Weeding out security bugs in Debian

How to improve security for our users

http://people.debian.org/~ jfs/debconf6/security/

Javier Fernández-Sanguino Peña

`jfs@debian.org`

Weeding out security bugs

- Main Goal: Provide information to DDs on how to avoid/fix security issues in their packages.
- How?
    - Describe status of security in our OS (risks?)
    - Describe the work of the different security-related teams.
    - Show some tools to audit source code.
    - Present lessons from the audit team.
    - Discuss recommendations for improvement.

Impact of security bugs in the OS

What happens when a serious security issue is found in our OS?

- Our users are at risk.
- DDs and security teams have to work fast to provide a patch.
- Our security mirror servers/bandwidth are stressed.
- Some systems might get compromised.
- Our public image is affected.

Resources required to deal with these bugs increase with time.

First comments on security bugs

- All software has bugs.
- Security bugs are of varying severity (CVVS):
  - remote vs. local
  - DoS vs. code execution
- Security bug types vary with time (investigators shift focus).

Note: Coverity analysis: 0.3 per 100k LOC in stable (and audited) projects.

Status of security issues in Debian

- The size of the distribution keeps increasing in every release, so do the bugs in it.
- We are not much better than we were 3 years ago (see my Debconf-3 talk)
  - But there are now more teams than the Security Team.
- Let’s see some liesˆ W data... (download file *data.tgz*)

