonymous FTP server, you would typically follow these steps:

1. Open a connection to the FTP server.
2. At the name prompt, type `anonymous`.
3. At the password prompt, type your email address to get to the FTP prompt.
4. From the FTP prompt, select whether to download, upload, or list files.

But, instead of manually making the connection to the FTP server, you can run the following script to automate this interaction with Expect. The script connects you to the FTP server and then runs the *interact* command, which gives control to the user. The variable in the second line of the script, `$argv`, takes the name or IP address of the FTP server as command line input.

```
#!/usr/local/bin/expect --
spawn FTP $argv
expect "Name"
send "anonymous\r"
expect "Password:"
send chiji@amazon.com\r
interact +
```

Expect is based on a subset of TCL, which can be used to write large and complicated programs. However, all the commands that give the user login access are included in the Bash script, which runs on the target EC2 instance. The Expect wrapper script simply automates the connection to each instance, and runs the Bash script that is on each instance. This uses simple Expect syntax and is relatively simple to write. Additional program features, such as setting timeouts and checking command line inputs, which need to be included in the wrapper script for robustness, might require more advanced syntax.

Granting Login Access: Steps and Commands

Scalable management of multiple user logins to a target Amazon EC2 Linux instance requires user account creation, key pair generation or public key retrieval, public key installation, sudo configuration, password login disablement, and user private key distribution. To grant a user, *John Smith*, with the login name *johnsmith*, login access to a target EC2 instance with the IP address 10.10.10.100 you must complete the following steps.

Login to the Target EC2 Instance

Log in to the target EC2 Linux instance as *ec2-user* with an SSH client, such as PuTTY from a Windows host, or the default SSH client from a Linux or Mac host. You must have the private key for the target EC2 instance on the device from which you log in. If you log in from a Linux host, the private key has a .pem extension, but PuTTY requires a .ppk extension. Use the PuTTYgen client to convert the .pem private key to a file with a .ppk extension.

From a Linux or Mac host, the command to log in to the EC2 target instance is:

```
$ ssh –i /path-to-private-key ec2-user@10.10.10.100


ECDSA key fingerprint is
d3:f2:70:3c:2b:cf:2b:c3:94:e4:94:74:dc:5c:97:4f.


Are you sure you want to continue connecting (yes/no)? Yes
[ec2-user@ip-10.10.10.100] $
```

Create the User Account

After you log in to the EC2 Linux instance, you create the new user account. To create this account, you run these commands:

1. Log in at the root level.

```
$ sudo –i
```

2. Change the user creation configuration so that the `~/.ssh` directory is created for each new user.

```
$ mkdir /etc/skel/.ssh
```

3. Set the default permissions for the .ssh directory of each new user.

```
$ chmod 700 /etc/skel/.ssh
```

4. Create the user *johnsmith*, add him to the *rootusers* group (for sudo root access), and include a summary of John Smith's role.

```
$ useradd -c "John Smith -- Engineer" -d /home –k /etc/skel –
m -g rootusers \ [-G other_group] Johnsmith
```

5. `Su` to the new user, *johnsmith*, log in to his `~/.ssh` directory, generate a key pair, and install the public key on the host.

```
$ su - johnsmith

$ cd ./.ssh
$ ssh-keygen –t rsa –b 2048 –f ./.johnsmith -N ""
```

   John Smith can then use this key pair to log in to every EC2 instance to which he is granted access.

   To revoke access, you can simply delete the public key from his home directory on the target host.

   To grant access, you must create an account on the target instance (if one does not exist) and install his public key in his home directory.

6. Return to the root level.

```
$ exit
```

Next, you move the private key to the *ec2-user* home directory and rename it. This is a security measure. The private key will be securely sent to the user from the *ec2-user* home directory, or written to a keys database for later distribution to the user.

1. Move the private key to the *ec2-user* home directory.

```
$ mv ~johnsmith/.ssh/johnsmith ~ec2-user/.privk_johnsmith
```

2. Rename the public key to *authorized_keys* and set the correct permissions (600) to enable the SSH connection.

```
$ mv johnsmith.pub authorized_keys

$ chmod 600 authorized_keys
```

3. If the user already has an existing key pair, you can copy the user's public key either from an EC2 instance (to which the user already has login access), from a keys host, or from a keys database to the target host. Install that public key in the user's home directory and change the permissions for the key.

```
$ scp –i /path-to-privatekey johnsmith@host-where-he-has- \
login/.ssh/authorized_keys johnsmith.pub

$ mv johnsmith.pub authorized_keys

$ chmod 600 authorized_keys
```

User account creation with the login name is complete. The login name should be the same as the user's Windows desktop or Corporate Account Login user name. After the account is created and sudo permissions are set, the user can log in to the instance with the new login name and private key.

