Robustness principle

From Wikipedia, the free encyclopedia

In computing, the **robustness principle** is a general design guideline for software:

> *Be conservative in what you do, be liberal in what you accept from others* (often reworded as "Be conservative in what you send, be liberal in what you accept").

The principle is also known as **Postel's law**, after internet pioneer Jon Postel, who wrote in an early specification of the Transmission Control Protocol that:[1]

> TCP implementations should follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others.

In other words, code that sends commands or data to other machines (or to other programs on the same machine) should conform completely to the specifications, but code that receives input should accept non-conformant input as long as the meaning is clear.

Among programmers, to produce compatible functions, the principle is popularized in the form be contravariant in the input type and covariant in the output type.

Contents

- 1 Interpretation
- 2 Criticism
- 3 References
- 4 External links

Interpretation

RFC 1122 (1989) expanded on Postel's principle by recommending that programmers "assume that the network is filled with malevolent entities that will send in packets designed to have the worst possible effect".[2] Protocols should allow for the addition of new codes for existing fields in future versions of protocols by accepting messages with unknown codes (possibly logging them). Programmers should avoid sending messages with "legal but obscure protocol features" that might expose deficiencies in receivers, and design their code "not just to survive other misbehaving hosts, but also to cooperate to limit the amount of disruption such hosts can cause to the shared communication facility".[3]

Criticism

In RFC 3117, Marshall Rose characterized several deployment problems when applying Postel's principle in the design of a new application protocol.[4] For example, a defective implementation that sends non-conforming messages might be used only with implementations that tolerate those deviations from the specification until, possibly several years later, it is connected with a less tolerant application that rejects its messages. In such a situation, identifying the problem is often difficult, and deploying a solution can be costly. Rose therefore recommended "explicit consistency checks in a protocol ... even if they impose implementation overhead".

References

1. Postel, Jon, ed. (January 1980). *Transmission Control Protocol* (https://tools.ietf.org/html/rfc761). IETF. RFC 761. https://tools.ietf.org/html/rfc761. Retrieved June 9, 2014.
2. Braden, R., ed. (October 1989). *Requirements for Internet Hosts: Communication Layers* (https://tools.ietf.org/html/rfc1122). IETF. RFC 1122. https://tools.ietf.org/html/rfc1122. Retrieved June 9, 2014.
3. Wilde, Erik (2012) [1999]. *Wilde's WWW: Technical Foundations of the World Wide Web*. Springer-Verlag. p. 26. doi:10.1007/978-3-642-95855-7. ISBN 978-3-642-95855-7.
4. Rose, M. (November 2001). *On the Design of Application Protocols* (https://tools.ietf.org/html/rfc3117). IETF. RFC 3117. https://tools.ietf.org/html/rfc3117. Retrieved June 9, 2014.

External links

- History of the principle (http://ironick.typepad.com/ironick/2005/05/my_history_of_t.html)
- Internet Protocol (http://www.postel.org/ien/txt/ien111.txt#line=1483), page 22; J. Postel, IEN 111, August 1979.
- The Harmful Consequences of Postel's Maxim (https://tools.ietf.org/html/draft-thomson-postel-was-wrong-00)

Categories: Internet Standards