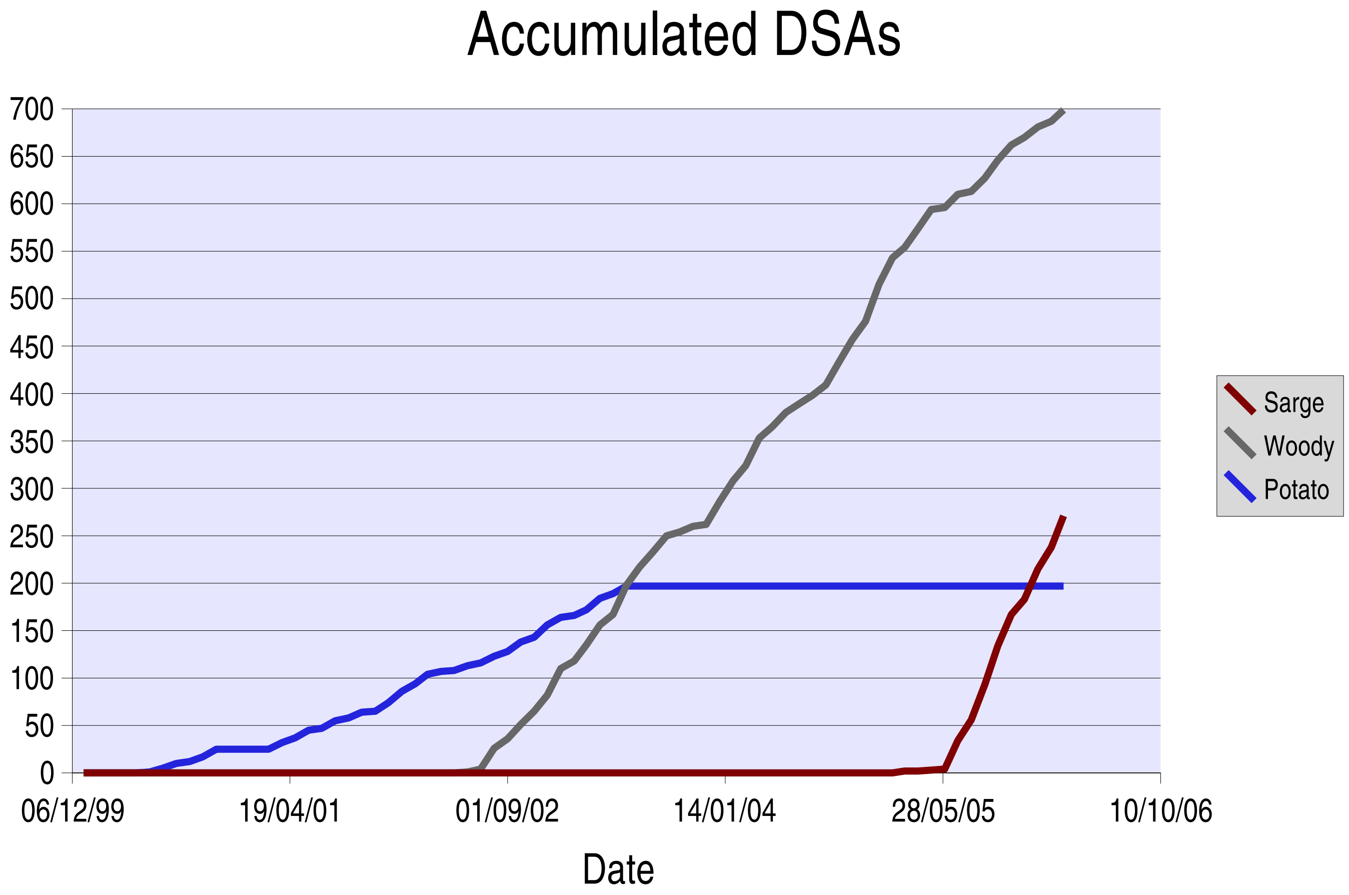
Security bugs in Debian: some lies

Total advisories published for Debian: 1231 advisories

- Potato: 197 DSAs (256) - 59 MLOC, maintenance 2.79 yr
- Woody: 699 DSAs (1070) - 105 MLOC, maintenance 3.7 yr
- Sarge: 271 DSAs (570) - 216 MLOC

Based on CVE Names: 1047 advisories since 2001 for 1387 distinct vulnerabilities.

Security in Debian: Fancy graph take 1



In sarge, most of them in packages of section *net* (˜ 16%) or *web* (˜ 23%)

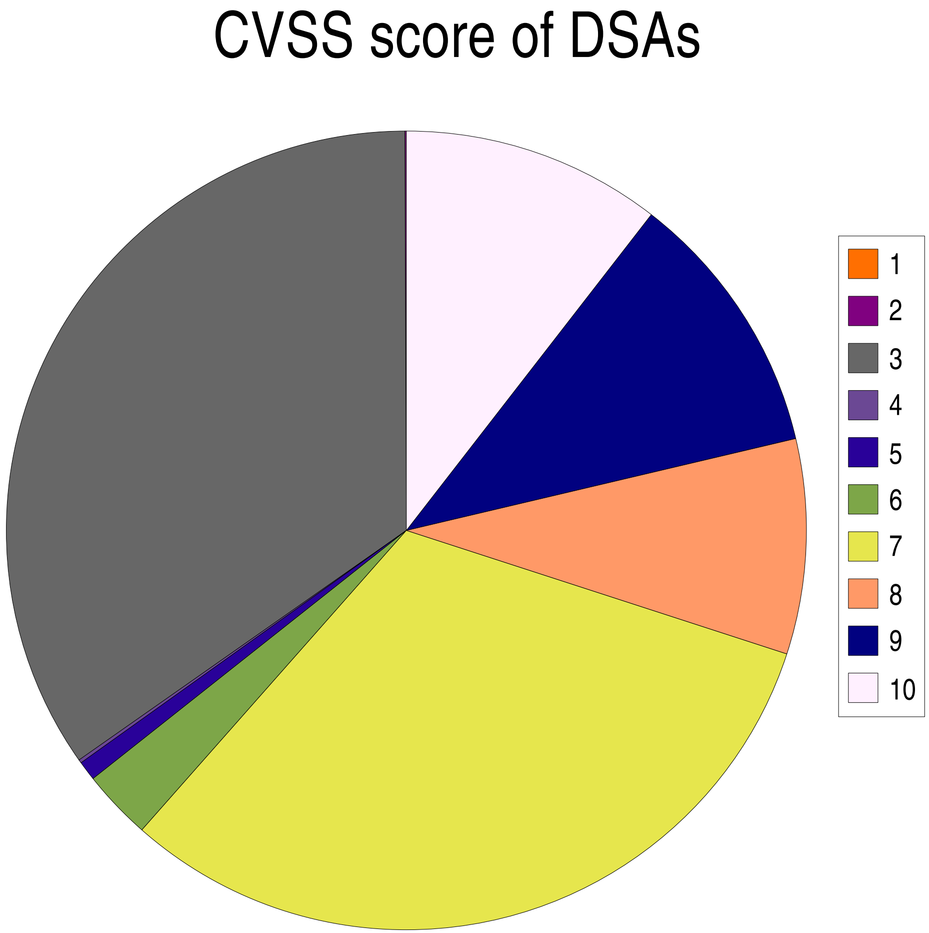
Security bugs in Debian: more damn lies

Different types:

- Buffer overflows: 26,9%
- Improper data handling: 26,3%
- Design issues: 18,2%
- Exceptional condition handling: 7,4%
- Boundary condition: 5,7%
- Access validation: 5,6%
- Unclassified: 3,9%
- Race condition: 2,8%

Approx. 65% remotely exploitable.
Note: Data of 1369 distinct CVE names from vulnerabilities from September 1998 to March 2006.

