back to posts

Attacks against GPG signed APT repositories

Feb 21, 2018 ▪ packagecloud

Tags: debian-guide

TL;DR

This blog post takes a closer look at some attacks against APT repositories outlined in the academic paper A Look In the Mirror: Attacks on Package Managers and how they apply to recent versions of APT software.

It is a common misconception that simply signing your packages and repository metadata with GPG is enough to create a secure APT repository. This is **false**. Many of the attacks outlined in the paper and this blog post are effective against GPG-signed APT repositories. GPG signing Debian packages themselves does nothing, as explained below.

The easiest way to prevent the attacks covered below is to **always serve your APT repository over TLS; no exceptions**.

In addition to the attacks mentioned in the paper, we'll also take a look at two other more recent security vulnerablities and discuss some mitigation techniques.

Building and maintaining secure APT repositories is difficult, time consuming, and must be handled with care. In some instances, some of these bugs or other issues may not be able to be easily dealt with as a repository server may not necessarily be able to enforce that clients use a particular version of APT when connecting to the repository server.

The easiest way to prevent as many security issues as possible with hosting APT repositories is to simply use TLS.

Create a secure APT repository in less than 10 seconds, free.
Sign up!

A quick note on APT, GPG signatures, and the current security model

Understanding the APT security model is important before diving into the various attacks. To understand the security model, you must first consider that there *two types* of GPG signatures that can be used. This point is a source of confusion for both APT repository hosts and users of APT.

The two types of GPG signatures that can be used are:

1. GPG signatures on the individual Debian packages themselves (which is not a useful activity, as explained next), and
2. GPG signatures on repository metadata.

Unless explicitly stated otherwise, the term GPG signature in APT software and documentation almost always refer to the latter case.

GPG signing Debian packages is not useful

GPG signing a Debian package does nothing because package signatures are not verified by default on any major distribution when packages are installed with `apt-get install`. See your `/etc/dpkg/dpkg.cfg` file for an explicit comment to this effect.

Here's mine (Ubuntu 16.04):

```
# Do not enable debsig-verify by default; since the distribution is not using
# embedded signatures, debsig-verify would reject all packages.
```

That said...

If you want users to verify GPG signatures of your packages *manually* (the only way they *could* verify them), you should provide explicit instructions on how to do this along with an XML policy file that `debsig-verify` can consume to verify the package's GPG signature.

It is possible there are Debian package providers doing this; we have yet to see anyone provide an XML policy document for verifying Debian package GPG signatures out in the wild.

GPG signing APT repository metadata

When APT software (such as `apt-get`, or `reprepro`) or folks offering APT repositories mention GPG signatures in their documentation they are typically referring to GPG signatures on *repository metadata*, not on packages themselves. Likewise, when you configure the `SignWith` option of `reprepro` (documented here), you are telling `reprepro` to sign your repository metadata with the specified GPG key; this does not sign any of the packages, though.

GPG signing repository metadata is intended to ensure that the repository metadata has not been modified after it has been generated. This is the typical method employed by APT repositories and is explained in the SecureAPT wiki page.

Please keep this in mind when thinking about the APT security model and the attacks presented below.

Overview of Attacks

The paper outlines a series of attacks against package managers, which we'll examine in more detail as they relate to APT.

The attacks outlined against various package managers throughout the paper are as follows:

1. Arbitrary package: an attacker can provide a malicious package in place of the package requested by the user.

2. Replay Attack: an attacker can replay older versions of correctly signed repository metadata causing a client to install a version of a package with a known vulnerablity.
3. Freeze Attack: an attacker can freeze the repository metadata a client sees at a particular moment in time preventing a client system from updating to a newer version of a package.
4. Extraneous Dependencies: an attacker rewrites the package metadata to insert dependencies on additional packages that have known vulnerabilities.
5. Endless Data: an attacker provides an endless stream of repository metadata or package data causing the client to fill its disk or memory and crash.
6. The paper also outlines a series of attacks done against the mirroring system for a variety of package managers. These attacks are very interesting, as the paper explains several ways in which a malicious mirror could be constructed. We won't be examining the mirror attacks in this blog post, but readers are encouraged to read the paper for more details and consider their usage of publicly available package repository mirrors.

We'll examine how each of these relate to APT repositories and how these attacks can be avoided or prevented. Additionally, we'll examine two other bugs that have surfaced more recently with very serious consequences:

1. Improper error handling in APT during GPG signature verification leads to arbitrary code execution.
2. `apt-key` silently fails to remove GPG keys requested by the user.

Create a secure APT repository in less than 10 seconds, free. Sign up!

