Taxonomy for Denial-of-Service Vulnerabilities in the Linux Kernel

Rolando Sánchez Fraga
Computer Research Center
National Polytechnic Institute
Federal District, Mexico
rsanchez_b12@sagitario.cic.ipn.mx

Eleazar Aguirre Anaya
Computer Research Center
National Polytechnic Institute
Federal District, Mexico
eaguirre@cic.ipn.mx

Raúl Acosta Bermejo
Computer Research Center
National Polytechnic Institute
Federal District, Mexico
racosta@cic.ipn.mx

***Abstract*—A full understanding of DoS vulnerabilities in the Linux kernel is important for the development of applicable solutions. Most part of the work on DoS vulnerabilities in the Linux kernel focuses on designing and implementing detection, prevention or recovery tools for the kernel, but little researches tries to identify the underlying causes. Because of that, despite all research efforts, this type of vulnerabilities still prevail in major software like the Linux kernel. Partially, this is due to the absence of a source code perspective taxonomy to address all types of DoS vulnerabilities. Therefore, this research presents a taxonomy for DoS vulnerabilities through an analysis and identification of characteristics.**

I. Introduction

Despite significant advances in system protection and exploit mitigation technologies, the exploitation of software vulnerabilities persist as one of the most common methods for system compromise. Vulnerabilities are continuously discovered even in the latest versions of widely used software.

Linux kernel is a commonly attacked target. Its size and complexity are the main reasons for the existence of vulnerabilities in it. Only in 2013, almost 200 vulnerabilities in the kernel were discovered and added to the Common Vulnerabilities and Exposures (CVE) database. This is a serious problem taking in count that the OS kernel is part of the trusted computing base (TCB) and thus, any vulnerability in it could compromise the security properties of the entire system.

In the last years, the number of new CVE entries related to denial-of-service (DoS) vulnerabilities in the Linux kernel has quickly increased. From 2005 to date, each year more than half of the new CVE entries related to the Linux kernel have been DoS vulnerabilities [1].

This situation has given rise to several researches on detection of vulnerabilities via code analysis and defenses against vulnerability exploitation. However, a complete understanding of these vulnerabilities is necessary for the development of applicable solutions.

The first step in understanding vulnerabilities is to classify them into a taxonomy based on their characteristics. A taxonomy classifies the large number of vulnerabilities into a few well defined and easily understood categories. Such classification can serve as a guiding framework for performing a systematic security assessment of a system. Suitable taxonomies play an important role in research and management because the classification of objects helps researchers understand and analyze complex domains.

II. Related Work

Substantial research has focused on creating security taxonomies. Many of them center on different aspects of a security incident: some classify vulnerabilities, some methods of attack, and others security exploits. In this section, we discuss some of the previous vulnerabilities taxonomies.

At the Lawrence Livermore Laboratory [2], developed one of the first security taxonomies as a part of the RISOS (Research In Secure Operating Systems) project. This taxonomy categorizes operating system integrity flaws into seven categories. In the late 1980s, Neumann and Parker [3] published a taxonomy of computer misuse techniques in which they define nine distinct categories. Lindqvist and Jonsson [4] refined that taxonomy by dividing three of the nine categories into subcategories. This extended taxonomy offers a fairly extensive classification of methods of attack. In 1995, Aslam [5] presented a taxonomy of security faults. This taxonomy, which was developed to organize information being stored in a vulnerability database, consists of three top level categories: operational faults (or configuration errors), coding faults, and environment faults. Aslam provides a further decomposition for the coding and operational faults categories. Krsul [6] extends Aslam's work by decomposing the environmental faults category into subcategories defined by environmental objects. Each entity therein is characterized as either containing or receiving information, and has a unique name and set of operations that can be performed on it. Most of these taxonomies were later subsequently reviewed and analyzed further, as was done by [7] and [8].

While these taxonomies certainly help to classify vulnerabilities in a general way, they fail to highlight the characteristics of each vulnerability. This problem led to the design of specific taxonomies. For example, [9]–[12] focuses on C overflow vulnerabilities. These taxonomies characterize buffer overflows and help presenting information like location, data type, scope, bound, etc. that with a general taxonomy would not be possible.

