Bugs

why and how to report + related tools overview

Red Hat
Ondřej Vašík

2009-09-11

Abstract

Basic overview of bugs (from command line point of view, no desktop involved), overview and basic usage of common tools useful to gather enough informations to fill perfect bug report.

Agenda

- 1) Bugs - summary of bug types
- 2) Why to report bugs?
- 3) How to report bugs?
- 4) Memory leak detection - valgrind
- 5) Gathering call traces - ltrace,strace
- 6) Profiling - oprofile
- 7) Tracebacks - gdb
- 8) Networking - tcpdump

Bug types (generally)

- a) String errors
- b) Crashes
- c) Unexpected behaviour
- d) Memory leaks
- e) Performance impact
- f) Security issues

String errors

- **What?**
  - typos
  - missing/incorrect documentation
  - bad/missing translation (just when translation is expected)
  - bad wording, grammar
- **Impact**
  - usually low/very low
- **Action**
  - usually very easy to fix - so don't hesitate to report it
  - do not expect update just for such small issues
  - fix in devel branch and inclusion in next regular update should be usually enough.

Crashes

- **What?**
  - segmentation faults
  - buffer overflows
  - ...
- **Impact**
  - very high if in common usecase, sometimes could be security
- **Action**
  - if you have the reproducer, should be always reported
  - with good reproducer and/or enough data for analysis (strace, backtrace with installed debuginfos - when possible) is likely it will get fixed soon

Unexpected behaviour

- **What?**
  - real "bugs", program behaves other way than documented/expected
- **Impact**
  - invalid data output, so impact could vary
- **Action**
  - first, read once more time manpage/FAQ/POSIX, many of these bug reports are false positives and expected behaviours
  - if you are sure that it's a bug, provide reproducer and report it

Memory leaks

- **What?**
  - process is leaking the memory
- **Impact**
  - bad for long running applications/deamons
  - otherwise the impact could be quite low, not worth of fix
- **Action**
  - use valgrind (described later) and send report about the leak with corresponding data
  - Some false positives are possible, so use valgrind only when you experience growth of memory consumption in time in some application/process or with increased amount of handled files/data

Performance impact

- **What?**
  - program works as expected, but in some cases it gets horribly slow or consumes too much resources
- **Impact**
  - waste of system time (and energy)
- **Action**
  - use profiler to analyze where is the culprit of the impact, report with reproducer
  - sometimes can't be easily fixed, but usually at least a workaround is found

Security issues

- **What?**
  - exploits of access rights/privileges, memory safety
  - race conditions, denial of service, code injection
- **Impact**
  - could be very high and needs immediate action
- **Action**
  - if you think you found a security issue (even if you are not 100% sure), report it ASAP as private (with security sensitive bug checkbox active) bugzilla ticket with keyword "Security"
  - DO NOT report/widespread the exploit via mailing list - security team will assign CVE number and severity and report it publically once the bug is fixed
  - `http://fedoraproject.org/wiki/Security/Bugs`

Why to report?

- maintainers generally have more packages to maintain and they have no time to test all the features of the application
- therefore they usually don't know about the issue until you report it.
- reporting means benefit for other users (and even you) in future
- once the bug is fixed, program is better

Don't be scared about bug reporting, it's not that hard to report the bugzilla ticket.

Where to report the bug?

- **Generally two ways:**
  - contact upstream (on their mailing list, bug tracker)
  - contact distribution vendor (e.g. Red Hat Bugzilla for Fedora, Product support for RHEL)
- Contacting upstream is better if you are sure that the issue still exists in latest upstream unmodified version - usually there are more active guys to fix the issue.
- Otherwise it's better to contact distribution vendor as the issue may be caused by distro-specific patches.

Reporting via RH Bugzilla in few steps (1/5)

- 1) Ensure yourself that the issue is not caused by proprietary drivers or installed things from RPM Fusion or 3rd parties (especially in the case of desktop bugs, issues with proprietary drivers are very likely to be closed without deeper investigation)
- 2) Check existing bugzillas to prevent duplicate report (try to use just several keywords as summaries are pretty generic) `https://bugzilla.redhat.com/query.cgi?format=advanced`
- 3) If you don't have bugzilla account, create new one - valid email address is required - as the email is visible to the public for registered users, it is better to use some secondary email

Reporting via RH Bugzilla in few steps (2/5)

- 4) Find propper product:
  - select your installed distribution from either Fedora or Red Hat section (EPEL is in Fedora section)
  - report only against supported versions (generally last two released versions of Fedora + devel branch)
- 5) Find propper component:
  - if the name of the failing/related binary is known, use combination **rpm -qf `which binaryname`** to get component name/version
  - otherwise it is good to search for similar error report to get proper component
  - if you experience SELinux AVC denial in common functionality, it will likely be incorrect context on some of your directories/files or problem in the selinux-policy component

Reporting via RH Bugzilla in few steps (3/5)

- 6) Adjust severity and priority, if you are sure about their values, otherwise keep defaults
- 7) Write short summary of your issue
- 8) Fill the description template
  - description:
    - describe your issue, more details are better
    - if you use different than C locale (check command ‘locale‘), try to reproduce the issue with C locale as well(LANG=C and reproducing command)
    - if not reproducible with C locales, mention your locales in description
  - version/release:
    - use **rpm -q <component(s)>**

redhat.