New Key Installation and Rotation

If the private key is lost, perform these steps to remove and replace the lost key:

1. Delete the user's public key from the home directory on each EC2 instance that the user has access to.
2. Generate a new key pair for the user.
3. Install the public key of this new key pair in the user's home directory on the requisite EC2 instances to reinstate login access for that user.
4. Securely send the new private key to the user.

As a mandatory security procedure, you should configure your environment to automatically rotate key pairs at frequent intervals. For more information, see Key Rotation.

Configure Sudo Permissions

For Amazon EC2 Linux, user privileges are defined in the `/etc/sudoers/cloud-init` file. The file usually contains only the entry for the *ec2-user*.

```
ec2-user ALL=(ALL) NOPASSWD:ALL
```

Add the login name *johnsmith* to this file to give him root sudo privileges

```
$ sudo –i

$ echo “johnsmith ALL=(ALL) NOPASSWD:ALL” >> \
/etc/sudoers.d/cloud-init
```

If you have a large number of new users to add to the sudo configuration file, or if you do not want to manually add each user, you can create a file with a list of named user groups and upload that file to each target host. The file must include the associated sudo privileges for the group. When the file is uploaded to the target host, the content is appended to the `cloud-init` file. You can then assign a new user to one of the groups named in the file and the user inherits the sudo permissions of that group.

For example you can create a file named `grp_permissions`, which specifies different permissions for different groups.

```
$ cat grp_permissions

%rootusers      ALL=(ALL)   NOPASSWD:ALL
%dbausers       ALL=(ALL) ALL, !/usr/bin/passwd root
%sysadmin       ALL=(root) /bin/mount, /bin/umount
%operator       ALL=(root) /bin/mount, !/bin/umount /efs
```

The *rootusers* group has full root privileges, so any user added to that group during account creation has full sudo privileges. The *operator* group also has root privileges, but cannot unmount an EFS file system.

Upload the `grp_permissions` file to each target EC2 instance and then use the `sed` command to append the entries in the file to the existing `cloud-init` file. Make sure to verify that the entries in the file do not already exist in the `cloud-init` file.

```
$ sudo –i

$ cd /etc/sudoers.d
$ /bin/sed –i –e '$r grp_permissions' ./cloud-init
```

Because the sudo syntax is complicated, and a syntax error might make it impossible to log in or run sudo on the instance, make sure that you are logged in as root through a separate terminal when you modify the `cloud-init` file (`/etc/sudoers` for other Linux versions. Errors can be corrected if you are already logged in as root; you can either correct the incorrect entry or overwrite the file with the backup file version. You should always create a backup copy before you modify the `cloud-init` file.

All sudo entries or edits should be made with *visudo* (not *vim*), which reviews new entries for syntax errors. If it finds an error, it gives you the choice to fix the error, to exit and not save the changes to the file, or to save the changes and exit. The last choice is not recommended, so visudo marks it with **(DANGER!)**.

Because you can break sudo when you update `/etc/sudoers.d/cloud-init` from a script, new sudo configuration changes, additions, and customizations should first be tested on a non-production host. Sudo configuration files that have been tested and work correctly, should then be checked in to a version control repository, such as AWS CodeCommit, from which all production deployments should be sourced.

Sudo with LDAP

Sudo permissions can also be defined in LDAP, which can synchronize the `sudoers` configuration file in a large distributed environment.

To define sudo permissions in LDAP:

1. Rebuild sudo with LDAP support on each EC2 instance.
   You can choose to rebuild one EC2 instance and then create an Amazon Machine Image (AMI), the *Golden Image*, which you can use to spin up other EC2 instances.
2. Update the LDAP schema.
3. Import the `/etc/sudoers.d/cloud-init` file into LDAP.
4. Configure the sudoers service in `nsswitch.conf` with this command:

```
sudoers:  files ldap
```

For more information, see the *sudoers.LDAP* manual page.

