RZ 3111 (#93157) 03/08/99
Computer Science/Mathematics 7 pages

Research Report

VulDa: A Vulnerability Database

D. Alessandri and M. Dacier
{dal,dac}@zurich.ibm.com

IBM Research Division
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents and will be distributed outside of IBM up to one year after the date indicated at the top of this page. In view of transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and to specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

IBM Research Division
Almaden • Austin • Beijing • Haifa • T.J. Watson• Tokyo • Zurich

VulDa: A Vulnerability Database

D. Alessandri and M. Dacier

IBM Research Division, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract

As a follow-up to the vulnerability data sharing workshop held more than two years ago in Gaithersburg, this document offers an update on VulDa, the Vulnerability Database maintained at the IBM Zurich Research Laboratory. It explains how we populate the database, how we use it, and under which conditions we could be ready to share this information with interested partners.

1. Introduction

The Global Security Analysis Lab (GSAL) is the IBM Research center of competence in applied network security. It develops methodologies and tools for detection, prevention, analysis, and repair of hacker attacks. It provides support for IBM Consulting practices and IBM Emergency Response Services (ERS). To achieve those goals, the GSAL in Zurich has maintained a vulnerability database (VulDa) for more than 3 years now. The principles of Vulda were presented in June 1996 at the vulnerability data sharing workshop organized by NIST in Gaithersburg. Proposals had been made to offer access to VulDa to parties interested in contributing to it. Today, VulDa contains more than 3.5 Gbytes of compressed data and is used within IBM for two purposes.

The rest of this document is structured as follows: Section 2 offers a brief overview of the data contained in VulDa and how people can access it. Section 3 presents an example of what we call a vulnerability description file. Section 4 describes the two reasons why we keep maintaining VulDa. Section 5 concludes our thoughts regarding the sharing of this information.

2. Content

The information contained in VulDa can be divided into two categories: public information and private information. Public information comprises of all the data that can be found on the Internet regarding vulnerabilities. Private information concerns information not available on the Internet that we receive directly from various IBM sources. For obvious reasons, this paper addresses only public information.

Public information is obtained through various means: news groups, mailing lists, FTP mirroring, and web sites. VulDa contains archives of more than 30 news groups and 60 mailing lists. We regularly mirror more than 45 FTP sites and have copies of dozens of self-proclaimed “hacker” web sites.

This huge amount of data is collected via semiautomatic methods. The most relevant documents, e.g., exploit scripts, are studied and stored in the database according to our classification scheme.

Three search engines enable users to retrieve information efficiently in VulDa: (i) a keyword search engine, (ii) a text-mining search engine, and (iii) a vulnerability description file search engine.

I. Keyword search engine

A classical full-text search engine can be used to look for documents within VulDa. Regular expressions and Boolean queries are supported. Owing to the size of VulDa, this is not the most efficient way to retrieve information. Therefore, VulDa has been partitioned according to several criteria that enable the user to restrict his search to a subset of the database. For example, he can search for mail posted to a single mailing list. He can specify that he is interested only in finding exploit scripts, the description of which matches a given regular expression.

II. Text-mining search engine

The keyword search is an easy way to find information as long as the user has some idea about the parts of VulDa he should look into. If this is not the case, we offer him the possibility to look for clusters of related documents. Each cluster is automatically created

thanks to a text-mining tool. This means that similar documents, from a semantic point of view, are collected in one place.
Instead of retrieving isolated documents that match a given expression, the text-mining search engine returns clusters of documents, the definition of which matches the given expression. The definition of a cluster is a list of words automatically selected by the tool that globally characterizes the documents in the cluster.
With this technique, it is possible to retrieve documents that do not necessarily contain the given expression but are of interest to the user because they belong to a cluster that is characterized by the expression he is looking for. Alternatively, it can eliminate documents that incidentally contain the list of words given by the user but whose contents are different from the other files the user is probably interested in. So far, the text-mining tool has proved to be a useful complementary technique.

