Mirrors

Mirrors hosting our packages.

Like a Linux distribution, XCP-ng's installation images and RPM repositories can be replicated by benevolent mirror providers.

XCP-ng uses mirrorbits to redirect download requests to an appropriate mirror based on their update status and geographical position.

🏢 Original mirror

`mirrors.xcp-ng.org` is the base for all download links, either downloads of installation ISO images, or RPM repositories used by `yum` to download updates.

Previous versions of XCP-ng used to download files directly from `https://updates.xcp-ng.org`, which since then has becomes one mirror among others (and a fallback in case of files missing from other mirrors).

📋 List of mirrors

You can check our live list of mirrors at this URL: https://mirrors.xcp-ng.org/?mirrorstats

TIP

If loading this page fails due to too many redirections, just go to https://xcp-ng.org once, then try again)

📥 Add your mirror

Anyone or any entity can submit a mirror to our approval so that we add it to the list used by mirrorbits. It is a way to contribute to the XCP-ng project!

Prerequisites

In order to guarantee the quality of the mirrors offered to our users, there a some prerequisites:

- Offer HTTPS
  - You need a valid certificate and must renew it in time to avoid downtime.
  - **If your certificate is provided by Let's Encrypt**, please read the dedicated section below.
  - If you already provided a mirror but are unsure if we added it as an HTTP or HTTPS mirror, check http://mirrors.xcp-ng.org/README.txt?mirrorlist&https=1, which selects HTTPS mirrors and lists the others in an "Excluded Mirrors" section.
- Offer read-only `rsync`. Two reasons:
  - Mirrorbits needs this to regularly check the state of the mirror and automatically disable outdated or broken mirrors.
  - This will allow nearby mirrors to sync from yours in the future, if needed.
- Provide both IPv4 and IPv6 addresses
- Sync from a quickly updated mirror (the first mirrors will sync from our main mirror) at least once an hour, preferably four times an hour.
- Sync the whole mirror tree.
- Provision enough disk space for future growth, and monitor available space to avoid sync failures.
- Minimum bandwidth: 100 Mbit/s. Preferably 1 Gbit/s or more. At some point, we may refuse applications that offer only 100 Mbit/s in areas with already enough quicker mirrors available.
- Up 24/24.
- Monitor your mirror to detect and fix any failures. We don't require 99.99% uptime, but broken mirrors cause extra work that we glady avoid if we can. We prefer no mirror rather than an unstable mirror.

If one of those prerequisites is causing an issue to you as a mirror provider, tell us.

How to sync

Here's how to sync from the main mirror. Adapt if syncing from another mirror.

```
rsync -rlptv --delete-delay updates.xcp-ng.org::repo/
/local/path/to/mirror
```

Note: if you sync from our main mirror, rsync access will be unlocked for your host after your application. See below.

Extra steps for mirrors using Let's Encrypt certificates

On September 30 2021, a root certificate used by Let's Encrypt, *IdentTrust DST Root CA X3*, expired. That was planned, and of course another root certificate is now used instead by Let's Encrypt, so in most situations this is transparent.

How it affects XCP-ng

We discovered that the version of openssl that is present in our installer didn't handle the situation very well. The Let's Encrypt certificates can be validated following several different chains of trust - one of which is not valid anymore - and openssl 1.0.2 rejects the certificate if one of the chains contains an expired certificate, rather than trying another - valid - chain.

Fortunately, this doesn't affect yum, so hosts can still update from mirrors that have such a certificate.

But it does affect netinstalls, because the installer checks the mirrors before it would install the packages with yum, and that check fails. So users would get the following error message when our mirror manager would redirect to one such mirror: "A base installation repository was not found at that location".

How to solve it

We will release (or have released, at the time of reading) updated installer images that fix this issue (or may already have when you read this), but this won't be enough anyway: users may still use the previous installation images to do a network installation. So we need to also fix it at the mirror level.

