

Security Vulnerability Severity Classification

Thomas Biege <thomas@suse.de>

27th January 2005

Abstract

This paper will describe a method of classifying the severity of security bugs in software for Unix-like systems.

On the following pages I will propose a metric with weights to describe the impact of vulnerabilities on a scala S with n elements ($S = \{n \in \mathbb{N} \mid 0 \leq n \leq 10\}$)¹ to provide an objective rating system.

This classification scheme should serve as reference for the *SuSE Security-Team* for releasing security announcements. Hopefully this mechanism will be adopted by other vendors to have a vendor-independent rating system.

Such a vendor-independent rating scheme will help customers, other vendors, and security companies/organisations to judge more precisely about the level of impact of a released security update.

Reasons for an unified Classification Modell

For every security vulnerability found several Linux vendors release updates for their customers. These updates are generally announced by sending an advisory to mailing lists and computer security organisations (like *CERT*). Most advisories include a severity level for this particular update but unfortunately most vendors use different schemes for classifying the same vulnerability and only a few of them are based on a comprehensive system.

This situation makes it hard for other groups to compare the different advisories and to make the correct assumption about the level of impact as well as ranking different vulnerabilities.

The security-teams themselves can gain advantage of a publically communicated ranking scheme by using this ranking to argue about update delays. A low classified update can take more time than a moderate or critical classified update. Some teams may even go further and assign update availability durations to the different ranking values.

¹The upper limit of 10 is for historical reasons and unfortunately avoids a finer mapping.

A positive side-effect of an independent scheme would be a more compact picture of the Linux community to the public.

The Classification System

The classification scheme is based on a general metric for defining a base value of severity and an additional weight system to make the severity rating more precise.

In our rating scheme I have to make some assumptions and rules that are based on experience during my work in the *SuSE Security-Team* to neutralize uncertainty and to keep it simple:

- every bug is exploitable
- only the privileges gained by exploiting a bug matters ²
- interaction between different bugs can not be taken into account
- a bug in a library is handled as bug in the most privileged application shipped linked with the affected library code

General Metric

The basic classification is made by the following situation:

- remote vs. local
- root user access vs. non-root user access
- root group access vs. non-root group access

	user level access	group level access	
remote	8	7	root
remote	6	5	non-root
local	4	3	root
local	2	1	non-root

²gaining root access through a hole in the kernel does not differ from root access gained through a hole in a *setuid* application

Weight System

Weights are vendor-dependent and have to be suffixed to the general rating value. For example this happens when a vendor has a daemon activated by default, or has set a *s-bit* for owner/group of an application, or uses different privileges for a service. Different weight-types are:

- system user (for non-root setup)
- system group (for non-root setup)
- human user (for non-root setup)
- human user group (for non-root setup)
- default package vs. extra package (installation selection)
- default active vs. default inactive ³
- user interaction needed ⁴
- chroot/jail environment
- denial-of-service condition
- arbitrary command execution
- temporary workaround (w/o shutting down the service) available

Type	Weight
system user	+1
system group	+1
human user	0
human user group	0
default package	+1
extra package	-1
default active	+1
default inactive	-1
user interaction needed	-1
chroot/jail environment	-2
denial-of-service condition	-1
arbitrary command execution	+1
temporary workaround available	-1

³including customized configuration as well as activated startup scripts

⁴Like opening an archive file or clicking on an image.

Note, by applying the weights to the base value we may become values smaller than 0 or bigger than 10. These values need to be corrected to the lower and upper boundary of our scala respectively.

Examples

The format pattern is as follows:

<basic value>(<non-root>, <package>, <activation>, <interaction>, ...
 ... <jail>, <impact>, <workaround>).

As example we take some recent advisories from the *SuSE Security-Team*:

Advisory	Actual Rating	New Rating
SUSE-SA:2004:020	6	$4(+1,+1,+1,-,-,+1,-,-) = 7$
SUSE-SA:2004:019	5	$6(+1,-1,-1,-2,+1,-,-,-) = 4$
SUSE-SA:2004:015	6	$8(-,-1,-1,-,-,+1,-,-) = 7$

Notation Mapping

To make this scheme more accepted and easier to parse for humans it is useful to use some *Fuzzy Logic* or threshold values to map the numeric values to a more abstract description (like „Critical“ or „Low“).

For simplicity we use a linear threshold modell to map our rating values to verbal terms.



Figure 1: Serverity Notation Mapping

Epilog

Thanks for suggestions and corrections:

- Roman Drahtmueller <draht@suse.de>
- Ludwig Nussel <lnussel@suse.de>
- Marcus Meissner <meissner@suse.de>