Reporting via RH Bugzilla in few steps (4/5)

- 8) Fill the description template (continue)
  - Steps to reproduce:
    - fill in some easy reproducer
    - (as easy as possible, try to reduce options and steps)
  - Actual results and Expected results:
    - use only if not 100% obvious from description
  - Attachments:
    - patches are always welcome (if not 100% obvious from description and easy to fix bug, not necessary for e.g. typos)
    - backtrace in the case of crash (more useful with debuginfo rpms installed)
    - valgrind reports in the case of memory-leaks
    - in some cases of unexpected behaviour it's good to attach strace/ltrace/tcpdump log (better with C locale, to reduce string translations)

Reporting via RH Bugzilla in few steps (5/5)

- 9) Wait for maintainer's reaction, ping him if there's no response within a few weeks (sometimes mail with bugzilla report gets lost/forgotten/catched by antispam, but pinging every day in case of low priority bug doesn't make sense), be prepared to give more information e.g. about your configuration

valgrind

- **very powerful suite for debugging/profiling, not just for checking memory leaks**
- Benefits:
  - easy-to-use tool for searching for memory leaks with detailed log output
  - locates exact place of allocating memory which is not freed
- Dangers:
  - sometimes false positives
- How?:
  - it is highly recommended to have debuginfo packages for libraries used by the binary you are checking (to get useful output)
  - highest impact have memory leaks in applications running for long time (e.g. daemons) or in applications working with a lot of data (long time run)
  - simple usage to gather a lot of information is: **valgrind -v --leak-check=full <binary> <arguments> 2>myvalgrind.log**

strace

- **monitoring utility showing system calls executed by program and signals/exitcodes it received**
- Usage:
  - revealing a place of hanging (e.g. waiting for I/O, timeout)
  - revealing the exit codes of syscalls called by program
- Benefits:
  - for experienced maintainer a lot of information about the program run, he could analyze where the problem occured and fix the issue
  - for common user – it's not needed to analyze the issue, just passes traces to maintainer in bug report
  - no need for debuginfos, a lot of informations gathered in one log file
- How?:
  - just run program "under" strace: **strace ./whatever 2>mystrace**
  - or attach strace to existing process: **strace -p PID 2>mystrace**

ltrace

- **monitoring utility (similar to ltrace) showing library calls made by program**
- Usage:
  - quick verification which functions are called and which exit codes they return
- Benefits:
  - quicker than usage of debugger
  - works even for executable without debuginfo
- How?:
  - same as in the case of strace
  - just run program "under" ltrace:
    ltrace ./whatever 2>myltrace
  - or attach ltrace to existing process:
    ltrace -p PID 2>myltrace

oprofile

- **system-wide profiler for Linux systems, capable of profiling all running code at low overhead**
- Usage:
  - gathering informations about most-expensive systemcalls
- Benefits:
  - useful to find calls most affecting performance
- How?:
  - 1) install kernel debuginfo package:
    yum install kernel-debuginfo
  - 2) set correct environment:
    opcontrol --separate=kernel --vmlinux=/usr/lib/debug/lib/modules/<kernel-version>
  - 3) run the test binary and check the results with:
    opreport -l /path/to/mybinary
  - 4) pack the results for later usage on another machine:
    oparchive -o /var/lib/oprofile/samples/current/

gdb (1/3)

- **lets you to see what is going on 'inside' another program while it executes**
- Usage:
  - debugging of compiled binary(breakpoints, watches)
  - analyzing core dumps
  - attaching to already running process
- Benefits:
  - very powerful tool
- Negatives:
  - requires at least basic knowledge of the binary source codes
  - more helpful on binary with disabled optimalization
  - up2date sources/debuginfo required for successfull debugging
- How:
  - just run binary inside gdb: **gdb <my binary>**
    and type **run <parameters for my binary>**

gdb (2/3)

- How?: (continue)
  - or attach to existing process with **gdb -p <PID>**
  - or read coredump with **gdb -c <coredump file>**
  - set breakpoints by **break <file>:<line>/<function>** (feel free to write just abbreviation b, gdb could handle it)
  - you could set also conditionals for breakpoint: **condition 1 foo=1**
    - program will stop only when value of foo is 1 when breakpoint 1 is reached
  - set watchpoints by **watch/rwatch/awatch <expression>** - stops when expression is written/read/both by program
  - use **info breakpoints** to see active breakpoints

gdb (3/3)

- How?: (continue)
  - type **next** for next line in the current stack level,
    type **step** for next line (dives into functions),
    type **continue** to continue the execution of program,
    type **print** <**variable**> to check actual variable/structure values
  - type **backtrace** to check call-stack of functions,
    type **up/down** to walk up/down through stack frames
  - type **quit** to exit gdb

- **you could install ddd or nemiver as gui frontends to gdb**

tcpdump/wireshark

- **common packet analyzers for monitoring network activity**
- tcpdump running from the command line, wireshark with graphical frontend
- Usage:
  - Monitoring network activity to/from specific machine and/or port
- Benefits:
  - Useful to gather information where is the issue in network communication (e.g. packet blocked by firewall)
- How?:
  - simple example for capturing tcp packets that flow over eth1, port 8081:
    tcpdump -w dumpfile -i eth1 tcp port 8081
  - displaying dumpfile in "human" readable form by:
    tcpdump -nnr dumpfile
  - more details in manpage

Summary

There is a lot of other topics or details related to that presentation e.g. systemtap (covered by Petr Műller) for kernel hooks or automated reporting covered by ABRT presentation. There was no time to cover details, so if you are interested, check related documentation.

The end.

Thanks for listening.