Thankfully, there is a way to workaround the issue at the mirror level:

- Update certbot to at least 1.12.0
- Edit the configuration file at /etc/letsencrypt/renewal/
- Append "preferred_chain = ISRG Root X1" to the end of the [renewalparams] section.
- Run "certbot renew --force-renewal"

Unless there are valid concerns with this approach, we ask that every mirror that relies on Let's Encrypt for their certificates apply this workaround.

Send your application

Send an e-mail to mirrors (AT) xcp-ng [DOT] org, following the example below:

Subject: "Mirror application for mymirror.example.com".

Body:

```
Server name: mymirror.example.com
City: Paris
Country: France
GPS coordinates: 45.738600, 4.889860 (or "auto" to let GeoIP locate
your mirror)
Bandwidth: 1 Gbit/s
Source mirror: updates.xcp-ng.org
Hostname, IPv4 and IPv6: mymirror.example.com, 203.0.113.23,
2001:db8::1
Sync frequency: every 15 min
HTTP(S) URL: https://mymirror.example.com/
RSYNC URL: rsync://mymirror.example.com/xcp-ng/
Workaround for Let's Encrypt certificates: applied (or "not required"
if you're not using L.E.)
Other prerequisites from https://docs.xcp-ng.org/project/mirrors
checked: yes
Main contact: John Doe <john.doe@...>
```

"Source mirror" is the mirror you will sync from. At this stage, we suggest that the source always be `updates.xcp-ng.org` which is the main, most up to date, mirror. Rsync is restricted by default on this mirror, so we need you to provide either a hostname or an IP address access that will be allowed to sync from it (Hence the "Host or IP to authorize:..." line above). You may sync from another mirror if you prefer: choose one close to you that updates quickly and review your choice from time to time.

Feel free to ask any question in your application message.

🔒 Security

Context

When you download something from the internet, there is always a chance that the file you get has been corrupted by the transfer, or worse altered voluntarily by someone with malicious intentions.

So for important files, a good habit is to check the file against a cryptographic hash of the file. Usually the (quite weak) `MD5` hash function, or something stronger like `SHA256`.

But what if someone also modified the hash file?

This is a valid concern: maybe you are downloading from a mirror whose owner you don't trust fully, or maybe the server you are downloading from has been hacked. If someone managed to modify the file you want to download, they probably also modified the hash file too. That's where we need cryptographic signatures. Files can be signed with a private key, and you can use the corresponding public key to check the validity of the signature.

In XCP-ng

Since XCP-ng 7.6, we use GPG to sign:

- SHA256 sums for installation ISOs
- SHA256 sums for repository configuration files (`xcp-ng-X.Y.repo`, downloaded as part of the upgrade process if you choose to use `yum` to upgrade)
- Every RPM package. They are automatically checked by `rpm` on an installed XCP-ng.
- Repository metadata. They are automatically checked by `yum` on an installed XCP-ng.

Import the public key

Before you can use our public key to verify the authenticity of a downloaded file, you need to import it.

Download the key

```
wget https://xcp-ng.org/RPM-GPG-KEY-xcpng
```

A key is of no use if you can't trust it. So check its fingerprint.

```
gpg --quiet --with-fingerprint RPM-GPG-KEY-xcpng
```

Expected output:

```
pub  2048R/3FD3AC9E 2018-10-03 XCP-ng Key (XCP-ng Official Signing Key)
<security@xcp-ng.org>
      Key fingerprint = 34AC 2EB6 8248 FED6 D076  838A CD75 783A 3FD3
```

```
AC9E
sub  2048R/2EB0DF19 2018-10-03
```

If your version of gpg doesn't display fingerprints anymore, try instead:

```
gpg --import --import-options show-only RPM-GPG-KEY-xcpng
```

with expected output:

```
pub   rsa2048 2018-10-03 [SC]
      34AC2EB68248FED6D076838ACD75783A3FD3AC9E
uid                      XCP-ng Key (XCP-ng Official Signing Key)
<security@xcp-ng.org>
sub   rsa2048 2018-10-03 [E]
```

Note: if you fear that someone altered this document as part of a sophisticated attack, you can also check that the fingerprint stored in our GitHub repository is the same: https://github.com/xcp-ng/xcp-ng-release/blob/master/RPM-GPG-KEY-xcpng-info.txt (and in case you think someone might have altered the above link, check that the repository does actually belong to the XCP-ng project).