Arbitrary package attack

It is possible for an attacker to cause a client of an APT repository to install an arbitrary package in three circumstances:

1. If the repository is served over plain-text HTTP with no GPG signatures, the repository metadata can be trivially replaced by an attacker who can perform a Man-in-the-middle (MitM) attack to cause the client to install an arbitrary package.
2. If the repository is served over plain-text HTTP with GPG signed repository metadata to a client running any APT version prior to 1.1 (e.g. Ubuntu 14.04 or earlier, Debian Jessie or earlier), or a more recent version of APT with `Acquire::AllowDowngradeToInsecureRepositories` set to `true` in `apt.conf` (the default is `false`, thankfully) an attacker who can perform a MitM can simply block requests for the GPG signed metadata and respond with hand-crafted unsigned metadata pointing at arbitrary packages.
3. If the repository is served over plain-text HTTP with GPG signed repository metadata to a client running any APT version with `Acquire::AllowInsecureRepositories` set to `true` (which is the default in Ubuntu 16.04) which has not previously been successfully authenticated against, an attacker who can perform a MitM can respond with hand-crafted unsigned metadata pointing at arbitrary packages.

In order to prevent this type of attack you should do the following:

1. Always use TLS for serving APT repositories. This will prevent an attacker from being able to perform a MitM.
2. Set the `Acquire::AllowInsecureRepositories` to `false` explicitly. This option will

prevent APT from using repositories that do not present GPG signed metadata, thereby preventing a downgrade attack. This option is present in APT 1.1 or newer (available on Ubuntu 16.04 or newer, and Debian Stretch or newer), but is defaulted to `true` for "backward compatibility" on several versions of APT. We strongly recommend setting this value explicitly to `false`.

3. Set the `Acquire::AllowDowngradeToInsecureRepositories` to `false`. This option is available on APT 1.1 or newer. Note that this option will only help if you've been able to connect to a repository and download the GPG signed metadata at least once already. This is better than nothing, but options 1 and 2 should be used, first.

If you are using a version of APT prior to 1.1, you should upgrade immediately. If your system does not provide a version of suitable version of APT, we provide a backported APT 1.2.10 for both Ubuntu Precise and Ubuntu Trusty, which includes these flags and additional recently added APT features for other APT bugs.

If you are using any version of APT, you should set `Acquire::AllowInsecureRepositories` to `false` in your `apt.conf`, because some versions of APT have this defaulted to `true`.

If you are hosting an APT repository for your users, customers, or clients: serve that repository over TLS, only.

Replay and Freeze attacks

An attacker who can perform a MitM attack on a client can replay valid APT metadata (along with the valid GPG signature for the metadata) to perform two types of attacks:

1. A replay attack where a client system is presented with APT repository metadata pointing to older versions of packages with known security vulnerabilities. APT will ask the user to confirm a downgrade, but automation tools (bash scripts, Chef, Puppet, etc) may swallow these warnings. It is typical to find scripts with the flag `--force-yes` being passed to APT.
2. A freeze attack where a client system is presented with APT repository metadata pointing to their current package version. This attack is presented for a period of time so that clients don't "see" new versions of packages when they are published in the future. This gives the attacker the opporunity to use exploits as they are discovered later.

In order to prevent these types of attacks there are several things you can do:

1. Always use TLS for serving APT repositories. This will prevent an attacker from being able to perform a MitM.
2. Repository hosts should ensure that the `Valid-Until` field is present in their generated APT metadata. If you use `reprepro` to generate APT metadata, you can enable this by setting the `ValidFor` option in your `conf/distributions` file. Of course, this will mean you need to regenerate your APT metadata before it expires so that you don't break any clients (possibly with a cron job, or some other mechanism). You'll need to choose a "reasonable" value for this, because replay attacks will still be possible until the expiration of the metadata.
3. APT clients should set the `Acquire::Max-ValidTime` option in their `apt.conf` to a reasonable value to guard against remote APT hosts with `Valid-Until` set to unreasonably large values. Additionally, APT clients should ensure the system clock is accurate (i.e., run NTP).

The `Valid-Until` field will only prevent replay attacks *after* the metadata has expired. The metadata can still be replayed until it expires, so this field does not protect against freeze attacks. Nevertheless, APT repository hosts who use this field should choose a short `Valid-Until` time. If you are using a tool like `reprepro`, this can be set by configuring the `ValidFor` option in `conf/distributions`. This also means APT repository hosts must regenerate their repository metadata regularly, even if no new packages are published, to prevent breaking clients.

Also note that the `Valid-Until` field relies on an accurate system clock on the client. APT clients should ensure their system clock is accurate by using a service like NTP.

Using TLS is the best option to solve this because:

1. You do not need to worry about regenerating your APT repository repeatedly to keep it up to date, even if no new packages are added.
2. You do not run the risk of a freeze attack during the metadata's validity window.
3. If your clients are independent entities, you may not be able to guarantee that their system clocks are accurate.

Create a secure APT repository in less than 10 seconds, free. Sign up!

Extraneous Dependencies

An attacker can provide hand crafted repository metadata that adds malicious dependencies to packages listed in the APT repository metadata in the following circumstances:

1. Plain text HTTP APT repositories with no GPG signatures.
2. Plain text HTTP APT repositories with GPG signatures that are served to clients running a version of APT that allows downgrade attacks or has `Acquire::AllowDowngradeToInsecureRepositories` set to `true`.
3. Plain text HTTP APT repositories with GPG signatures that are served to clients that do not allow downgrade attacks, but for which the client has not yet cached a copy of the signed metadata.

