Reproducibly building artifacts that contain embedded signatures

.ical

10-21, 11:40– (Europe/Paris), Main track (Gym)

Signatures are annoying when you are trying build software reproducibly, especially when deeply embedded in the output artifact. Let's look at how we can tackle this problem elegantly with Nix.

Reproducibility means that someone else can independently recreate exactly the same binary artifact. This is very useful for confidently knowing what source code a binary artifact was built from. People are analyzing artifacts with tools like `diffoscope` to exactly locate differences between two artifacts built using the same build instructions. For complex projects even when looking at an exact difference in the output that way, it is not always easy to find the cause of that difference.

In general using Nix to split the build instructions into smaller steps can help us make this process easier, because we can notice differences at the end of the intermediary step that introduced them, as long as we are `nix build --rebuild`ing the right build steps.

Even then signatures are still a problem, because we can never really reproduce a signed artifact without access to the signing key and even with access to the key not all popular signing schemes produce signatures deterministically. We either have to substitute in the expected signatures or keep track of those expected differences.

There is a nice pattern that we can use for always substituting the correct signatures with Nix, which makes it easy to verify embedded signatures as part of such an independent recreation process even for a large and complicated artifact. The same pattern also takes advantage of Nix's binary caches to automatically obtain all the required signatures, which are ideally the only thing we cannot reproduce.

What level of experience in Nix is the talk addressed to? –
Mid-level

Do you allow your talk to be recorded? – yes

Martin Schwaighofer

Martin Schwaighofer is a PhD student at JKU in Austria, interested in proving the link between a running system and it's source code.

Before his PhD and his journey into Nix he worked on Android apps with strict security requirements, where among other things he put a lot of effort into cleanly defining 'hard-to-manage' project dependencies so that they could be more easily consumed by his colleagues.

You can reach him via email, message him on Twitter or check what he's up to on GitHub. He's happy to get in touch with people that have the same interests and maybe even collaborate.