III. Vulnerability description file search engine

For each new vulnerability, we create what we call a vulnerability description file. This file offers a characterization of the vulnerability together with pointers to all relevant documents available in the database. In the next section, we present one such vulnerability description file.
A vulnerability description file comprises a brief description of a vulnerability, a list of its characteristics, and a list of pointers to related documents within VulDa. We decided not to use any existing taxonomy so far [1-9]. Rather, we want to investigate a data-driven approach: vulnerability description files that share many characteristics will be gathered into meaningful, possibly overlapping sets. How these sets can be mapped against the existing taxonomies is something we are currently looking into.
Currently, the main purpose of these files is to offer a very easy way to retrieve vulnerabilities that match well-defined criteria. For example, we can search for remote vulnerabilities that offer root privilege on a Unix box for which an exploit is publicly available. A couple of hundreds of vulnerabilities have been classified so far and new ones are added every week.
By building VulDa, we found out that many vulnerabilities are nothing but various instantiations of a generic problem. To deal with this, we have created generic vulnerability description files. Their purpose is to provide information that is common to many other vulnerability descriptions. Figure 1 shows such a generic description file for the vulnerabilities found in SNMP. Other examples cover, e.g., the generic notions of buffer overflow, denial of services, etc. It is purposely a rather vague notion becaues we did not intend to take any *a priori* decision regarding the way we wanted to organize the vulnerabilities.

3. Vulnerability description files

Figures 1 and 2 show the content of the SNMP generic vulnerability description file. Figure 3 indicates a few pointers to other documents that can be retrieved with it. The vulnerability description itself is made up of various parts, the contents of which will be described in the following.

I. Content

Description
This section offers a very brief description of the vulnerability. More information is usually available in related documents listed in the “References” section.

Fault

We use the notion of “fault” to describe the supposed or adjudged root cause of the problem. By doing this, we try to be as consistent as possible with similar efforts that have been carried out in the dependability community to characterize accidental faults [10].

We describe the fault by providing answers to the following questions:

1. How difficult is it to understand?
2. How has the fault been introduced?
3. Where in the system is the fault located?
4. How can it be verified that the fault is actually present in the system?

Attack

An attack is a specific way to activate a fault. In other words, it is a specific way to exploit a given vulnerability. There is no bijective relationship between faults and attacks. Usually, there is more than one way to exploit a flaw.

An attack can be described in terms of 16 attributes. In order to guarantee consistency for each attribute, a list a of allowed keywords (e.g. characteristics) has been defined, only a few of which are contained in this example:

1. Difficulty to run: How difficult is it to run the attack? Launching a complex attack does not necessarily have to be difficult; it might be as simple as running a script.
2. Difficulty to understand: How difficult is it to understand the attack?
3. Availability: Is there an exploit commonly available for this attack? Is it a feature that can be abused? ...
4. Exploitability: Can one exploit the flaw remotely or is it a local attack?
5. Required privileges: What are the privileges required to run the attack?
6. Attack principle: What is the basic concept this attack is based on (e.g. buffer overflow)?
7. Location, where to detect: Where is it possible to detect that someone is running this attack?
8. Latency: How long does it take until the attack shows its effects (presuming the attack was successful)?
9. Direct impact: What are the immediate consequences of a successful attack?
10. Indirect impact: What else can you expect as a follow-up to a successful attack?

Operating System

For each attack, we provide, when available, information regarding whether an operating system is vulnerable.

Software

For each attack, we provide, when available, information regarding whether a software version is vulnerable.

Vulnerability Detectors

We provide here information regarding the various methods or software that can be used to detect the presence in a system of the described vulnerability.

Vulnerability Removal

We provide here information regarding the various methods or software that can be used to remove the described vulnerability from a system.

References

When looking at the left column of the example shown in Figure 1, information about the document such as creation and modification date, general comments, etc. can be found. Furthermore it is possible to initiate a query to retrieve documents relevant to the vulnerability. For each index available in Vulda a corresponding reference can be defined. This enables a user to learn more about a vulnerability after having obtained an overview of it.


Netscape: VulDa - 'SNMP.generic' Vulnerability Description File

File Edit View Go Communicator Help

IBM VULDA

© IBM Corporation

About the Vulnerability

Keywords:
SNMP, SNMPv1, SNMPv2, network management, MIB, management information base

Comment:
This is a very broad, generalized description of the SNMP protocol, which can be considered insecure by design.