This attack is particularly painful because the original package the user requested is installed giving the user the impression that they have correctly installed their desired software.

The best way to fix this issue is to host your APT repository via TLS.

Alternatively, APT may be modified to verify that dependencies specified in the repository metadata are checked against the dependencies listed in the package itself when it is downloaded and reject any repositories with mismatches.

Endless Data

The APT client on Ubuntu Xenial (16.04.3) was vulnerable to this attack.

A malicious APT repository can provide repository metadata that is extremely large. APT clients will download this metadata and fill the disk.

Reproducing this is relatively straight forward:

1. Create an APT repository on the server.

2. Remove the Release files and replace the `Packages` files with extremely large files. We used zero-filled files generated with `dd`. We reproduced this bug by replacing the `Packages` file with a 10GB file that was zero filled.
3. Run `apt-get update` on the client.
4. Watch the disk fill up and the errors begin scrolling from APT when there is no space left on the disk.

Whether or not this type of attack would result in your system failing depends specifically on the operational parameters of the software being run on that system, however it is safe to assume that this could lead to a denial of service.

Fixing this might be a bit tricky as different folks may have different preferences on how much disk space used is too much, but either way patching APT to check the `Content-Length` HTTP header (if it exists) to determine if the metadata being requested will actually fit on the disk prior to attempting to download it is probably a good first step.

Without a custom patched APT client, the easiest way to guard against an attack like this would be to:

1. Ensure that `Acquire::AllowInsecureRepositories` is explicitly set to `false`. It is defaulted to `true` on Ubuntu 16.04.3. This would prevent downloading repository metadata if a Release file is missing.
2. Ensure you are using adequate system monitoring software to alert you when your disk reaches low disk space.
3. Perhaps consider writing a script or system monitoring to check the size of the APT cache files in `/var/lib/apt` and `/var/cache/apt` and alerting based on the size and free disk space available.

Create a secure APT repository in less than 10 seconds, free. Sign up!

Improper error handling in APT during GPG signature verification leads to arbitrary code execution.

On December 13, 2016 the debian-security-announce list announced a security vulnerability which leads to arbitrary code execution due to a bug in the GPG signature verification code in APT:

> An attacker able to man-in-the-middle HTTP requests to an apt repository that uses InRelease files (clearsigned Release files), can take advantage of this flaw to circumvent the signature of the InRelease file, leading to arbitrary code execution.

It is recommended that all APT users ensure they are running the latest version of APT which has patched this issue.

It is recommended that all APT repository hosts serve their APT repositories over TLS to prevent MitM attacks which can expose users to this and many other attacks, as described previously in this blog post.

`apt-key` silently fails to remove GPG keys requested by the user.

The `apt-key` utility is typically used to manage GPG keys installed on a Debian or Ubuntu system which are in turn used to verify repository metadata.

It is critical to be able to add and remove GPG keys to ensure that your system will be able to verify the integrity of the GPG signed APT repositories that APT connects and downloads metadata from, especially if no other security mechanisms (like TLS) are in place.

Unfortunately, some version of `apt-key` silently fail to remove GPG keys. The second comment explains the impact of this bug:

> What's not obvious from the original post, though, is that apt-key reports back that it deleted the key "OK" - but actually did not [...] This can result in users and applications alike meaning to revoke trust on an APT archive keyring, being told they succeeded in doing so, but actually failing.
>
> As such, from my perspective, this is a security bug.

This bug is currently marked as triaged, with one user reporting that they are still affected in 17.10. It remains to be seen if the next Ubuntu LTS, 18.04, will also be affected by this issue.

In the meantime, users are strongly encouraged to manually check their APT key rings in `/etc/apt/trusted.gpg` and `/etc/apt/trusted.gpg.d/` and verify that GPG keys they revoke are actually revoked.

Having a stale or untrustworthy GPG key in your APT keyring could be extremely detrimental, so it is worth the extra effort to manually verify GPG keys have been correctly removed until this bug is fixed.

Conclusion

Always serve and access APT repositories over TLS. No exceptions; there are simply too many potential attack vectors and bugs in the GPG verification system provided by APT to use only GPG with plain text HTTP.

Using GPG signatures on APT repositories in addition to serving the repository over TLS is, of course, fine and encouraged.

Private DEB repository

Private RPM repository

Private RubyGem server

Private PyPI server

Private Maven repository

HOWTOs

NPM/NodeJS HOWTO

Maven HOWTO

Java
HOWTO

Debian HOWTO

RPM HOWTO

RubyGem HOWTO

Python HOWTO

Linux HOWTO

Guides

Maven Guide

Debian Guide

RPM Guide

RubyGem Guide

Python Guide

Linux Guide

Docs

General Docs

API Docs

Command Line Interface

Community

Blog

Slack

Status

Contact

Legal

Terms of Service

Privacy Policy