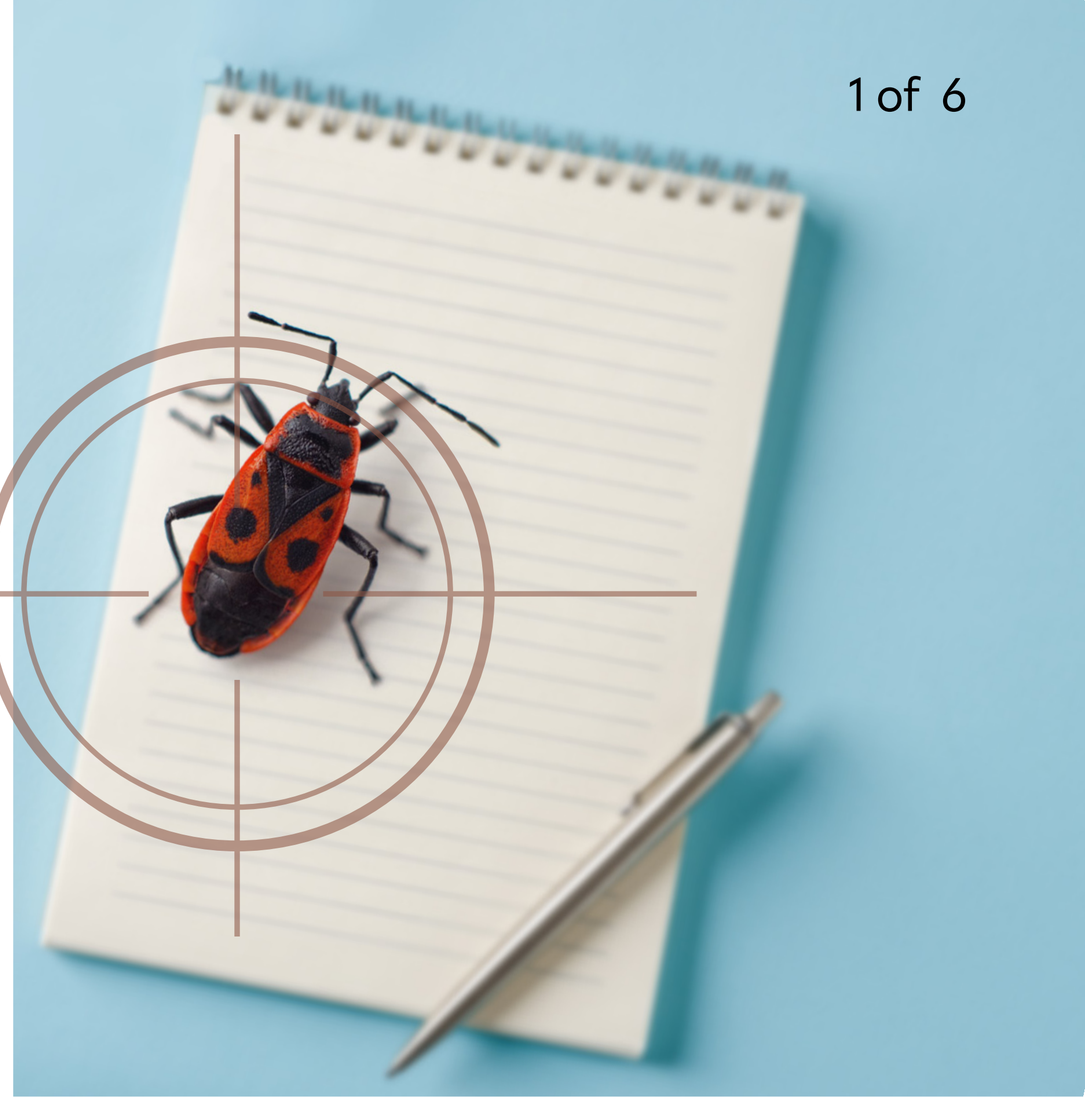
Writing Effective Bug Reports

BY TOM JONES



A few years ago, our cherished former FreeBSD Journal editorial board member, Kristof Provost, wrote a great piece on the ideal way to write a bug report. Sadly, Michael W Lucas uses up our per-issue allotment of sarcasm early in the year, so to avoid going into debt, it was decided that I reframe Kristof's original post in a way that would avert ironic bankruptcy.

Every system maintainer has their own preferences for how to receive reports of bugs and requests for improvements. As Kristof is so core to pf development in FreeBSD, he was the ideal person to write a piece for the journal.

In response to being asked to reframe his original piece, Kristof responded:

"I cannot build on perfection."

So, it is up to me to rephrase Kirstof's original words and provide you with some starting advice to help you quickly sort out your favourite and most troublesome bugs. You can find his blog post at https://www.sigsegv.be/blog/2014/M.

Bugs

Often, software projects feel like they are made out of bugs; developers will tell you everything is made from shoe polish and duct tape. The issues that cause the most trouble are often the hardest to pin down. "Late at night, when we have 12 concurrent requests hitting our geodetic load balancer" isn't an uncommon start for a complex bug. Sometimes there is a dreaded "after 1000 hours..." lead into an issue description.