However, thanks to the dimension/facet/attribute concept, categories of characteristics inherent to vulnerabilities presented on general taxonomies, like cause of the vulnerability, could be reuse on a new taxonomy.

Finally, the only founded DoS-related taxonomies are [7] which focuses on DoS attacks, and [13] which focuses on DDoS attack and defense mechanisms.

III. Objective

Design a taxonomy for the classification of denial of service vulnerabilities in the source code of the Linux kernel through the analysis of vulnerabilities and identification of characteristics to make easier the development of applicable solution for their mitigation and identification.

IV. Methodology

Based on [14], we define in Fig. 1 our methodology for the design of the taxonomy. We begin examining a subset of the vulnerabilities we want to classify. Next, we identify general characteristics of these objects. Identification of these characteristics leads to the first effort at a taxonomy. The characteristics are grouped into dimensions that form the initial taxonomy. Each dimension contains characteristics that are mutually exclusive and collectively exhaustive. This process is based on the empirical data that has been gathered about the vulnerabilities and deductive conceptualization.

Once the taxonomy is defined, we review it to look for additional conceptualizations that might not have been identified or even present in the collected data. In the process, new characteristics may be deduced that fit into existing dimensions or new dimensions may be conceptualized each with their own set of characteristics. It may even be the case that some dimension or characteristics are combined or divided. After that, we examine the vulnerabilities using the new characteristics and dimensions to determine their usefulness in classifying vulnerabilities. Out of this step comes a revised taxonomy. Then we repeat this approach, as appropriate, until we are sufficiently satisfied that the taxonomy is mutually exclusive, exhaustive, unambiguous, repeatable, accepted and useful. However, such closure is subjective and difficult to define. After the taxonomy is completed, we proceed to evaluate and test the taxonomy identifying missing or unclassified vulnerabilities.

A. Analysis of the vulnerabilities

The analysis process consist of four steps [15]: information recollection, verification, reproduction and documentation as presented in Fig. 2. For the recollection phase, multiple reliable sources of information of the vulnerabilities should be consulted. The second step involves the verification and technical examination of each of the vulnerabilities. Based on the recollected information, a manual review of the source code of the kernel is performed to confirm their existence in code and determine their cause and location. Then, in the reproduction step, the objective is to create or test a proof of concept or exploit to reproduce the vulnerability and collect additional information. Finally, all the findings must be documented. This phase is done in parallel with the other phases and ends with a *finding summary* as shown in Table I.

B. Evaluation and Testing

The evaluation of the taxonomy consists on proving that the taxonomy complies with the characteristics of a *well-defined*

