**Bryan Kennedy**

**BRYAN KENNEDY**

Tinkerer & Doer. Co-Founder of Sincerely

sincerely.com

@plusbryan

feed

**SVBTLE**

**MARCH 3, 2013**

# My First 5 Minutes On A Server; Or, Essential Security for Linux Servers

Server security doesn't need to be complicated. My security philosophy is simple: adopt principles that will protect you from the most frequent attack vectors, while keeping administration efficient enough that you won't develop "security cruft". If you use your first 5 minutes on a server wisely, I believe you can do that.

Any seasoned sysadmin can tell you that as you grow and add more servers & developers, user administration inevitably becomes a burden. Maintaining conventional access grants in the environment of a fast growing startup is an uphill battle - you're bound to end up with stale passwords, abandoned intern accounts, and a myriad of "I have sudo access to Server A, but not Server B" issues. There are account sync tools to help mitigate this pain, but IMHO the incremental benefit isn't worth the time nor the security downsides. Simplicity is the heart of good security.

Our servers are configured with two accounts: root and deploy. The deploy user has sudo access via an arbitrarily long password and is the account that developers log into. Developers log in with their public keys, not passwords, so administration is as simple as keeping the *authorized_keys* file up-to-date across servers. Root login over ssh is disabled, and the deploy user can only log in from our office IP block.

The downside to our approach is that if an authorized_keys

**Bryan Kennedy**

**BRYAN KENNEDY**

Tinkerer & Doer. Co-Founder of Sincerely

sincerely.com

@plusbryan

feed

**SVBTLE**

file gets clobbered or mis-permissioned, I need to log into the remote terminal to fix it (Linode offers something called Lish, which runs in the browser). If you take appropriate caution, you shouldn't need to do this.

*Note: I'm not advocating this as the most secure approach - just that it balances security and management simplicity for our small team. From my experience, most security breaches are caused either by insufficient security procedures or sufficient procedures poorly maintained.*

**Let's Get Started**

Our box is freshly hatched, virgin pixels at the prompt. I favor Ubuntu; if you use another version of linux, your commands may vary. Five minutes to go:

```
passwd
```

Change the root password to something long and complex. You won't need to remember it, just store it somewhere secure - this password will only be needed if you lose the ability to log in over ssh or lose your sudo password.

```
apt-get update
apt-get upgrade
```

The above gets us started on the right foot.

**Install Fail2ban**

```
apt-get install fail2ban
```

Fail2ban is a daemon that monitors login attempts to a server and blocks suspicious activity as it occurs. It's well configured out of the box.

Now, let's set up your login user. Feel free to name the user something besides 'deploy', it's just a convention we use:

```
useradd deploy
```

```
mkdir /home/deploy
mkdir /home/deploy/.ssh
chmod 700 /home/deploy/.ssh
```

### Require public key authentication

The days of passwords are over. You'll enhance security and ease of use in one fell swoop by ditching those passwords and employing public key authentication for your user accounts.

**Bryan Kennedy**

**BRYAN KENNEDY**

Tinkerer & Doer. Co-Founder of Sincerely

sincerely.com

@plusbryan

feed

```
vim /home/deploy/.ssh/authorized_keys
```

Add the contents of the id_rsa.pub on your local machine and any other public keys that you want to have access to this server to this file.

```
chmod 400 /home/deploy/.ssh/authorized_keys
chown deploy:deploy /home/deploy -R
```

### Test The New User & Enable Sudo

Now test your new account logging into your new server with the deploy user (keep the terminal window with the root login open). If you're successful, switch back to the terminal with the root user active and set a sudo password for your login user:

```
passwd deploy
```

Set a complex password - you can either store it somewhere secure or make it something memorable to the team. This is the password you'll use to sudo.

```
visudo
```

Comment all existing user/group grant lines and add:

**SVBTLE**

```
root    ALL=(ALL) ALL
deploy  ALL=(ALL) ALL
```

**Bryan Kennedy**

**BRYAN KENNEDY**

Tinkerer & Doer. Co-Founder of Sincerely

sincerely.com

@plusbryan

feed

The above grants sudo access to the deploy user when they enter the proper password.

**Lock Down SSH**

Configure ssh to prevent password & root logins and lock ssh to particular IPs:

```
vim /etc/ssh/sshd_config
```

Add these lines to the file, inserting the ip address from where you will be connecting:

```
PermitRootLogin no
PasswordAuthentication no
AllowUsers deploy@(your-ip) deploy@(another-ip-if-any)
```

Now restart ssh:

```
service ssh restart
```

**Set Up A Firewall**

No secure server is complete without a firewall. Ubuntu provides ufw, which makes firewall management easy. Run:

```
ufw allow from {your-ip} to any port 22
ufw allow 80
ufw allow 443
ufw enable
```

This sets up a basic firewall and configures the server to accept traffic over port 80 and 443. You may wish to add more ports depending on what your server is going to do.

**Enable Automatic Security Updates**

I've gotten into the *apt-get update/upgrade* habit over the years, but with a dozen servers, I found that servers I logged into less frequently weren't staying as fresh.

**Bryan Kennedy**

**BRYAN KENNEDY**

Tinkerer & Doer. Co-Founder of Sincerely

sincerely.com

@plusbryan

feed

Especially with load-balanced machines, it's important that they all stay up to date. Automated security updates scare me somewhat, but not as badly as unpatched security holes.

```
apt-get install unattended-upgrades

vim /etc/apt/apt.conf.d/10periodic
```

Update the file to look like this:

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Download-Upgradeable-Packages "1";
APT::Periodic::AutocleanInterval "7";
APT::Periodic::Unattended-Upgrade "1";
```

One more config file to edit:

```
vim /etc/apt/apt.conf.d/50unattended-upgrades
```

Update the file to look like below. You should probably keep updates disabled and stick with security updates only:

```
Unattended-Upgrade::Allowed-Origins {
        "Ubuntu lucid-security";
//      "Ubuntu lucid-updates";
};
```

**Install Logwatch To Keep An Eye On Things**

Logwatch is a daemon that monitors your logs and emails them to you. This is useful for tracking and detecting intrusion. If someone were to access your server, the logs that are emailed to you will be helpful in determining what happened and when - as the logs on your server might have been compromised.

```
apt-get install logwatch

vim /etc/cron.daily/00logwatch
```

SVBTLE

add this line:

```
/usr/sbin/logwatch --output mail --mailto test@gmail.com
--detail high
```

**All Done!**

I think we're at a solid place now. In just a few minutes, we've locked down a server and set up a level of security that should repel most attacks while being easy to maintain. At the end of the day, it's almost always user error that causes break-ins, so make sure you keep those passwords long and safe!

I'd love to hear your feedback on this approach! Feel free to discuss on Hacker News or follow me on Twitter.

**Update**

There's a great discussion happening over at Hacker News. Thanks for all the good ideas and helpful advice! As our infrastructure grows, I definitely plan on checking out Puppet or Chef - they sound like great tools for simplifying multi-server infrastructure management. If you're on Linode like us, the above can be accomplished via StackScripts as well.

**8,225**
**KUDOS**

**Bryan Kennedy**

**BRYAN KENNEDY**

Tinkerer & Doer. Co-Founder of Sincerely

sincerely.com

@plusbryan

feed

**SVBTLE**

READ THIS NEXT

**Welcome to the interview; please sit down, an...**

← FULL BLOG

**Bryan Kennedy**

**BRYAN KENNEDY**

Tinkerer & Doer. Co-Founder of Sincerely

sincerely.com

@plusbryan

feed

**SVBTLE**