The sudo with LDAP method has many benefits:

- Because there are only two or three queries per invocation, it is very fast.
- Data loaded into LDAP always conforms to the schema. So, unlike sudo, which exits if there is a typo, sudo with LDAP continues to run.
- Because syntax is verified when data is inserted in LDAP, locking is not necessary with LDAP. This means that visudo, which provides locking and syntax verification when the `/etc/sudoers.d/cloud-init` file is modified, is no longer needed.

For information about how to use a keys host, see Step 5 in Script Development: Linux Commands and Code Samples.

Automation: The Process

Normally, to complete the process to grant a user login access to a specific EC2 instance, an administrator or root user must manually log in to each target instance and run all the commands described in the preceding sections.

The Expect wrapper script automatically logs in to each target EC2 instance, uploads the Bash script, and then runs it to create accounts and the sudo permissions that enable login on that target for the specified users. This eliminates the need to manually log in to each target instance to run the script.

Information to include in the Expect wrapper script can be provided as a .csv flat input file with the format in Figure 1 below.

Instance IP Address	User Login Name	User Full Name	User Role	User Groups	Action
1.1.1.1	johnsmith	John Smith	SA	rootusers, users	add
1.1.1.1	heidismith	Heidi Smith	Supermodel	users	add
1.1.1.1	abejohn	Abraham John	Senator	dbausers	remove
2.2.2.2	alberteinstein	Albert Einstein	Scientiest	rootusers	add
2.2.2.2	galoisevariste	Galois Evariste	Math Whiz	mlusers	add
3.3.3.3	genghiskhan	Genghis Khan	Mongol	rootusers	remove

Figure 1

When the Expect script is run against this input file, it takes the information for each user (login name, full name, user group, and user role), logs in to each instance with the admin's private key, and runs the Bash script. This creates a login account (if one does not exist), and adds or revokes access for each user to that instance.

The Expect wrapper script and Bash script jointly constitute the base tool for managing EC2 Linux login access. However, they must be integrated into the authorization and security processes of the organization to be used robustly for instance management.

Figure 2 below is a flow chart that shows the typical steps to grant a user login access to a specific EC2 instance. It starts at user account creation, continues through setting sudo permissions, and finishes with login account testing. The same steps are performed—either serially or in parallel—to grant or revoke user login access to multiple instances.

