SSH Authentication with AuthorizedKeysCommand

Imagine a world where adding or removing a user's SSH access was as simple as a single button in AWS.

I'm not sure about you, but the problem of provisioning and de-provisioning SSH users on a fleet of nodes is a giant pain in the ass. The conventional wisdom is have a file called `~/.ssh/authorized_keys` in the home folder of the user on the server you want to log into. That file will contain the SSH public key of authorized users. When a new engineer joins, you'll have to add her to the system including adding her SSH public key.

If your company is old-school and doesn't use a configuration management system, you know this pain. If you want to provision or deprovision a user, you'll have to execute an SSH script and execute it on every node, then validate that that person is actually out of the system. Typically what you'll see with a configuration management solution like Chef, as an example, is one where each user has their own databag with their public key in it, or there's a flat file of users and public keys. The problem with this approach is that you have a explicitly do a Chef deploy to be able to add or remove users. From a security perspective this is a really bad thing, because in the event of someone's laptop and SSH key being compromised, every second counts.

I'd like to propose a better way of handling SSH keys with any generic public key store. The key to this is the `AuthorizedKeysCommand` in the `sshd_config`. From the man page:

```
AuthorizedKeysCommand
```

```
Specifies a program to be used to look up the user's public keys.
The program must be owned by root, not writable by group or others
and specified by an absolute path.  Arguments to
AuthorizedKeysCommand accept the tokens described in the TOKENS
section. If no arguments are specified then the username of the
target user is used.
```

```
The program should produce on standard output zero or more lines
of authorized_keys output (see AUTHORIZED_KEYS in sshd(8)). If a
key supplied by AuthorizedKeysCommand does not successfully
authenticate and authorize the user then public key authentication
continues using the usual AuthorizedKeysFile files.  By default,
no AuthorizedKeysCommand is run.
```

All this means is instead of giving a SSH public key statically to the server, you give ssh a command that is executed every time someone logs in. The first argument of this command must be the username of the user attempting to log in and the output is effectively an authorized keys file. Let's get your wheels turning on some examples of how this could work at your company.

Let's say you work at a company that uses AWS, take a look at this very simple script that will authorize users based on SSH public keys that are entered in their IAM user profile.

```ruby
#!/usr/bin/env ruby

require 'aws-sdk-iam'

client = Aws::IAM::Client.new
resp = client.list_ssh_public_keys(user_name: ARGV[1], max_items
pub_key_id = resp.ssh_public_keys.first.ssh_public_key_id

puts client.get_ssh_public_key({
  user_name: ARGV[1],
```

Then in your `/etc/sshd_config` you need only add:

```
# /etc/ssh/sshd_config

AuthorizedKeysCommand /path/to/iam-ssh-auth
AuthorizedKeysCommandUser nobody
```

I hope you can see that this is not at all tied to AWS. You could just as easily use this method, with Github, Consul, SQL, or almost any data store you like. Some considerations though is that if you replace you authorized_keys file with this, you need to be very sure that wherever you're pulling data from is always accesible. I'd suggest something that chooses partition tolorance and availability from CAP threorem for this use case.

I hope this has shed some light on a really interesting and not often used feature of SSH.