References

In the following indices references have been found:

- Vuln. Desc. Files
- Advisories
- Attacks
- Tools
- Papers
- RFCs
- Misc
- NSA-users (unfiltered)

Retrieve Reset

About the Document

Source file:
SNMP.generic.vln

Creation:
August 4, 1998

Last update:
August 4, 1998

Available views:
- *short*
- *normal*
- *extended*

Change view

Simple Network Management Protocol

Description

SNMP (Simple Network Management Protocol) is used to monitor and manage network equipment. Version 1 (SNMPv1) is documented in RFC 1157. SNMP uses UDP port 161 for communication. Authentication is performed via a mechanism referred to as a 'community string'. Read community strings allow information to be read from the managed equipment. Write community strings allow the software configuration of the equipment to be altered. SNMP traffic should not be allowed to enter from the Internet. SNMP community strings should be difficult to guess as well. SNMPv1 does not support authentication nor secure communication.

SNMP version 2 (SNMPv2) is defined in RFC 1442 through RFC 1450. It offers extensions of the protocol and the MIBs and incorporates some security functionality. Using SNMPv2 it is possible to perform source-authentication and integrity-protection by using a hash function (MD5) and a secret that is shared between the manager and the agent. Also it is possible to encrypt messages (mostly DES).

The problem with SNMPv2 is that it is often not fully supported by the device what may force people not to use the autentication mechanisms. Also many older installations are still using SNMPv1.

Especially SNMPv1 has weak security properties and may enable an intruder to obtain information about the system configuration and often even makes it possible to manipulate the configuration of a system.

Fault 1

- Difficulty to understand: *easy*
- Fault cause: *design fault, misconfiguration*
- Location: *system services, protocol*

Attack 1

- Difficulty to run: *easy*
- Availability: *feature*
- Exploitability: *remote*
- Required privileges: *network access*
- Attack principle: *permission misuse, poor authentication*
- Location, where to detect: *network, audit trail, syslog*
- Latency: *immediate*
- Direct impact: *read system information, write system information, password revelation, DoS*
- Indirect impact: *DoS, read system information, write system information, local root*

Comment:
There are several ways of attacking a system running an SNMP agent. Because of the lack of security properties in the protocol, the community names may be easily guessed or sniffed from the network.

Figure 1: Vulnerability description file for SNMP (1/2)