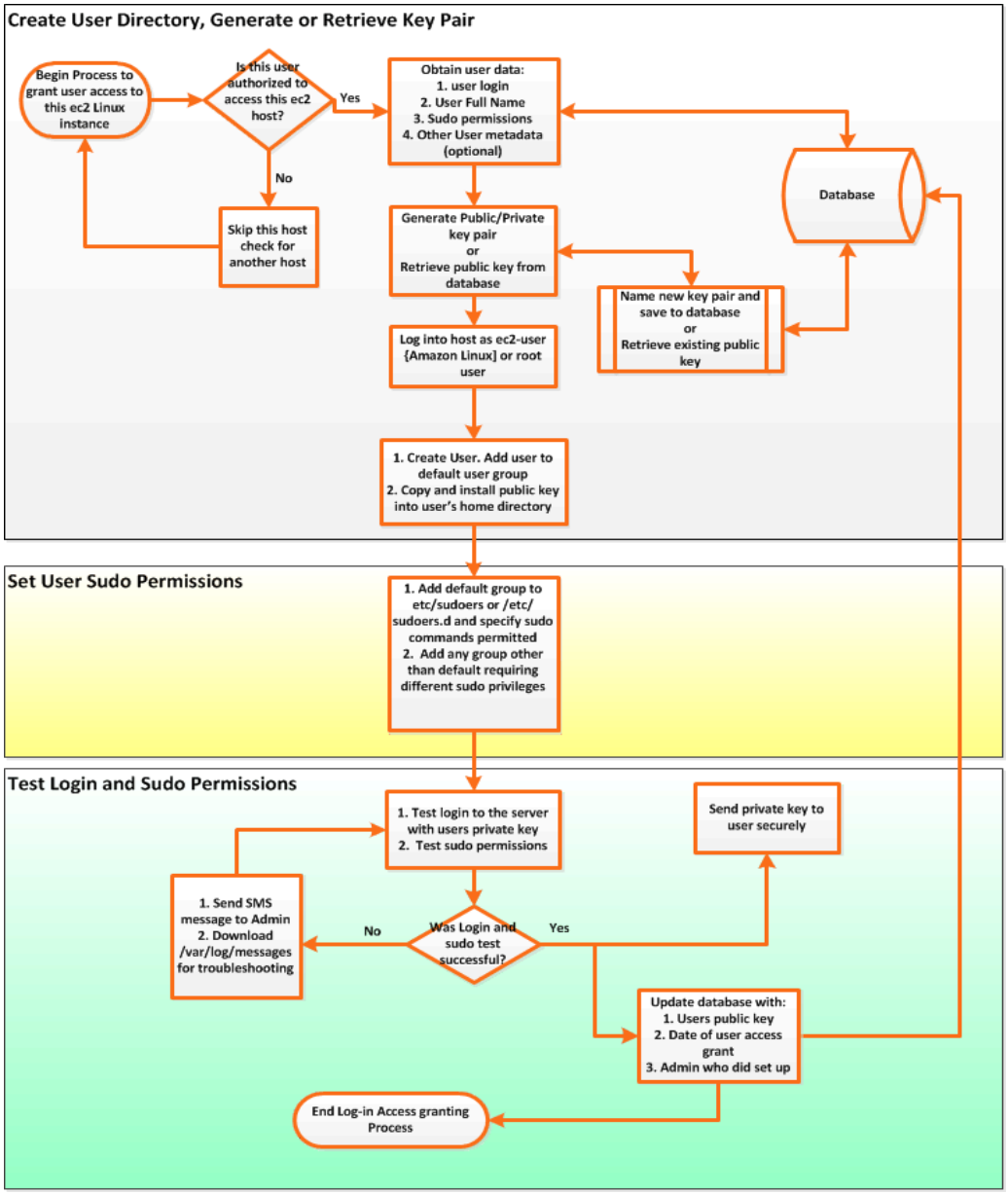

Figure 2: Steps to create and grant login access to an EC2 Linux instance

This document does not include information about the specific internal processes that determine which EC2 instances a user can access, and which commands the user is authorized to run, because they are included in the company's security and access policy. Additional steps might be added as necessary to conform to any custom security or management requirements for the environment. Companies and individuals are advised to review AWS Security Best Practices and make sure they understand how to properly secure their environments.

Script Development: Linux Commands and Code Samples

To successfully automate user logins to make sure that the process is both robust and secure, the automation scripts must be created correctly and the procedures must be correct. To complete this process, the administrator who runs the scripts must perform all the procedures that follow.

Confirm Authorization and Network Access

1. Confirm that login access for the user to the target Linux instances has been authorized through the requisite internal processes.
2. Confirm that the remote Linux host that runs the Expect wrapper script is able to connect to each target Linux instance to which the user is to be granted access. This is important if the target instance might not have a public IP address. In that case, the host from which the wrapper script is run must be on the same network as the target instance.
3. Confirm that after the administrator logs in to each target instance as *ec2-user*, the administrator can `su` to root because the user creation script must be run as root.

Create User, Generate Key Pair, Install Public Key

1. Log in to the target server as `ec2-user` and run the Bash script.

   This creates the user account, a home directory with an .ssh directory, and sets correct directory permissions (700).

   a. Add the user to the group from which to inherit sudo permissions.

   b. Generate or retrieve a key pair and install the public key in the user's home directory.

   c. Set the correct permissions on both keys and move the private key to a secure repository.
   `~ec2-user/.privk` temporarily holds the private key for download before it is deleted.

   For the specific login name *johnsmith*, these commands create the *johnsmith* user account, generate a key pair, and install a copy of the public key in johnsmith's home directory. Both keys are then moved to a subdirectory on the target host (`~ec2-user/.privk`). This is the local key directory and is defined in the variable `LOCAL_KEYS_REPO` in the Bash script. Newly created keys are downloaded from this directory on the target host. The private key is then securely forwarded to the user, and copies of the private key and public key are moved to the database or repository

path specified in the variable KEYS_DATABASE. A user or application with proper IAM permissions can retrieve this public key from the repository or database and install it on any target EC2 instance to which login access for the user is required.