Whatever form your bug takes, from logical impossibility to a typo, I am going to show you a framework you can follow to get things fixed.

Thankfully, not all bugs are like this. FreeBSD will panic in many circumstances, and for a user, that can be a bad day, but for a developer, a panic message can be a handy clue about where things are going wrong. I'd take a panic over a silent data corruption issue any day.

Some bugs are purely cosmetic, fields aren't displayed as well as they may be, or documentation is unclear (yes, we consider that a bug!).

Whatever form your bug takes, from logical impossibility to a typo, I am going to show you a framework you can follow to get things fixed.

The bug life cycle

Before we go into the best way to write up a bug, let's discuss what happens afterward, as I think this provides the context for why you need to add as much information as you can to initial reports.

In FreeBSD, most bugs propagate either via mailing lists or the bug tracker (bugs.freebsd.org). Developers regularly read the mailing lists, but the best way to get attention for a bug is a ticket on the bug tracker.

A newly created bug is marked as "new" and is in what we call the new bug state. At this point, it has been submitted by a user, but no relevant project areas have been alerted to its existence.

The Bugmeister team goes through new submissions and reassigns them from the new bug state, making a best attempt at setting the category of the bug correctly. Developers can choose to be notified about bugs in specific categories. Relevant project mailing lists will also receive notifications about new bugs. Additionally, a summary email containing "bugs of interest to this group" is sent periodically.

Bugs may be assigned to a particular developer by Bugmeister or by other FreeBSD project members.

Bugs may be assigned to a particular developer by Bugmeister or by other FreeBSD project members.

A developer may at some point decide that they want to "take" a bug, becoming the owner of the issue. This is typically the point at which the developer chooses to work on the bug or pursue analysis.

There may be back and forth on the issue as the developer asks for more information, but if everything goes well, eventually the developer will create a code review (on reviews.freebsd.org). The code review may be referenced in the bug, but it might be that the developer is only seeking knowledgeable feedback from certain other developers familiar with the subsystem.

If testing is required to determine if the bug is resolved, it will likely be highlighted.

Finally, once code review and testing are completed, the bug will be committed to FreeBSD; typically, the commit message will contain a reference to the bug, and an automated update will be added. Sometimes it takes many fixes to address the issue underlying a bug.

Once the fix(es) are committed, you (the user) can use the software again bug-free, assuming you are running a CURRENT snapshot. Many fixes are merged from CURRENT back to stable branches; we call this a merge from current (MFC).

If the bug is MFC'd, then the fix will be available on a stable branch and will be available in the next point release.

This process, from reported issue to fix, can involve a lot of steps; some problems can be resolved in 20 minutes, but others might take months or years. I have seen bugs closed out as fixed after more than a decade of debugging.

The time it takes to resolve an issue depends significantly on the recency of the problem. If you catch the bug soon after the commit lands, there is a good chance it will be fixed today. Other bugs take longer to show up, or only appear in environments running FreeBSD releases.

The time it takes to fix those bugs largely depends on the quality of the bug report and the ongoing discussions about the issue. So let's look at what a good bug report contains.

Where to submit a bug report

The FreeBSD project encourages users to share their experiences with the software we develop through multiple channels for reporting issues. In order of preference, notifications about new or interesting bugs would come as one of the following:

- an easy-to-apply change with clear tests provided as a new code review on reviews.freebsd.org
- a hacked-up solution to an issue on reviews.freebsd.org
- a new bug on bugs.freebsd.org
- an email (or other communication) directly to the developer
- a complaint on the forums with speculation about the cause
- IRC, matrix, Discord, or another chat venue
- ~~an angry post on social media that FreeBSD is the worst~~

The first two options assume quite a high level of technical competence. If this is too much for you, don't worry; we are still happy to receive bug reports through other methods.

The FreeBSD project encourages users to share their experiences with the software we develop through multiple channels for reporting issues.

We don't expect everyone to do advanced software development, just that every reporter provide as much relevant detail as they can at each step.

For a typo, most users can immediately suggest a remediation, but creating a documentation change requires using developer tools, and we don't ask that of everyone. If you can use these tools, creating a review is preferred over creating a new bug, as the final change will likely require a review either way.

More complex bugs might be addressable immediately; some complex bugs have simple answers. However, other bugs are challenging to diagnose, and the best information we have about them for a long time is sideways glances as something seemingly unrelated fails.

If you have a proposed fix of any quality, the most important thing to do is to share your suggestion with the public. Work down the list until you find the correct place.

Writing a good bug report

There is a lot of information out there on what should go into a good bug report; the breadth of software in FreeBSD makes any single template pretty tricky to use.

There are some good rules of thumb for what you should include:

- Information about your setup.
- What you expect to happen.
- What actually happens.
- What is the difference between what should happen and what happens?
- This isn't always obvious.
- Events leading up to the bug being triggered.
- Temporal factors help developers debug issues. If the bug occurs immediately after a reboot or only after 5 days, different strategies are required.

