Package Verification with dpkg: Implementation

John Goerzen <jgoerzen@progeny.com>

Version 5; January 4, 2001

1 Introduction

As part of a continuing effort to improve the security of users of Debian and Debian-based systems, one step that is important to take is to allow the users to verify for themselves that .deb packages that they have are from an authentic source. This mechanism is intended to prevent various types of attacks. Some are:

- Preventing people from setting up a fake Debian archive, containing potentially trojaned packages, and masquerading as a real Debian archive. This attack can be thwarted by checking an origin signature, checking archive signatures, or checking maintainer signatures.
- Preventing altered .deb packages from being installed undetected. Since all signatures are applied to the contents, any signature that is checked will protect against this attack.
- Preventing people from separating or inserting forged data or metadata into a package. Because all signatures are applied to both the control and the data chunks together, separating them will not be possible without causing a signature failure.
- Helping to prevent old packages from being presented as new. By checking the signature date, policy files can be set to reject packages that contain old signatures. Sites might re-sign their packages periodically with an "archive" or a "timeliness" signature that could be used specifically for this purpose.
- Allowing verification to ensure that a given package was part of a particular archive site, distribution, made by a particular developer, etc. Signatures could be applied for various stages of development – approval by QA managers, release managers, etc. The multiple signature per .deb architecture outlined in this proposal allows this.
- Allowing verification of an isolated .deb, even in the absence of an Internet connection or access to an archive. By including the signatures inside the .deb itself, it is not necessary to download signed index files to verify a signature.

1.1 Unaddressed Problems

The approach set forth in this document leaves one hole that must be solved by other means. That problem is the task of insuring the integrity of the keys, policy files, and programs themselves used to carry out the verification. One potential solution is to modify apt-get to support https and use trusted third-party (e.g., Thawte) to shore up this problem.

The mechanism used to ensure old packages are not presented as new might be resource-intensive on the server in some situations. One potential avenue to explore there might be to provide signed Packages files on the server.

1.2 Rationale

Some thought has gone into the design of this particular solution. For benefit of future discussions, here is some information on how this mechanism was chosen and reasons for it. Here are the main alternatives that have been considered:

GPG To implement this, additional support would be required from dinstall, apt, genchanges, and various other utilities. The additional complexity of doing so and the possibility of separating the signature from the .deb makes this approach undesirable.

GPG format. However, this would change the basic format of the .deb and render it unusable without a current dpkg and GPG installed on the system. Since it breaks backward compatibility in such a severe way, this approach is also undesirable.

Sig-per-chunk A sig-per-chunk system would place separate sigs in the ar file for the data and the control chunks. Unfortunately, this would allow someone to mix and match data and control data in making trojan .debs, which can also be a security threat.

Sig-per-repository A sig-per-repository system would place an md5sum of a .deb into the Packages files and sign those. However, this omits fine-grained control and prevents verification of a .deb in absence of the Packages file from the Internet. It also requires solutions to problems of determining which archive a package came from.

The policy files (see section **??**) are designed to be simple, straightforward, and easy to both write and parse. By keeping them as simple as possible, the potential for both human authoring errors and parsing errors in software can be minimized.

2 Modifications to .deb

Currently, a .deb consists of an ar archive containing control.tar.gz and a data.tar.gz files. These files represent package meta-data and package contents, respectively.

Each signature will be applied to the control.tar.gz and data.tar.gz files concatenated together in that order. The files should not be decompressed before applying the signature.

Each signature file will be stored in the ar archive and have a name of this form:

_gpg*type*

Therefore, an origin signature would be named _gpgorigin. The "type" string must not exceed 10 characters in length nor be less than 1 character.

Therefore, an origin signature in GnuPG format would be named _sigorigin.gpg. At this time, all signatures must be in GnuPG format.

3 Signatures and Types

Each .deb file that is signed must have at least an origin signature. This signature should be unique per entity. For instance, Debian, Helix, and Progeny would each provide an origin signature. The origin signature is used as a key into the policy database such that the proper policy description can be loaded to describe the file.

Beyond the origin signature, the signature types that are present or required are defined by the site policy. Examples of other signatures might be a maintainer signature from the person that built the package, a signature from quality assurance certifying that the package has passed QA tests, a signature from a release manager certifying that a package was part of a given distribution release, and a signature from an archive maintainer certifying that the package went into an archive.

4 A Runthrough

To make sure the entire process is entirely clear, here is what will occur when a package is processed:

1. The Key ID of the origin signature will be obtained. This will be used to determine the directory in which to look for policy files and keyrings as specified in section 5.

2. The checker will iterate over each policy file in that directory in an arbitrary order, doing the following:

   (a) Compare the Origin: line in the policy file (see section **??**) with the Key ID of the origin signature. If there is a difference, immediately die because of suspected corrupted policy files.

   (b) Evaluate the selection clauses, if any, in the policy file. If there are any that do not pass, skip to the next policy file. If all of them pass, or there are no selection clauses, adopt the current policy file as authoritative for the package.

(c) If no appropriate policy files are found, reject the package as having a bad (unverifyable) signature.

3. Given the appropriate policy file, the checker will iterate over every verification clause in that file. If all of them pass, the package is considered to pass the test. If any of them fail, the package is considered to have failed the test and processing immediately exits. No other policy files will ever be tested after this point. If the policy file contains no verification clauses, it will be rejected as invalid.

It is possible that other package verifiers that use the same .deb architecture but different verification procedures may be introduced in the future. This section is not intended to imply a prohibition against such.

5 File Locations

The following locations are defined for files that comprise this system:

- /etc/debsigs/policy/*Origin-ID*/* contains one or more policy files that are to be applied for .debs whose Origin key was signed by the Key ID as listed in the path. The policies are tried in a non-deterministic order until one matches the selection criteria. Once one matches the selection criteria, the success or failure of the entire check is entirely contingent upon the success or failure of the verification clauses. If no policy file matches the selection criteria, then the entire operation is determined to fail.

- /usr/share/debsigs/keyrings/*Origin-ID*/* contains one or more keyrings that are used by the policy files for the given origin. They may be symlinks to keyrings located elsewhere if desired. This is used as the directory for the keyrings in the RequiredSig and OptionalSig lines unless an absolute path is specified there.

The policy files are located in /etc because sites may opt to alter the criteria for installing files on their site. The keyrings are located in /usr/share because they should not be altered locally. Because of the multi-file and multi-directory nature, it is easy for organizations to provide packages providing information for verification of certain things. For instance, Debian could provide a debian-debsigs-potato package which contains a policy file and a potato release file. It could depend on debian-debsigs-main package, which contains the keyring for the origin field, a symlink to the main Debian keyring for the maintainers field, and a dependency on debian-keyring.

6 Related Documents

References

[1] de Winter, Brenno. *Gnu Privacy Guard Mini Howto* at http://www.dewinter.com/gnupg_howto/english/.

[2] Free Software Foundation. *The GNU Privacy Handbook* at http://www.gnupg.org/gph/en/manual.html.

[3] Free Software Foundation. *gpg manpage* available on local systems with gpg or at http://www.gnupg.org/gpgman.html.

[4] Braakman, Dorman, et al. *Debian Packaging Manual* at http://www.debian.org/doc/packaging-manuals/packaging.html/.

[5] Quinlan, Daniel. *Filesystem Hierarchy Standard* at http://www.pathname.com/fhs/.