The following variables refer to addresses or paths to the locations of users' SSH keys and must be set in the Bash script.

```
$KEYS_DATABASE="path-to-pubic-keys-location-for-all-users"
$LOCAL_KEYS_REPO="path-to-repository-on-target-instance-that
temporarily-holds-users'-newly-created-key-pair"
$RECEIVED_KEYS_REPO="Directory-on-target-instance-to-which
users'-public-keys-are-copied-or-uploaded"
```

2. If *johnsmith* already has a key pair, retrieve his public key either from a host to which he already has access, or from a keys repository. Upload the public key and the user-creation script to the target instance. The RECEIVED_KEYS_REPO variable specifies a directory on the target instance to which a user's *existing* public key should be uploaded.
    - For automation, when the *user-creation* script runs on the target Linux instance, it first verifies if a public key for the user is present in the RECEIVED_KEYS_REPO directory. If it is not, the script generates a new key pair, installs the public key, prompts the admin on the remote host to download both keys from LOCAL_KEYS_REPO, and then deletes both keys.
    - If a public key with the user's login name is present in the local RECEIVED_KEYS_REPO directory, then that public key is moved to the .ssh directory of that user on that instance to grant login access to the target instance.
    - The keys database address is included in the shell variable KEYS_DATABASE and keeps the login data for each user (full name, login name, public key, private key, authorized hosts, sudo permissions, and other user metadata). The KEYS_DATABASE could refer to a .csv file in S3 or an AWS-managed relational database, such as Amazon Relational Database Service (Amazon RDS), which provides six familiar database engines to choose from (Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, and Microsoft SQL Server). Amazon Aurora is a MySQL-compatible relational database engine that combines the speed and availability of high-end commercial databases with the simplicity and cost-effectiveness of open source databases. Amazon Aurora provides up to five times better performance than MySQL with the security, availability, and reliability of a commercial database at one tenth the cost. For more information about the infrastructure design on AWS that provides robust login access management, see Architecture for EC2 Linux Login Access Management.

The Bash script can read the keys database to find the existing public key, and other metadata, for a user. It can also write a new key pair to the keys database.

For greater security, you can create separate keys databases for user public and private keys.

Key Distribution and Testing

1. Download the private key for the user to the keys database.

   To test sudo for this user, update the /etc/sudoers.d/cloud-init file, change to the user, and run a sudo command.

   For example to test a login for john smith who has full root access, run the following commands as *ec2-user* on the target instance:

   Source John Smith's full environment

```
$ sudo su – john smith
```

   Test sudo to root. For a user with limited access, the explicit sudo commands permitted should be tested.

```
$ sudo –i
```

2. Securely send the private key to the user for logins to all EC2 Linux instances to which the user has been granted access. The user's public key will thereafter be used to grant login access to EC2 Linux instances. To remove the user's login access, remove the `authorized_keys` file from the .ssh directory in the user's home directory.

Two Sample Scripts

The commands to automate user login access that are described in this document are included in a .zip file with two working scripts: the **user-creation** Bash script and the **auto-instance-connect** Expect wrapper script. You can download both scripts from my S3 bucket, unzip them, and test them on an EC2 Linux host.

- The Bash *user-creation* script runs the actual commands required to create the user and grant login access on a target instance. These are the commands captured when you log in to the target instance and manually perform these operations. The script takes a user login, the IP address of a keys host, and the action to perform (add or remove login access of a user to the instance) as input. It then creates the user account, generates a key pair or retrieves the user's existing public key, installs the public key in the user's home, sets the user's sudo privileges, and tests the user

account. To revoke user access, it simply removes the user's public key from the user's home directory on the target instance. It must be runs as root.

A *for-loop* in the Bash script can be used to revoke or grant login access to multiple users for the same EC2 instance.

- The Expect *auto-instance-connect* script automates connection to each target instance. It is designed to be run by an administrator from a *remote* Linux host and connects to each *target* Linux instance (the instances to which the user is to be granted login access) to configure user login access. It uploads the user-creation Bash script to the instance (if it is not already installed), and then runs it with the required command line inputs—user's login name, keys host IP address, add or remove access—to grant or remove login access to the instance for the specified user. Because the Expect script simulates the interactive commands required to make an SSH connection to an EC2 instance and run commands, it requires the path to the private key of the *ec2-user* user who runs it. A *For-loop* in the Expect wrapper script can be used to connect to multiple instances to grant or revoke user login access to one or more users.

  This Expect wrapper script is typically run from an EC2 Linux instance but can also be run from a Windows host. The latter requires installation of Expect and SSH packages for Windows. For more information, see Further Reading.