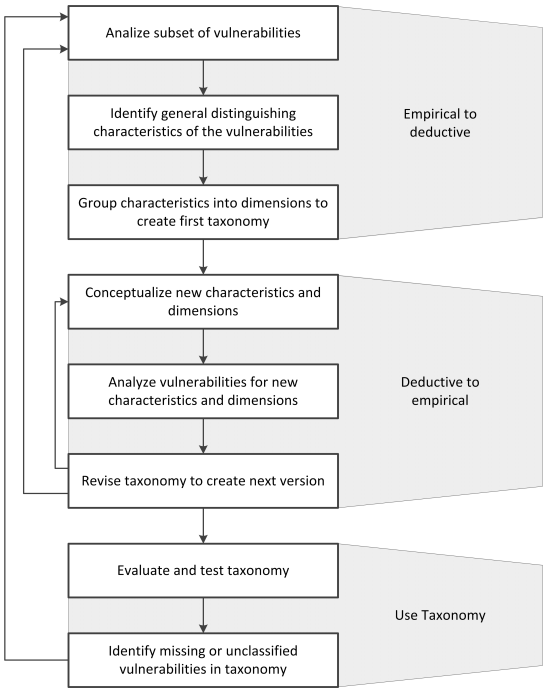

Fig. 1. Research Methodology

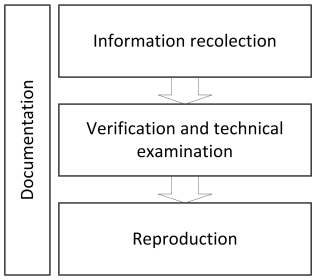

Fig. 2. Vulnerability analysis process

TABLE I. Finding Summary

Property	Value
ID	Different identifiers founded. For example, CVE and bugtraq id's.
Localization	Localization of the vulnerability. File(s), function(s), line(s).
Description	Description of the vulnerability.
Cause	Cause of the vulnerability.
Prerequisites	Prerequisites for the vulnerability to be exploited.
Remediation	Founded remediations. Patches, modules, tools, etc.
Exploit	Existence of exploits.
CVSS Severity	CVSS Severity score.
DREAD Risk	Score according to DREAD metrics.

taxonomy. According to [16], satisfactory taxonomies have classification categories with the following characteristics:

- Mutually exclusive. Classifying in one category excludes all others because categories do not overlap.
- Exhaustive. Taken together, the categories include all possibilities.
- Unambiguous. Clear and precise so that classification is not uncertain, regardless of who is classifying.
- repeatable. Repeated applications result in the same

classification, regardless of who is classifying.

- Accepted. Logical and intuitive so that categories could become generally approved.
- Useful. Could be used to gain insight into the field of inquiry.

This evaluation is subjective and difficult to define. Because of that, the phase is complemented with a test, which should show the deficiencies of the taxonomy in a clearer way.

The test is formed by two parts: The classification of the analyzed set of vulnerabilities, and the classification of a set of vulnerabilities that were not analyzed. The results of these two tests will show the correctness and robustness of the taxonomy.

Finally, according with the results, missing and unclassified vulnerabilities have to be identified and evaluate if the process should be done again or the taxonomy is finished.

V. Results

In this section, we present the current results of our research. At the time of writing this paper, a single iteration of the methodology shown in Fig. 1 (step 1 to 8) has been performed.

A. Analysis of the vulnerabilities

For the analysis, a set of 36 CVE entries of DoS vulnerabilities in the Linux kernel 3.12 were selected. These vulnerabilities comprehend the entries added in the first 8 months (between November 2013 and June 2014) of the 3.12 kernel lifetime. Linux kernel 3.12 was selected, as the latest longterm kernel at the time this research started.

For the recollection phase, multiple reliable sources were selected. We collect information of each vulnerability from sources like CVE [17], CVSS [18], CVEDetails [1], NVD [19], Openwall [20], SecurityFocus [21] and the Git source tree [22]. Collected information included a brief description of the vulnerability, severity scores (usually CVSS based), vulnerability types, vulnerable software and versions and references to advisories, confirmations, tracks, solutions, exploits and patches.

During the reproduction step, a subset of the vulnerabilities was selected to try or create a proof-of-concept (PoC) or exploit to reproduce them. Only those vulnerabilities with a low access complexity (according to CVSS score) or with a usable existing PoC or exploit were considered. A virtual machine with Debian 7.1.0 and a 3.12.0 vanilla kernel were used as a test system. Additionally, kernel dumps were configured and inspected to complement the verification step.

B. Taxonomy

With our first analysis, we identified 5 dimensions:

1) Cause of the vulnerability. Reflects the king of error in the implementation that caused the vulnerability.
2) Location of the vulnerability. Refers to the location, in the source code, where the vulnerability is present and defined as subsystems of the kernel.
3) Availability impact. Refers to the impact to availability of a successfully exploited vulnerability. All DoS attacks impact the availability of the system.
4) Access vector. Reflects how the vulnerability is exploited.
5) Exploitability. Classifies vulnerabilities according to the current state of exploit techniques and code availability.