Security in Debian: Fancy graph take 2



Median CVSS value is 7:
See http://nvd.nist.gov/ and http://www.first.org/cvss/

Hands-on: hello-insecure

Download *hello-sample.tgz* from either ftp://homer.mexico.debconf.org/share/jfs/ or http://people.debian.org/˜ jfs/debconf6/security/samples/:

- hello-insecure-2.1.1.debian.diff: changes to the hello package
- hello-daemon-insecure_2.1.1-5_i386.deb: the binary package. **WARNING**: installing this opens up a remote root hole in 1025, is your firewall up?
- server-spotted.c: Security bugs in the server daemon commented in.

How many (security) bugs can you spot?

Teams handling security bugs

There are three different teams handling security bugs in Debian:

- Security Team: handles security bugs (aka patches) in *stable*.
- Security Testing Team: handles security bugs in *testing*.
- Security Audit Team: looks for security bugs.

The Debian Security Team

- Made up of 4-6 members.
- Relates with other teams through vendor-sec and CERT.
- Reviews public-disclosure bugs (do they affect us?).
- Produces and tests security patches.
- Writes security advisories.
- Publish patches through a specific buildd network.
- (sometimes) Follow up on compromise of Debian systems.

The Debian Security Testing Team

- Made up of 6 (?) members.
- Works with public information (CVE names)
- Reviews status of security fix propagation from sid to testing.
- Issue DTSAs.

Security support for testing started September 2005, integrated in main archive in May 2006.

The Debian Security Audit Team

- Made up of 4 members.
- Some members started auditing in year 2003, group formed year 2004.
- Priorise packages.
- Focused on certain things:
  - bugs in setuid/setgid applications (games)
  - misuse of sprintf/fscanf/syslog/...
  - temporary file race conditions
- Developed some tools developed to do automatic code review.
- As a result: 81 DSAs (13 %), 121 security (non-DSA) bugs

Debian Security Audit Team: tools

Some tools used by the audit team (http://www.debian.org/security/audit/tools):

- RATS: C tool to review C/C++/Perl/PHP/Python, works with an XML database to detect problematic functions.
- Flawfinder: Python tool to analyse C/C++, looks at functions and how they are *used*
- pscan: not general purpose, just format string overflows.
- Audit::Source (http://hinterhof.net/˜ max/audit-perl): Run all of these at the same time (and colour the code)
- Other tools: grep, bfbtester, other black box tools...

Hands-on: multiple-bugs.c

Download *multiple-bugs.tgz* from either ftp://homer.mexico.debconf.org/share/jfs/ or http://people.debian.org/˜ jfs/debconf6/security/samples/:

- Review *multiple-bugs-nocomments.c*: how many security bugs can you spot?
- Run RATS, Flawfinder and pscan in it: how many did they spot?
- Review comments in *multiple-bugs.c*
- Compare source with *multiple-bugs-fixed.c*
- Run RATS, Flawfinder and pscan in *multiple-bugs-fixed.c*: how many did they spot?

Audit Team: Lessons learned

Some lessons learned by the security audit team:

- Many developers are not aware of common security flaws: incorrect design of software (setuid/setgid, root daemons...), buffer overflows, sanitise user input..
- Many more security bugs waiting to be fixed (specially in software which is not popular)
- Too much software to audit, no easy way to do source code review (no centralized repo).
- FLOSS source code reviewing tools useful but need improvements.
- Fixing security bugs takes a lot of time.

Audit Team: Lessons learned DSA-656

Some lessons learned DSA-656 (see *DSA-656.tgz*), arbitrary file overwrite in vdr (network music daemon):

- Having a server disabled per default is not a security measure, users will start it up anyway.
- Maintainers don't heep upstream's comments, from the INSTALL file: don't run this as root!
- It's difficult to do a redesign in a DSA (see #287899), thus stable users do not get all the benefits of an audit.

Hands-on: DSA-893

Pick up *DSA-893.tgz* from either ftp://homer.mexico.debconf.org/share/jfs/ or http://people.debian.org/˜ jfs/debconf6/security/samples/:

- acidbase_CVE-2005-3325.bad.diff: upstream's fix
- acidbase.CVE-2005-3325.diff: my fix for DSA-893 (actual package changes in acidlab.CVE-2005-3325.pack.sarge.diff)
- acidlab-0.9.6b20-12to13.diff: changes between version in sid/sarge (checkout changes to acidlab.apache.conf)

Audit Team: Lessons learned DSA-893

Some lessons learned DSA-893, SQL injection in acidlab:

