Weeding out security bugs in Debian

Javier Fernández-Sanguino Peña

May 18, 2006

1 How do security bugs affect the Debian project

The Debian project asserts that it provides a high-quality distribution (Debian GNU/Linux) with a release cycle that is not forced upon by marketing requirements and, consequently, makes it possible to provide a distribution without important (i.e. release critical) defects.

However, once a release is done, any security bug affecting a software package that is part of the release has many direct consequences:

- our users' systems are immediately in danger of being compromised due to the security bugs (this will depend on the nature of the bug itself, and whether it's a local or remote exploitable bug)
- our security team needs to deal with the security bug in order to provide a new fixed software package backporting, or writing themselves, a patch fixing the security bugs
- when the fix is developed our buildd infrastructure needs to handle the fix and generate new packages in short time
- when an advisory is sent, after a new package version is available with the fix in our security servers, our security support infrastructure (bandwidth and services) has to cope with hundreds of users downloading the new version of the package to install the upgrade
- even if the security bug is fixed, there is always the possibility that the fix or the changes in the package introduce new bugs that will affect our users (even though they may not be security related)

If any of these steps fail and, consequently, the "window of exposure" (time it takes from a security vulnerability to be known to a patch be available by us) increases then this impacts negatively in the project, new sites will pick this up and it will become bad publicity.

Security bugs have a negative impact even if the our patching process works out flawlessly: we are able to produce patches in time for all our supported architectures (or even before the vulnerability is publicly known) and there are no hiccups with any of our infrastructure. When doing a review of the number of security bugs found for a given release, reviewers might find that the release process has not been adequate if the bugs found **after** the release is too high. Indeed, our release process was designed partly to find (and fix) these kind of bugs, if there are too many advisories published after a release then that might be an indication that there is a flaw in our release process.

There is also the issue of quantity, regardless of the previous issues, an increasing number of security bugs require an increasing number of resources from the Debian project. These resources increase: CPU (in different architectures to drive the security buildds), bandwidth (for the download of the patches), and, most important, human (the people that have to develop the patch, test it and write the advisory).

2 Security issues in the Debian distribution

The Debian Security Team has issued (since 2001 and up to April the 5th 2006) 1047 advisories for 1387 distinct vulnerabilities. Of these, over 65% have been related to remote vulnerabilities. This is not necessarily the **real** distribution of vulnerabilies of the different releases the project, it is the number of vulnerabilities that the project has issued advisories for.

This is the list of classes of security bugs found in Debian packages[1] as well as the percentage of vulnerabilities fixed in issued advisories:

buffer overflows the input being received by a system, be it human or machine generated, causes the system to exceed an assumed boundary. This might be considered a subset of improper data handling, but the large number of applications and the consequences of this bug (code execution) justify it being considered a different class of bug (almost 27% of security vulnerabilities);

improper data input handling the input being received by a system is not properly checked such that a vulnerability is present that can be exploited by a certain input sequence. This issue leads to many type of different attacks, such as cross-site scripting in web applications, or SQL injection (almost 25% of security vulnerabilities);

design errors when there does not exists errors in the implementation or configuration of a system, but the initial design causes a vulnerability to exist (18,7%);

boundary condition error the input being received by a system, be it human or machine generated, causes the system to exceed an assumed boundary. It is also a subset of input validation (7%);

exceptional condition handling handling (or mishandling) of the exception by the system that enables a vulnerability (6,5%);

access validation error the access control mechanism is faulty (4,7%);

race conditions the non-atomicity of a security check causes the existence of a vulnerability (2,6%);

[1]It is based on the published DSAs crossed with the information available in the National Vulnerability Database `http://nvd.nist.gov/` (NVD, formerly ICAT) based on the CVE name of vulnerabilities.

configuration error user controllable settings in a system are set such that the system is vulnerable (2,4%);

environmental error: the environment in which a system is installed somehow causes the system to be vulnerable (0,9%).

All these bugs are, in themselves, defects in the software itself. An application that fails to validate input[2] coming from untrusted users[3] might introduce a security vulnerability which can range from a buffer overflow remotely exploitable in a server daemon to a SQL injection error in a web-driven application.