After you run both scripts, the user is granted login access, with a unique login name, to the specified EC2 instances. The login name can be the same as the user's single sign-on (SSO) login name, and can be given root or other limited sudo privileges.

Both scripts illustrate automation of the process to grant or revoke login access. They must be modified before they can be used in production. For example, you could add logging, robust failure recovery, etc. Administrators and developers might also need to modify the scripts for use on other EC2 Linux versions or for custom management and security needs.

Instead of uploading the *user-creation* Bash script to each target Linux instance at runtime, you can preinstall the script on each instance (include it in an Amazon Machine Image), install it as an RPM package on the EC2 Linux instance, install it from *ec2-user* data on initial boot, or install it from a configuration server, such as a Chef server. After it is installed on the EC2 Linux instance, the Expect wrapper script does not have to load the script on to each target instance, instead, it connects to the target and runs the installed scripts.

Architecture for EC2 Linux Login Access Management

Database Tier

To perform automated user logins in production environments, the user login data and associated metadata should be stored in a database. Because there is no requirement for millisecond latencies, and the amount of login data for all users is unlikely to be very large, Amazon RDS is an ideal keys database. Amazon RDS is a managed database service available in a choice of engines. It also provides significant security benefits. For more information, see the Overview of AWS Security -- Database Services whitepaper.

The RDS database instance holds the user data required to provide login access to any EC2 Linux host. A database schema for user logins should include the following tables and fields:

- User table – UserID (primary key), user login name, first name, last name, email, mobile phone number, user role, public key, private key, key pair creation date, admin creator of key.
- Linux host table – Hostnames, IP, FQDN, EC2 type, host function (database, web), environment (production, QA, test)
- Access options table – Group or user (rootuser, poweruser, operator, and custom sudo configurations etc.), sudo permissions, authorization date

You can choose not to store the user's private key in the database. This means that if the user's private key is lost, a new key pair must be generated for that user. Public keys that are rotated are irretrievably lost, and so old public keys should not be retained in the database. Only administrators should have read access to the keys database. The database should also be replicated across Availability Zones (Multi-AZ) for high availability, otherwise it might not be possible to grant access to users if the database is down or unreachable.

Because significant security problems could arise if the users' login data and metadata are compromised, the data in the Amazon RDS database instance should be encrypted at rest. Amazon RDS also supports encrypting an Oracle or SQL Server DB instance with Transparent Data Encryption (TDE). TDE can be used in conjunction with encryption at rest, although using TDE and encryption at rest simultaneously might cause a slight decrease in database performance.

To manage the keys used to encrypt and decrypt your Amazon RDS resources, you can use the AWS Key Management Service (KMS). AWS KMS combines secure, highly available hardware and software to provide a key management system that is scaled for the cloud. With AWS KMS, you can create encryption keys and define the policies that control how these keys can be used. AWS KMS supports CloudTrail, so you can audit key usage to verify that keys are used appropriately.

You can use SSL from your application to encrypt a connection to a DB instance that runs MySQL, MariaDB, Amazon Aurora, SQL Server, Oracle, or PostgreSQL. This provides end-to-end encryption of data in transit and at rest.

Application Tier

The infrastructure for login access management in a production environment can be configured as a standard, three-tier architecture, with the database behind a Security Group that is only reachable from the application server. The application server can be a small T2 instance. The *user-creation* script does not query the database directly, but goes through the application server. The *user-creation* script and the Expect wrapper script can be configured to run only from the application server. You can then limit logins to the application server to specific administrators.

Web Tier

The web tier provides the interface through which users can request login access to specific servers and enables users to securely download their private keys. The web server can also be a T2 instance and should only allow connections over HTTPS. Connections from the web server to the application server can also be encrypted.

Automation Improvements

To simplify administration of user login accounts, you can add a graphical user interface (GUI) in front of the backend. From this interface, you can click a user name and select the group of instances to which you want to grant or remove access for that user. The backend processing is still performed by the **user-creation** script and the **auto-instance-connect** script.

After a production version of the automation script is built with the three-tier architecture discussed in the preceding section, integration with Active Directory (AD) or any SAML 2.0-compliant system is feasible. The target Linux instances, as well as the privileges the user should have on the instances (root or non-root), can be read from your AD server and mapped to the Linux user group that has equivalent sudo permissions. When the user account is created, it automatically inherits the sudo privileges of that Linux group. However, to implement this solution, you must specify a call to the LDAP/ADSI API of your AD server to retrieve the hosts and privileges authorized for each EC2 instance for that user.