Now import the key with `gpg`:

```
gpg --import RPM-GPG-KEY-xcpng
```

Check a downloaded file

The signature for `somefile` is stored in `somefile.asc`. Download both, then:

```
gpg --verify somefile.asc somefile
```

Expected output:

```
# gpg: Signature made Wed 10 Oct 2018 12:50:06 PM CEST using RSA key ID
3FD3AC9E
# gpg: Good signature from "XCP-ng Key (XCP-ng Official Signing Key)
<security@xcp-ng.org>"
# gpg: WARNING: This key is not certified with a trusted signature!
# gpg:          There is no indication that the signature belongs to
```

```
the owner.
# Primary key fingerprint: 34AC 2EB6 8248 FED6 D076  838A CD75 783A
3FD3 AC9E
```

Only the date should change depending on the date of signature. **`Good Signature`** is what tells us that the verification is successful. The warning is also expected because the signing key itself is not signed and we've not setup a list of trusted sources on the computer, so there's no way for gpg to tell if the key is to be trusted or not.

Check an ISO image

In this example we will first check the authenticity of the SHA256SUMS file, then use it to check the integrity of the ISO image. We'll download `xcp-ng-7.6.0.iso`, `SHA256SUMS` and `SHA256SUMS.asc` from https://updates.xcp-ng.org/isos/7.6/ but you can transpose those steps to a newer ISO.

First: import the GPG key if not done already (see above).

Verify the authenticity of the `SHA256SUMS` file.

```
gpg --verify SHA256SUMS.asc SHA256SUMS
```

Now that we trust the authenticity of the `SHA256SUMS` file, let's check the integrity of the `.iso` file.

```
LC_ALL=C sha256sum -c SHA256SUMS 2>&1 | grep OK
```

It should display:

```
xcp-ng-7.6.0.iso: OK
```

Check a repository (`.repo`) file

This is exactly the same as for the `.iso` file above: we provided a `SHA256SUMS` and a `SHA256SUMS.asc` file.

First: import the GPG key if not done already. You can follow the steps above, but if you are on an XCP-ng 7.6 or newer, you can take a shortcut because the key file is already on your system: `gpg --import /etc/pki/rpm-gpg/RPM-GPG-KEY-xcpng`

Example with `xcp-ng-7.6.repo`:

```
wget https://updates.xcp-ng.org/7/xcp-ng-7.6.repo -O xcp-ng-7.6.repo
wget https://updates.xcp-ng.org/7/SHA256SUMS -O SHA256SUMS
wget https://updates.xcp-ng.org/7/SHA256SUMS.asc -O SHA256SUMS.asc
```

Verify the authenticity of the `SHA256SUMS` file.

```
gpg --verify SHA256SUMS.asc SHA256SUMS
```

Now that we trust the authenticity of the `SHA256SUMS` file, let's check the integrity of the `.repo` file.

```
LC_ALL=C sha256sum -c SHA256SUMS 2>&1 | grep OK
```

It should display:

```
xcp-ng-7.6.repo: OK
```

About repository metadata and RPMs

Those are signed with the same key, but you don't need to verify them manually: since XCP-ng 7.6, the key is installed directly on the system and used by `yum` and `rpm` each time you update or install packages from the repositories of XCP-ng.

This is defined for each subrepository (`xcp-ng-base`, `xcp-ng-updates`, `xcp-ng-extras`, etc.) by the following variables in `/etc/yum.repos.d/xcp-ng.repo`:

```
gpgcheck=1
repo_gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-xcpng
```

- `gpgcheck=1` means that the signature of every RPM that comes from this repository must be verified. Protects against malicious changes to the RPMs on the mirrors.
- `repo_gpgcheck=1` means that the signature of the repository's `repomd.xml` must be verified before using the repository. Protects against malicious changes to the

repository metadata, such as replacing a recent RPM with an older, security-flawed, signed RPM.

- `gpgkey` is the URL to the local file containing the GPG public key used for verifying the signatures.

Edit this page