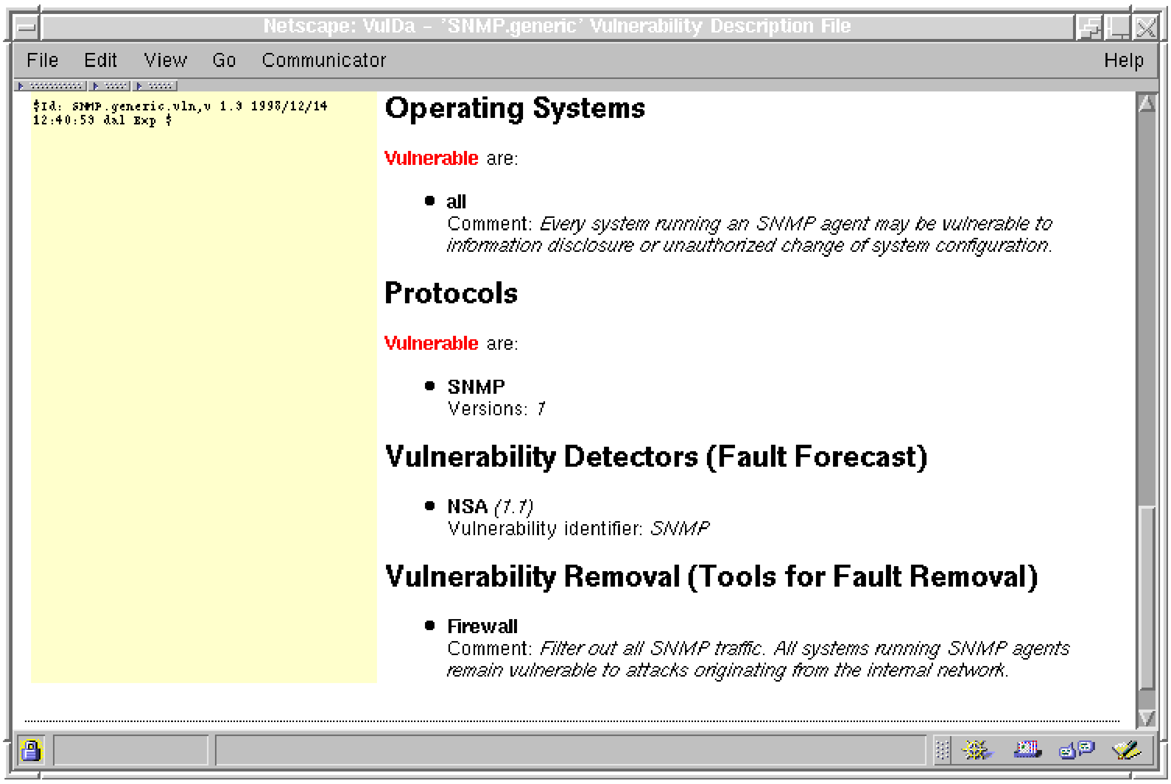

Figure 2: Vulnerability description file for SNMP (2/2)

Figure 3 shows the resolved references for the generic SNMP vulnerability description file. The first section shows all the vulnerability description files that are somehow related to the previous one. The next sections show all the references found in the remaining indexes.

4. Purposes

VulDa serves two purposes at the moment: (i) intrusion-detection research and (ii) real users within IBM who have a need-to-know.

I. Research on intrusion detection

The main research projects of the GSAL in Zurich are related to intrusion detection. One of them attempts to correlate alarms coming from various intrusion-detection tools, products, and prototypes to eliminate false alarms (positive and/or negative). Such a task can only be achieved if notions such as “alarm”, “attack” and “vulnerabilities” are uniquely defined. The vulnerability description files are used to provide some consistency to the names given by the various approaches to the attacks they claim to detect. In other words, let us suppose that an intrusion is denoted XX by product A and YY by product B, but that they actually represent the fact that an outsider is trying to exploit the same vulnerability V. In this case, we will save that information in the vulnerability description file related to V. The tool that we are developing takes advantage of this not only to correlate different alarms coming from different sources but also to present to the end user a consistent view of what has happened on his network, no matter which tool has detected the intrusion.

