blog.jonliv.es

Jon Cowie's blog

Creating your own Signed APT Repository and Debian Packages

We create a lot of our own debian packages at Aframe where I work, and until recently have been keeping them in a flat repository without properly signing any of our packages. However, since there's a possibility some of those packages may be released publicly (since Opscode may not be providing debian lenny packages for Chef anymore), I decided that it was high time to properly organise the repository and sign all our packages plus the repository itself.

As anyone who has tried this will no doubt have found, there is a large amount of conflicting information out there as to how exactly this can be achieved. To ease the burdens of my fellow sysadmins, I thought I'd gather all the necessary info together into one easy post.

The first thing we're going to need to do is to generate a GPG key to sign our repository and all of our packages with. If you've already done this, please skip this section. If not, you can follow these simple steps!

Creating a GPG Key for Signing Stuff

- Run the following command: `gpg --gen-key`
- Select Option 5 to generate an RSA Key
- Choose your keysize from the options given (doesn't really matter for our purposes
- Enter your name and email address when prompted

There, we now have a GPG key we're going to use to sign everything. The next stage is to produce a ASCII formatted public key file which we can distribute to people so they'll be able to tell apt to trust our repository.

Creating a ASCII Formatted Public Key File

By default, the gpg utility's export function produces a binary formatted output of our public key. If you're planning to distribute your public key via the web, this isn't very handy so we're going to use the gpg utility's `--armor` option to produce an ASCII formatted version.

You'll want to substitute the email address of the key you're exporting, and the output filename as appropriate in the following command.

- `gpg --armor --export jon@aframe.com --output jon@aframe.com.gpg.key`

Save this keyfile somewhere, we'll be making it available over the web for people to add to their apt keychains – this is what'll say that our repository is trusted for package installs.

Signing Some .deb Packages

I'm going to assume that you already know how to create .deb packages, by way of keeping this blogpost short. This section will simply cover signing a package you're creating and resigning an existing package.

The good news is that if you've already generated a GPG key as detailed above, your packages will be automatically signed as long as the name and email address in your package's changelog file are the same as that of the GPG key you created. This means that simply running `dpkg-buildpackage` will now give you signed packages. If you're unsure of how to build debian packages at all, there's plenty of information out there on doing this. I might write a blog post on that soon 🙂

If you want to resign an existing debian package, for example if you're setting up your own backport of a package (as with my usecase, backporting Chef 0.9.16 into debian), then this is very easy too if you already have a GPG key set up. We use a tool called dpkg-sig.

Install the dpkg-sig tool by running the command

- `apt-get install dpkg-sig`

Then run the following command to sign the package, substituting the name of the deb package as appropriate.

- `dpkg-sig --sign builder mypackage_0.1.2_amd64.deb`

The “builder” name mentioned is a debian convention, indicating that the builder of the package signed it. The GPG key used will be the one you set up above, providing you’re running the command as the same user you set up the key with.

Creating a Signed APT Repository

OK, so you’ve got a bunch of signed packages, but now you need some way to make them available to the world. Preferably one which doesn’t throw up authentication warnings every time they try to install from your repository. What you need is a signed repository!

The first step to creating your repository is to create a base directory for it to live in. Doesn’t matter where, any directory will do. Now inside that directory, create a folder called `conf`.

Inside your `conf` folder, you’re going to create a text file called `distributions`. Below is the one I’ve used for my repository:

```
Origin: apt.aframe.com
Label: apt repository
Codename: lenny
Architectures: amd64 source
Components: main
Description: Aframe debian package repo
SignWith: yes
Pull: lenny
```

Some of the above will be self explanatory, for example if you're not packaging for debian lenny, you'd replace all occurrences of lenny with squeeze, for example. Additionally, if you're going to package for more than one distribution, you'll want to copy the above section for each distro. It all stays in the same file, you just change the "lenny" parts as applicable.

Also, since I'm only creating 64-bit packages, I've said that my repository will only contain the amd64 and source architectures. If you're packaging for i386 or other architectures, remember to add them!

The important line in the above example is the one that says `SignWith: yes`. That line states that we want to sign the repository with our GPG key. Again, if you're running commands as the same user you used to create your GPG key, this should work fine.

Now that we have the `descriptions` file all ready to go, we're going to use a tool called `reprepro` to add our packages to the repository. You can install reprepro by running the following command:

- `apt-get install reprepro`

The nice thing about this tool is that it will create all the structure it needs inside the repository automatically so you don't need to worry about it. Here's an example of how to add a package to our repository. **PLEASE NOTE** you have to run the below command from your repository directory, ie the directory containing the `conf` folder.

- `reprepro --ask-passphrase -Vb . includedeb lenny /home/joncowie/libjavascript-minifier-xs-perl_0.06-1_amd64.deb`

So what does this command mean? Well, the `--ask-passphrase` part tells it to prompt you for the passphrase for your GPG Key. You did set one, right? The `-Vb . includedeb lenny` part tells the command to be verbose, and set the base directory for the command as the current directory. It then says we're going to be passing the command a deb file (that's the `includedeb` part) and then says that we're going to be adding it to the `lenny` distribution in our repository. The last part is the absolute path to the .deb package we're adding.

When you run this command, you'll see a bunch of output telling you that various folders have been created, and once you've entered the password for your GPG key your package will be tucked away nicely in your properly structured, properly signed debian repository. Running the above command on a further package or packages will add the new package(s) into the existing structure. You'll probably now notice that your repository directory is structured much more like the official debian repositories – that's because it's now structured "The Debian Way".