Where to get information and what to provide

As the system runs, it generates a lot of information that can be helpful for debugging. You may be asked for logs. If you are, you should try to include things like:

- dmesg: the output of the `dmesg` command
- all of the text of a kernel panic message, including the stack trace
- the output of `pciconf -lv` if it is a hardware issue
- syslog output
- tool output and error messages

Err on the side of providing more information than less; if a developer has to ask for more information, that might add a week before they can look at the issue in detail.

Information about your setup

You need to strike a balance between describing the Anycast load balancing setup for your Fortune 500 company and simply showing a single firewall rule. Typically, reports provide less information about their environment than they could; the combination of tools and network configurations helps developers understand bugs.

So what constitutes your setup?

We need to know the version(s) of FreeBSD you are using and how many there are. The fact that the bug occurs between 14 and 15 hosts is more relevant than knowing it is just 14.

We need to know about customizations and detours from the norm; you should include them if you built FreeBSD yourself or if you are using packages. We support both, but if you aren't running the release engineering version, then that is something to consider.

If you aren't running FreeBSD, but a downstream such as pfSense or HardenedBSD, you *need* to include that. It doesn't mean FreeBSD developers won't help you, but omitting this fact will likely annoy someone.

This is relevant because downstreams build FreeBSD with different flags, packages, and ship their own tuning. All of this can contribute to a bug existing on one platform and not another.

We really need your setup and not an idealised version or an example from a tutorial.

What you expect to happen

Another essential part of reporting an issue is describing what you expect to happen. Some software can't do what you want, either now (due to feature development) or ever (due to NOT ACTUALLY BEING A BUG!).

`ls` won't copy files or play videos, but pfctl should be able to clear rule state counters. Please tell us what you expect the output and effects to be so we can double-check all of your assertions.

What actually happens?

Provide output from the tool, describe the actual effects you can measure, and highlight any issues.

What is the difference between what should have happened and what happened?

It will be evident to you that the tool is broken. You are submitting a bug, but as a developer triaging an issue, additional help in spotting what is going wrong is essential.

Many bugs aren't easy to see in a textual report, and performance issues can be challenging to relate to (and to diagnose and fix).

Be very clear about what has happened and what isn't correct.

Events leading up to the bug being triggered

For many bugs, the events leading up to the bug being triggered are important. Can't do X after Y is an excellent bug report — it implies a path to developer gold (see below).

"`ls` reports incorrect file sizes once disk is full" tells us a lot of information that "`ls` reports incorrect file sizes" doesn't.

It might help to review dmesg before and after an instance of a bug; kernel subsystems will report errors and resource exhaustion there. In dmesg, we can see that the interface is going up and down constantly.

How should you provide information?

The information you include should ideally be textual. If there are error messages, you should include all of them; some software will always print error messages or print odd diagnostics when it performs operations. In a bug report, you should try to include this information.

If you are experiencing panic, you might struggle to copy and paste text from somewhere. In that case, taking a photo with your phone is a valid strategy; however, you should try to ensure a balance between file sizes and image clarity. Ideally, you would take a photo and transcribe the text as accurately as possible.

Your photo host is likely to go away in the future, but text strings in the FreeBSD bug tracker live for decades.

For many bugs, the events leading up to the bug being triggered are important.

Writing an excellent bug report

An excellent bug report will carry most of the above information, but it will also include developer gold: A reproduction case.

For a simple issue, this is something like a command invocation, but for some problems, you might have to write shell scripts that work together. Reproducers for network bugs can be very difficult to write, but thanks to vnet jails, it is actually possible to do this with just a shell script (that is, in fact, how the firewall test suites work).

A reproducer aims to reduce the reproduction of the problem to its barest components.

Proactively help to resolve the issue

Once you have submitted the bug, that might be enough to get a developer's interest to fix it. If you highlight a regression or new bug within 20 minutes of a commit, you are likely to get the developer's attention right away. Most developers watch mailing lists and the src-commits list after making changes to catch exactly these reports.

If your bug languishes or only gets assigned to a team after a while, don't panic; that is a pretty usual experience. You can proactively contact developers who work in similar areas or on similar ports. Just be polite; you are using a volunteer's time.

Once a developer has picked up your bug, they will likely ask you questions. If it has been a long time, that question might be, "Does this still happen on a snapshot?"

Otherwise, they may ask for more information or for you to test patches. If the bug is only on specific hardware, we can frequently have a fix sitting in the bug tracker, but without

someone to confirm it solves the bug, or at least doesn't cause a regression, the patch will languish.

Some bugs are just difficult to figure out. If a developer is working with you, feel free to suggest your theories for what might be the trigger. Some bugs sound like ghost stories until we find an eventual fix.

Good luck!

Writing and submitting a bug report is a lot of work; it is easy to feel frustrated that your effort seems ignored or not followed up on. FreeBSD is a volunteer project, and developers work on things as they can based on their interests. We have a great set of developers who will hunt down obscure and complex issues, but they need your help and cooperation to gather enough information to reproduce and test changes.

If you follow the advice here, you should have a much better experience getting exposure for your bugs and getting them fixed.

TOM JONES is a FreeBSD committer interested in keeping the network stack fast.