Security challenges for the Qubes build process

9-12 minutes

Security of the build and distribution process is something that is notoriously ignored by many open source projects (see below). In Qubes, however, we have been paying lots of attention to this problem since the very beginning. And the primary two reasons for this are:

1. We want to build (and distribute) non-backdoored software.

2. We don't want the build process itself to be able to compromise the developer's machine.

Why do we have Qubes Builder?

In order to address the above problems, we introduced Qubes Builder nearly 5 years ago. Unlike many other build systems – of which humankind has created plenty – Qubes Builder's primary focus is on the following tasks:

1. To perform verification of all the input sources, git repo commits, and other components (such as the stock RPMs and DEBs we also use), i.e. that they have proper digital signatures created by the *select* keys that *we* chose to trust.

2. Provide strong sandboxes for building the less trusted parts of the Qubes OS, such as the various templates, so that even if the (properly signed) sources or other components turn out to be malicious[*], the rest of the generated system, such as the Xen hypervisor and dom0, are not affected (nor is the developer's machine).

[*] Of course, one should understand that the mere fact that packages or sources are properly signed, even with key(s) we have decided to trust, doesn't guarantee that the code has not been backdoored. This could happen if one of the developers turned out to be malicious or was somehow coerced to introduce a backdoor, e.g. via some kind of a warrant or blackmail, or if their laptop were somehow compromised. We would like to defend against such potential situations.

Various Qubes Builder hardening approaches

We discuss below the challenges with the first task and how we solve them. The second problem mentioned above has been resolved with the use of Disposable VMs for template builds and is not the focal point of this article.

Additionally, starting from Qubes R3, we have been building all the official releases with all the networking traffic routed through a Qubes Tor gateway VM. The rationale for this has been to make it significantly more difficult to target the specific developer/machine building official Qubes packages/releases *even if* an adversary has somehow managed to get control of one of the signing keys for one of the components we've chosen to trust (e.g. the signing keys of Xen, Fedora, or Debian). It would be more difficult indeed, because such an adversary would normally be interested in providing the backdoored binaries only to the specific developer who builds Qubes, and not to everyone in the world who downloads e.g. Xen sources or Debian packages, as otherwise the adversary risks the attack being quickly detected.

On top of that, we have always built all the official Qubes packages and ISO images on our private computers, i.e. ones whose physical security we can reasonably guarantee. This is because we have always assumed all external infrastructure (aka "the cloud") to be untrusted. Indeed, because datacenter

personnel can always (stealthily) read/write the memory of systems or VMs running in their datacenters, allowing the build process to run there would always make it possible for external parties to either tamper with the build process and/or steal the release singing keys, if these were also uploaded, as many projects do. Intel SGX, a very new technology which has entered the market in recent months, might change this a bit in the future.

Ultimately, we would like to introduce a multiple-signature scheme, in which several developers (from different countries, social circles, etc.) can sign Qubes-produced binaries and ISOs. Then, an adversary would have to compromise *all* the build locations in order to get backdoored versions signed. For this to happen, we need to make the build process deterministic (i.e. reproducible). Yet, this task still seems to be years ahead of us. Ideally, we would also somehow combine this with Intel SGX, but this might be trickier than it sounds.

Source verification challenges

But let's go back to the first problem listed above: source integrity verification. We solve it by having Qubes Builder's main Makefile *automatically* verify digital signatures on any sources (i.e. git tags on the last commit) of each and every component it uses for the build (as configured via the `COMPONENTS` variable).

Yet, this straightforward approach is complicated by the fact that some components, such as the Xen hypervisor would like to download additional sources on which their own build targets depend. Sadly, this is often done in a very insecure way, as demonstrated by Xen's build scripts, which are wget-ing additional sources from the Internet over plaintext connections, without checking the signatures (however absurd that might sound in 2016…). The Fedora Project (on which we have based Qubes dom0 and whose tools we use for building the final ISO) is, sadly, another example of build security negligence: not only do Fedora tools not verify signatures on the downloaded packages (from which the ISO is to be built), their developers have also avoided merging our patches to fix that problem for nearly a year now!

So, in order to account for such misbehaving components, we've had to patch the corresponding build scripts so that they also verify the (insecurely downloaded) additional sources. We found it easiest to introduce an additional target for each of the components: `verify-sources` that is then called by the main Builder's Makefile after the `get-sources` target, which instructs the component to download any of the 3rd party sources its build process might need.

The sources verification bug

Sounds simple enough, right? However, very recently it was pointed out to us (and this is the reason for this post today), that our script for the `verify-sources` target in the Xen component contained a silly bug that resulted in the script *not* exiting upon failure to verify one of the sources and instead only printed an error message about this. The culprit was a misuse of brackets in the shell syntax.

Rather than fixing the brackets, however, we decided to implement a different patch. Now we verify each of the sources immediately after they get downloaded, and if they don't pass the verification, they are removed from the filesystem. This way, we aim to maintain the invariant that at any given time there are no unverified 3rd party files in the component's directory.

We would like to thank obotobo@openmailbox.org for pointing out the problem and working with us on finding the best solution.

We debated whether this bug justified a Qubes Security Bulletin and ultimately decided against it. The reason for this is that the application (or lack thereof) of the presented patch has no immediate effect on any Qubes user's security who doesn't build Qubes from sources themselves. At the same time, publishing this as an article allows for greater visibility of this problem in the future (can easily be linked from many places in the docs), which is important for prospective Qubes developers who should gain a good understanding of build security issues (and avoid doing nasty things like their colleagues from the

Xen project in the first place…).

Discussion of impact in practice

So, what are the chances that, due to this bug, the Xen Makefile managed to download a maliciously-modified source tarball and that we used it for building one of the official Qubes RPMs or ISOs?

Both Marek and I *believe* the chances are very slim. This is because, while building all of the official Qubes releases, both I (for the R1 and R2 releases), and Marek (for all R3 releases) *believe* we have always run `make get-sources` first, before the actual build. The rationale for this was rather prosaic: we wanted to ensure all the networking-intensive cloning/wget-ing was done before we started the actual multiple-hour-long build process. In that case, it would be very difficult for us not to spot the error message for the Xen component sources verification…

The word *believe* in the statement above is an important word, a word that we'd rather not need to use… Indeed, it would be very useful if we could have signed logs from the build processes of each official release. Such logs, in addition to displaying any such potential error messages, might also show hashes of all the input sources and packages, as well as those that were the outcome of the process (this in turn to make it more difficult for an adversary who has stolen Qubes signing key to perform targeted attacks on users).

Proposal for true append-only build process logging

We haven't been doing that so far, however, because there are a few things that should be done right for this to make actual sense. Specifically, we would like to account for the fact that if the build VM gets compromised somehow, then the logs could show that (and, of course, if the logs were to be copied from the build VM, they might be modified before we copy them to another VM and sign them there).

A simple solution for this would be to have a dedicated Qubes qrexec service (`qubes.AppendLog`) that would pipe logs to a separate 'log' VM in realtime. This way, it would not be possible for the source VM to tamper with any of the logs previously sent. Once the build process is complete, we could sign the logs in the log VM using Split-GPG and the Qubes Release Key, then upload them somewhere as additional evidence.

Implementing this should be rather straightforward but would involve some playing with the Qubes Builder scripts, testing, etc. We would be very happy to accept community patches for this so that it can be implemented before the upcoming Qubes 3.2 release.