Making your Repository Available to the World

The final section in this blog post is how to make your wonderful new repository available to the rest of the world. This comes in two parts. The first is making your repository directory available over the web. I'm going to assume if you're creating packages you can probably do this part by yourself, so I'm going to skip over it 😉

The second part is making our public GPG key available to the world so they can mark our repository as trusted. I'd suggest keeping the public GPG key we created above in the root of your repository, to make it easy for people to find. Just make sure you only store the *public* part of the key there. That's the part we created using the `gpg --armor --export` command.

I'd also recommend publishing a simple command for people to download your public key and import it into their apt keychain in one step – this makes it nice and easy for them to get up and running with your repo. This command is as follows (change the URL to your key as appropriate):

```
wget -O - http://apt.aframe.com/jon@aframe.com.gpg.key | sudo apt-key add -
```

Once your users have run the above command, they can add your nicely-formatted repo to their `/etc/apt/sources.list` file by adding the following line (change URL and distribution as appropriate):

```
deb http://apt.aframe.com/ lenny main
```

Then they just run `apt-get update` and they're all ready to use your repository containing all of your signed packages – and not an authentication warning in sight.

April 26, 2011 jonlives Linux, packaging

22 thoughts on "Creating your own Signed APT Repository and Debian Packages"

mverwijs

January 11, 2012 at 9:10 am

Man, you just saved me some pain with that reprepro command. I didn't know about the includedeb switch. Thanks!

Also: with the -b switch, you are able to specify the location of the folder that your repository is at. So standing in the root of the repository while executing reprepro is not a requirement. Just replace the '.' with the full pathname to the repository. (Possibly you knew this, but it wasn't clear from the text).

Kindly,

mverwijs

Busso Dario D.

February 10, 2012 at 7:33 am

Great great job!
Dario

Colin

May 31, 2012 at 10:53 am

I could not get this to work:
wget -O – http://apt.aframe.com/jon@aframe.com.gpg.key | sudo apt-key add –

The command I entered is this:
wget -O – http://10.31.31.89/root_repository/public_key_file.gpg.10D3661CEAAFA847 | sudo apt-key add –

I have the repository and it connects using http but I get this:

Connecting to 10.31.31.89:80… connected.
HTTP request sent, awaiting response… 404 Not Found
2012-05-31 07:50:44 ERROR 404: Not Found.

gpg: no valid OpenPGP data found.

Colin

May 31, 2012 at 10:59 am

I was able to do this:

wget -O – http://10.31.31.89/repository/keyfile > shift
sudo apt-key add shift

The only problem I have now is that it is a bad signature. 😟

Colin

May 31, 2012 at 12:22 pm

Solved it had to update my index file and then update the Release file.

K Richard Pixley

June 13, 2012 at 2:58 am

dpkg-buildpackage is apparently not signing my debs, just the dsc and the changes. This is awkward because signing the deb after the changes file is created means that the changes file has the wrong checksums for the deb.

I don't see a solution yet, but if someone else does, that'd be great.

Eugenio Vidal

June 28, 2012 at 7:50 pm

I think that I have the same problem. I get a *.dsc and a *.changes file. I've have imported the public key to the apt ring…. but when I try to install it I get a warning saying that it is an untrusted source. I don't know what is happening. Anybody can help me please?

Maciej Małycha

October 10, 2012 at 7:25 pm

The dpkg-genchanges-hook could be used, but debuild is little bit buggy here and needs to be patched. I've submitted bug report with patch attached here: https://bugs.launchpad.net/ubuntu/+source/devscripts/+bug/1065201

After patching, you can put following line in the ~/.devscripts file:

DEBUILD_DPKG_GENCHANGES_HOOK="dpkg-sig -k –sign builder ../%p_%v_*.deb"

or call debuild with –dpkg-genchanges-hook="dpkg-sig -k –sign builder ../%p_%v_*.deb" option.

Pingback: Hosting a Private Apt Repository on S3 « Zach's Blog

---

Pingback: debian wheezy apt package repository gpg | buntblock

Pingback: Harjoitustehtävä 4: Paketinhallintaa « eliimatt

Pingback: apt Infrastructure with Puppet – Terrarum

Pingback: Linux palvelimena: deb-paketin teko ja repositoryn luominen | Joni Junni

Pingback: Linux palvelimena: Kotitehtävä 6 – Metapaketin luonti (.deb packet) | Tapio Naumanen

Pingback: Kotitehtävä 6. – Metapaketit | juusopaakkonen

Pingback: Kuudes kotitehtävä | Martti Kitunen

Pingback: Hyödyllisiä linkkejä ja komentoja | juusopaakkonen

Pingback: Hyödyllisiä komentoja | juusopaakkonen

bojanahennie

August 19, 2013 at 8:57 am

Hmmm that is an interesting question.

I suspect you could build the package twice using the same orig.tar.gz, and a different .deb + changes for each and upload both. Since the .deb files would have different archs and the same version I think it would work out OK.

Here I think it would work because you can either import the .changes file or the .deb file. So import the .changes file for the x86 package – then afterwards import the x64 .deb file

Failing that hacking the .changes files manually to include both binary packages would presumably work, but would be more of a pain to apply since you'd need to resign the .changes file.

I guess the best thing to do is to try it and see!
vps

Pingback: Crete Package ← Bookmarks

Pingback: Create Package ← Bookmarks

Huxy

February 9, 2014 at 6:58 pm

Reblogged this on CodeApe and commented:
Handy guide to help you quickly get an APT repo up and running. It took me some time until I found a guide as clear as this..

Proudly powered by WordPress