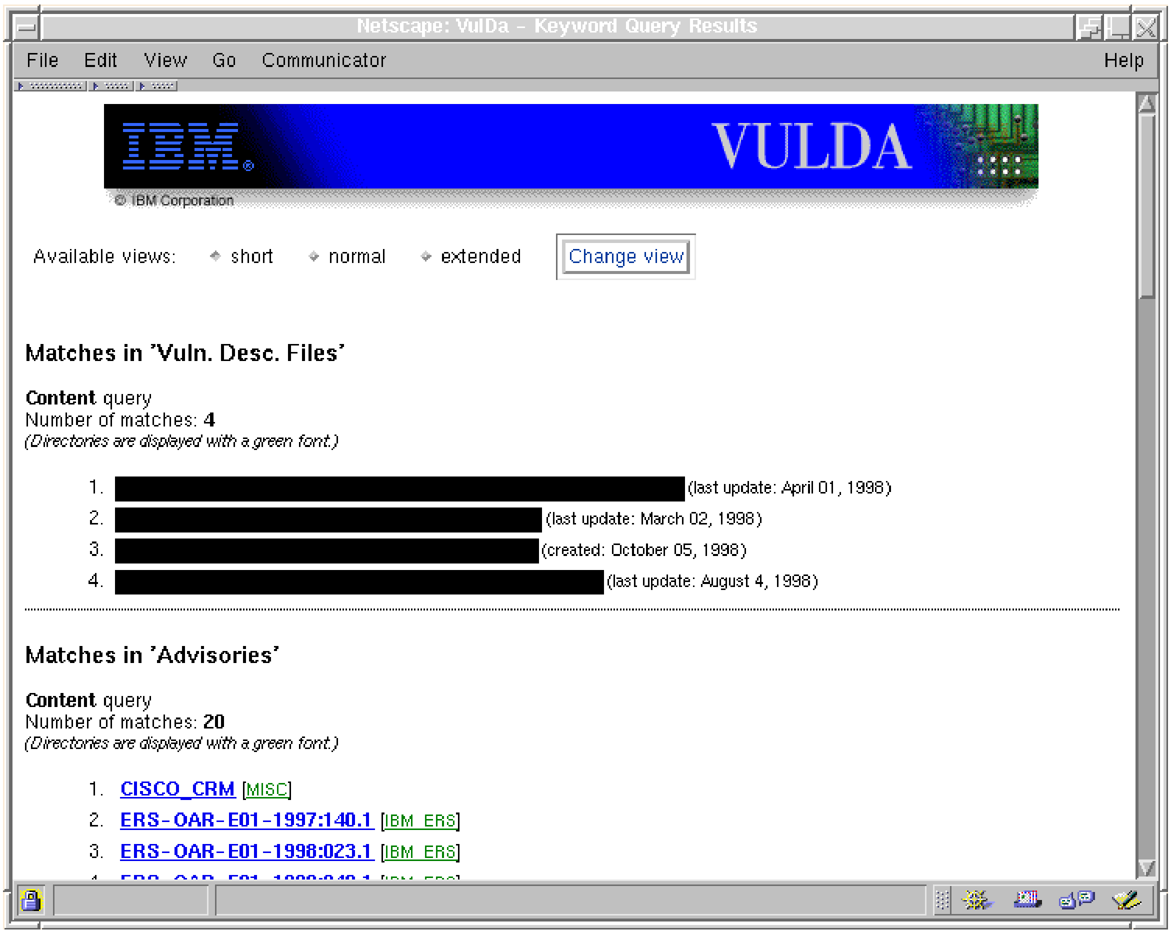

Figure 3: Example of references

II. Real-world users

VulDa is used on a daily basis by many people within IBM who have a need-to-know. Access to the database is, of course, protected by various means, and only well-identified people may gain access to it. Moreover, VulDa is located on a separate, highly protected network, accessible neither from the Internet (!) nor from the "normal" internal IBM network. People requesting access to VulDa can be from various divisions. They may be consultants, IT architects, developers, etc.

5. Cooperation issues

Even if most — not all — of the public information in VulDa is still available on the Internet, VulDa is much easier to use and richer that any of the so-called "hacker" sites. Making VulDa publicly available would not, in our opinion, be a good idea — to say the least. On the other hand, we are interested for many reasons in sharing information with others. Two years ago, we proposed opening the database to trusted parties who might have an interest in contributing to it. We are still willing to discuss these issues. Among other things, we are interested in working within the intrusion-detection community to reach an agreement on various terminology issues. This is only possible if a discussion takes place based on some concrete and pragmatic set of information that all participants have access to.

References

[1] J. Dodson, *Specification and Classification of Generic Security Flaws for the Tester's Assistant Library*, M.S. Thesis, University of California at Davis, 1996.

[2] Taimur Aslam, Ivan Krsul, and Eugene H. Spafford, *Use of A Taxonomy of Security Faults*, COAST Laboratory, Purdue University, Technical Report TR-96-051, September 1996.

[3] Sanded Kumar and Eugene Spafford, *A Taxonomy of Common Computer Security Vulnerabilities based on their Method of Detection*, COAST Laboratory, Purdue University, Technical Report, June 1995.

[4] Sanded Kumar, *Classification and Detection of Computer Intrusions*, PhD Thesis, Purdue University, Computer Sciences Department, August 1995.

[5] Andrew H. Gross, *Analyzing Computer Intrusions*, PhD Thesis, University of California at San Diego, San Diego Supercomputer Center, 1997, p. 233.

[6] Taimur Aslam, *A Taxonomy of Security Faults in the UNIX Operating System*, Master's Thesis, Purdue University, Computer Sciences Department, 1995, p. 120.

[7] Carl E. Landwehr, Alan R. Bull, John P. McDermott, and William S. Choi, *A Taxonomy of Computer Program Security Flaws, with Examples*, Information Technology Division, Naval Research Laboratory, Washington, DC 20375-5337, 1993.

[9] Ivan Victor Krsul, *Software Vulnerability Analysis*, PhD Thesis, Purdue University, Computer Sciences Department, May 1998, p. 171.

[10] J. C. Laprie (ed.), *Dependability: Basic Concepts and Terminology, Dependable Computing and Fault-Tolerant Systems*, IFIP WG 10.4, Vol. 5, Springer-Verlag, Vienna, New York, 1992.