The following is the number of advisories (and vulnerabilities) for the different distributions Debian has released[4]:

- 197 advisories for 256 vulnerabilities were published for Debian 2.2 (*potato*) which was in security maintenance for 2.79 years. There are 59 million lines of source code in this release;
- 690 advisories for 1070 vulnerabilities have been published for Debian 3.0 (*woody*) which has been in security maintenance for 3.7 years. There are 105 million lines of source code in this release;
- 271 advisories 570 vulnerabilities have been published for Debian 3.1 (*sarge*) in less than a year. This release has 216 million lines of source code.

Nobody will be surprised when told that the number of security vulnerabilities (and, consequently, advisories) published for an operating system is very dependant on the amount of software it includes, more software means more bugs. A recent analysis by Coverity[5], a company that provides a closed-source source code audit software, shows an average of 0.3 defects per thousand lines of code for some of the most popular and used FLOSS projects. Not all of these defects might be *exploitable* security bugs, but the more the distribution grows[6] the more security bugs it will hold.

It is important for Debian developers to know and understand the different types of vulnerabilities as well as to know what they could have done to prevent a programming bug to become a security issue. This includes: designing servers so that they properly implement privilege separation instead of running as root, avoiding the use of setuid or setgid binaries and providing good installation defaults such as not starting up a service if it is not properly configured or limiting access to an application to only the server it is installed on.

[2]For more information see the "Validate All Input" section of the David Wheeler's Secure Programming for Linux and Unix HOWTO `http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/input.html`.

[3]In these case they can be either remote users, for daemons, or local users for setuid/setgid applications.

[4]I have also include the size of the distribution in millions lines of source code based on the Libre software engineering (Libresoft) research group from the Universidad Rey Juan Carlos, as detailed in Debian Counting `http://libresoft.dat.escet.urjc.es/debian-counting/`.

[5]For more information see Automating and Accelerating Open Source Quality `http://scan.coverity.com/`, an analysis of thirty open source projects including the Linux kernel, gcc, FreeBSD, NetBSD, Apache, Samba, Perl, Firefox and GNOME. LWN coverage (with interesting discussion) is at http://lwn.net/Articles/174426/

[6]And based on Libresoft's data it is currently doubling its size every two years!

3 Work of the Debian security audit team

The Debian Security Audit Team `http://www.debian.org/security/audit/` started working in 2004 to focus work on auditing Debian packages for security issues. It has been directly responsible of 82 Debian Security Advisories and has opened up 122 security-related bugs in the BTS (up to march 2006).

The Audit Team is composed of loosely coordinated group of people. Although they use a public mailing list, more of the audit work is "hidden" and is not even discussed on list until an advisory is published. Currently, the different members of the Audit Team focus on one specific type of bug and work their way through the package sources in order to find instances of that type of security bug.

One of the goals of the Audit Team is to have security bugs fixed in the distribution before they are really an issue (i.e. before the affected package versions are released).

Occasionally, members of the team also review security bugs and advisories from other distributions and make sure that the Debian package that provides the same software is fixed in Debian too. At times, this overlaps with the work already done by the Stable and Testing Security Teams but it often means that there are more "eyes" looking for (known) security bugs that might be present in the software we distribution.

These are some of the lessons learned by the team:

- many developers are not aware of the consequences of some security bugs and need to be shown that a security bug is of higher severity;
- even though some bugs have been found and reported, there are many more *security* bugs present waiting to be removed. This specially applies to software that is not too popular (consequently, not many people are looking for bugs in it) or security type of bugs that are not being often reviewed;
- there is too much software in the distribution and auditing resources are scarce;
- the available free software tools for source code review are insufficient for the task at hand;
- it takes quite some time to fix security bugs. Specially security bugs which are not highly critical (such as temporary file vulnerabilities). This is related to the limited resources of the Debian Security team but it also happens because of maintainers being unresponsive.