The script receives that input and creates the user accounts, adds or revokes their access, and raises or removes permission by updating the group the user belongs to on the target instances.

Use Cases

There are several production use cases for which automated login access management are required.

Ec2-User (Default User) Key Rotation

Key rotation for the *ec2-user* account should be frequent, but is rarely done in production environments simply because of the amount of manual effort required to create and reinstall keys in a moderately large environment. When you use the automated login access scripts, the effort required is significantly reduced, so key pair generation and rotation can be performed more frequently, which significantly improves security. A key pair can be created, named, and imported to the AWS account with the AWS CLI.

The import command is:

```
$ aws ec2 import-key-pair --key-name my-key --public-key-
material \
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuhrGNglwb2Zz/Qcz1zV+
l12fJOnWmJxC2GMwQOjAX/L7p01o9vcLRoHXxOtcHBx0TmwMo+i85HWMUE7aJtYc
lVWPMOeepFmDqR1AxFhaIc9jDe88iLA07VK96wY4oNpp8+lICtgCFkuXyunsk4+K
huasN6kOpk7B2w5cUWveooVrhmJprR90FOHQB2Uhe9MkRkFjnbsA/hvZ/Ay0Cflc
2CRZm/NG00lbLrV4l/SQnZmP63DJx194T6pI3vAev2+6UMWSwptNmtRZPMNADjmo
50KiG2c3uiUIltiQtqdbSBMh9ztL/98AHtn88JG0s8u2uSRTNEHjG55tyuMbLD40
QEXAMPLE

Output:
  {
   "KeyName": "my-key",
   "KeyFingerprint":
"1f:51:ae:28:bf:89:e9:d8:1f:25:5d:37:2d:7d:b8:ca"
  }
```

With this command, the public key is imported to AWS. When you spin up Linux instances, that key pair is available for selection from the drop-down list of existing key pair names on the Console. Any new instance that selects this key pair will have the public key installed on the instance. Administrators with the private key can log in to new instances they did not create as *ec2-user*.

For *ec2-user* key rotation, a new key pair is created, named, and installed on all instances for *ec2-user* with the *user-creation* script. The public key of the new key pair is then imported into AWS with either the console or CLI, the old key pair is deleted, and the new private key is securely distributed to authorized users.

Cross-Environment Access

Regardless of the risk associated with granting login access to production systems to developers or third-party consultants, it is sometimes necessary. To provide root access in those circumstances is dangerous. The automated login access management tools described in this document can give administrators control over the commands that a user can run on a production EC2 instance. An administrator could create a group with limited sudo privileges and add users to this group when accounts are created. The administrator could also remove the user's public key to revoke login access to the instance after the specific task for which access is needed is completed.

Authorization and Permissions for Non-Employees

The capacity to grant or revoke login access to target EC2 instances, and provide granular control over the actions that can be performed by a user on the target instance, offers great flexibility. It is particularly useful when you need to give login access to temporary employees, partners, consultants, software vendors, or applications whose actions on target hosts must be limited. In addition, *every* action performed on the target host by the user can be monitored and captured by a shell, such as sudosh or rootsh, which logs all key strokes. A tracking shell can be specified when the account is created for a specific user on the target instance.

Conclusion

The concepts explained in this document for automating login access, a process which is ordinarily interactive for all types of Linux, and is therefore manually intensive, can be used to develop an application or script that has great benefits and broad utility across the enterprise. Automated login access management will eventually become a native feature of Amazon EC2 Linux instances. For now, developing and using an automation tool will be invaluable to administrators, engineers, architects, system administrators, and account managers for managing user access.

Contributors

The following individuals and organizations contributed to this document:

Chiji Uzo, AWS Solutions Architect

Further Reading

For more information, see these resources:

user-creation script:
https://s3-us-west-2.amazonaws.com/samplescripts/user-creation

Overview of AWS Security – Database Services (whitepaper):
https://d0.awsstatic.com/whitepapers/Security/Security_Database_Services_Whitepaper.pdf

Expect for Windows:
http://docs.activestate.com/activetcl/8.4/expect4win/ex_usage.html#cross_platform

OpenSSH for Windows:
http://sshwindows.sourceforge.net/