- Upstream doesn’t always know how to fix security bugs
- Security bugs of some packages might affect other packages with common codebase (BASE -> ACID)
- It’s better to restrict access to sensitive web interfaces by default (security bug in default install -> security bug enabled by admin)
- Fixes for SQL injection bugs and XSS bugs in PHP apps are similar: review user’s input!
- A security fix is not always 100% thorough ("time to fix" pressure)

Audit Team: More lessons

Some more lessons learned:

- DSA-647, Temporary filename race condition in MySQL: even popular software has obvious security bugs.
- DSA-334, 354, 356, 368, 369...> vulnerability in application setGID games = compromise of users running any games in the system. Also #291613 (setGID games writing in user’s dirs without dropping privs). Are global hiscores worth it?
- #334616, yiff-server running as root can "play" any file: why does a sound daemon need root privs.
- #329365, mailleds can be used to kill any system process: watch your umasks!

Audit Team: Even more lessons

- #291389, tcl: No tempfile/mktemp/mkstemp implementation in toolkit language - some bugs do not help implement secure code.
- #255033, securecgi design flaws: writting security code is not simple, a *secure* in the name does not make it so.
- #291376, cdrtools: Unsafe recommendation (and implementation) of debugging in rscsi - some maintainers sit on security bugs (*lack of time*?). Please do credit where credit is due.
- #291635, format string bug in man2html: some unaudited sofware ends up being used in CGI gateways.

Audit Team: Bored of lessons?

- #298114, nvi init script can be used for mischiveous purposes: bugs can remain undetected for a very long time and not all security fixes reach stable.
- #323386, kismet, CAN-2005-2626 and CAN-2005-2627 present in sarge and etch: lazy maintainers do not want to track bugs in stable.
- #289560 vim, Race conditions and symlink attacks in vim scripts: why provide obsolete/unsupported stuff? rewritting security patches sometimes introduce new mistakes, why take patches from Ubuntu when we have our own?

Weeding out security bugs: How can I help?

- Learn how to spot security bugs, review upstream’s code.
- QA your *own* code for security bugs.
- Learn how to program with security in mind and do proper design of your packages.
- Review applications you maintain:
  - Track security bugs upstream.
  - Follow guidelines for handling security bugs.
- Join one of the security teams.

Prevent/minimize security bugs

- Do not package or include alpha/beta/unsupported software (or prevent it into getting into *stable*.
- Use low-privilege users for daemons and cron tasks (see #337086)
- Avoid *setgid* and *setuid* software (review the Policy)
- Default *safe* configurations
- Review applications you maintain:
  - Security track record?
  - Responsiveness of upstream for security bugs?

Conclusions

- Some new technologies (SElinux, GCC 4.1 SPP, PaX, exec-shield, RSBAC..) might enhance protection of our users, but they might not cover *all* possible security bugs.
- Removal of security bug is a group work: make sure you've done your part.
- Try to code in a secure way (learn how if you don't know) and review your upstream's code (help them learn too)
- Use tools to help you in review (but don't trust them fully)
- Learn from past mistakes (even other's).

Thanks

Thanks!

For more information

Recommended reading thingies:

- Debian specific:
  - Debian Security Team FAQ: http://www.debian.org/security/faq
  - Debian Securing Manual: http://www.debian.org/doc/manuals/securing-debian-howto/
  - Debian Security Audit Team: http://www.debian.org/security/audit/
- David Wheeler's *Secure Programming for Linux and Unix HOWTO*: http://www.dwheeler.com/secure-programs/
- Fortify's *Taxonomoy of Coding Errors*: http://vulncat.fortifysoftware.com/

For more information

- Courses:
  - Dan Bernstein's *UNIX Security Holes Course*: http://cr.yp.to/2004-494.html
  - University of Purdue's *Secure Programming Educational Material*: http://www.cerias.purdue.edu/secprog
- Books:
  - *Practical Unix Security*: Simon Garfinkel and Gene Spafford. ISBN 0-596-00323-4
  - *Secure Coding, Principles and Practices*: Mark Graff and Kenneth R.van Wyk. ISBN: 0-596-00242-4

Answers: hello-insecure

Hello-insecure security bugs (knowingly introduced):

- Design problems: running as root, startup a debug daemon listening in all interfaces
- Maintainer postinst bug: create stuff in /tmp
- Maintainer compile bugs: why -DDEBUG?
- Server code bugs: format string, buffer overflow, log in /tmp and DoS due to memory exhaust

Answers: multiple-bugs

Hello-insecure security bugs (knowingly introduced):

- BoF using getenv with sprintf
- Hardcoded path of logfile in /tmp
- fopen use with race condition
- Stack overflow due to gets
- Static bof due to fixed size buffer (sprintf)
- Format string overflow because of misuse of syslog
- Command injection due to misuse of system ()