4 How can a developer improve security in the Debian OS

When you are packaging software for other users you should make a best effort to ensure that the installation of the software, or its use, does not introduce security risks to either the system it is installed on or its users.

You should make your best to review the source code of the package and detect issues that might introduce security bugs before the software is released with the distribution. The programming bugs which lead to security bugs typically include: buffer overflows `http://en.wikipedia.org/wiki/Buffer_overflow`, format string overflows, heap overflows, integer overflows (in C/C++ programs), and temporary symlink race conditions `http://en.wikipedia.org/wiki/Symlink_race`(very common in Shell scripts).

Some of these issues might not be easy to spot unless you are an expert in the programming language the program uses, but some security problems are easy to detect and fix. For example, finding temporary race conditions in source code can easily be done by just running *`grep -r "/tmp/" .`* in the source code and replace hard coded filenames using temporary directories to calls to either *mktemp* or *tempfile* in Shell scripts, or File::Temp in Perl scripts, and *tmpfile* in C/C++ code. You can also use source code audit tools [7] to assist to the security code review phase.

When packaging software make sure that:

- It is not alpha or beta software, if it is, prevent it from going into *testing* (by introducing an RC bug for it). If it's not ready for release, don't let it be released.
- The software runs with the minimum privileges it needs. That is:

1. the package does not install binaries setuid or setgid[8];
2. if the package provides a service, the daemons installed should run as a low privileged user, not as root.

- Programmed periodic tasks (i.e., *cronjobs*) installed in the system do not run as root or, if they do, do not implement complex tasks.
- The default configuration is sane and limits exposure. Don't think that everybody will install the software in a development enviroment and needs all the bells and whistles the program might provide.

If you are packaging software that has to run with root privileges or introduces tasks that run as root, make really sure it has been audited for security bugs upstream. If you are not sure about the software you are packaging, or need help, you can contact the Debian Security Audit team and ask for a review. In the case of setuid/setgid binaries, you must follow the Debian policy section on permissions and owners `http://www.debian.org/doc/debian-policy/ch-files.html#s10.9`.

Once your software has been released, make sure that you track security bugs affecting your packages either through upstream mailing lists or through security mailing lists. If a security bug is detected that affects your package you must follow the Handling security-related bugs `http://www.debian.org/doc/manuals/developers-reference/ch-pkgs.en.html#s-bug-security` guidelines in the Developer's reference. Basically this boils down to contacting the security team to let them know, and help produce (and test) patches for the software versions released.

Finally, invest time in reading about security bugs and how to prevent them. David Wheeler's Secure Programming for Linux and Unix HOWTO `http://`

[7]More information available at http://www.debian.org/security/audit/tools

[8]*Lintian* will warn of setuid, setgid binaries in the package

www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html should be a must read, it is an online book freely available packed full of valuable content. For developers that package web-based applications, the OWASP Guide http://www.owasp.org/documentation/guide.html is also a must read. Other recommended reading would be Secure Coding: Principles |& Practices http://www.securecoding.org , by Mark G. Graff and Kenneth R.Van Wyk (ISBN 0596002424)

5 Conclusion

The constant growth of the distribution makes it inevitable that a large number of unfound security issues are present in each release. Security bugs drain important resources from the project but developers have it in their own hands to improve the situation by making sure they provide releasable software and they prevent software that might not be releasable (unaudited, alpha or beta software) from getting into the distribution.on.

Security safeguards might be introduced in the distribution, such as stack overflow prevention measures (as implemented in OpenBSD or Adamantix) or Mandatory Access Control mechanisms (such as SElinux). But these safeguards will only protect our users against specific set of attacks, users cannot (and should not) rely on them to protect their systems against every possible instance of security bugs.

Also, unfortunately, due to the current status of automatic source code audit tools it is not possible, for the moment, to design or provide something akin to lintian.debian.org to warn Debian developers (and users) of possible security bugs in Debian packages. We are currently missing metrics to evaluate the quality (security-wise) of Debian packages (and the software they include) to both detect and make decisions about software distributed within Debian.

That makes developer awareness on information security issues something even more important if we want to be successful in providing a high-quality universal operating system.