The cause dimension includes the next 6 categories:

- Synchronization Error: These are caused by the improper serialization of the sequences of processes or an error during a timing window between two operations.
- Boundary Condition Error: These are caused by a failure to ensure that the acceptable domain or range of data is respected.
- Null Dereference: These are caused by dereferencing a pointer that is expected to be valid, but it is null.
- Failure to Handle Exceptional Conditions: These vulnerabilities arise due to failures in responding to unexpected data or conditions.
- Memory Mismanagement: This category includes vulnerabilities in kernel memory management, such as extraneous memory consumption, memory leak, double free and user-after-free errors.
- Input Validation Error: These are caused by a failure to recognize syntactically incorrect input.

The location dimension has the next categories: System Call Interface (SCI), Process Management (PM), Virtual File System (VFS), Memory Management (MM), Network Stack (NS), Arch-dependent code (Arch) and Device Drivers (DD). The availability dimension includes the next categories: Partial and Complete. The access vector dimension include the next 3 categories: Local, Local Network and Remote. And the exploitability dimension includes the next four categories: Unproven, Proof-of-Concept, Functional and Complete.

C. Evaluation and Testing

As part of our current results, we only have performed the evaluation and test of our taxonomy with the set of analyzed vulnerabilities (36 vulnerabilities). We classify our set of vulnerabilities on each of the described dimensions of our taxonomy. Tables II, III, IV and V show the results of the classification.

Table II shows that almost half of the vulnerabilities were classified as *failed to handle exceptional conditions* error. Table III shows the distribution of the vulnerabilities across the source code of the kernel. We can see that half of the vulnerabilities are included under Network Stack and Device Driver criteria. Table IV shows that almost every vulnerability would lead to a complete loss of the availability of the kernel . Table V shows that more than half of the vulnerabilities haven't been proved to be exploitable or at least exploits haven't been released. Finally, the Table V shows that only 8 of the 36 vulnerabilities could be accessed remotely.

During the evaluation, we identify the following problems on our taxonomy:

TABLE II. VULNERABILITIES CLASSIFICATION BASED ON THEIR CAUSE

Type of Vulnerability	Number
Race Condition Error	4
Boundary Condition Error	5
Null Dereference	3
Failure to Handle Exceptional Conditions	17
Memory Mismanagement	5
Input Validation Error	2

TABLE III. VULNERABILITIES CLASSIFICATION BASED ON THEIR LOCATION

Location of Vulnerability	Number
System Call Interface	3
Process Management	1
Virtual File System	3
Memory Management	3
Network Stack	14
Arch-dependent code	6
Device Driver	6

TABLE IV. VULNERABILITIES BY AVAILABILITY IMPACT

Availability Impact of Vulnerability	Number
Partial	1
Complete	35

TABLE V. VULNERABILITIES BY EXISTENCE OF EXPLOITS AND ACCESS VECTOR

Existence of exploits	Number	Access Vector	Number
Unproven	23	Local	22
Proof-of-Concept	11	Local Network	6
Functional	1	Remote	8
Complete	1		

1) Exclusivity problems with the cause dimension. Some of the vulnerabilities could be classified in multiple categories.
2) The location and access vector dimensions are ambiguous. Different subsystems share source code location, and the access vector dimension categories are not clear enough.

Because of that, we decided that at least another iteration is necessary in order to refine our taxonomy.

VI. CONCLUSION

Because vulnerabilities are central to exploiting a software application, one can prevent an exploit by identifying, and subsequently eliminating, vulnerabilities present in a software application. However, identifying if and which vulnerabilities are present is a difficult task. Factors contributing to this difficulty include the complexity of software applications, the number of potential vulnerabilities and the complexity of vulnerabilities.

Individually and collectively, these factors make identifying vulnerabilities a formidable task. However, understanding the particulars of vulnerabilities, how they are exploited, and their relationship(s) to software applications and computer system resources can facilitate this identification.

Our taxonomy, along with our analysis of vulnerabilities can help then to facilitate the identification and mitigation of denial of service vulnerabilities by their classification based on their characteristics.

REFERENCES

[1] S. Özkan. (2014, june) Linux Linux Kernel : CVE security vulnerabilities, versions and detailed reports. [Online]. Available: http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33

[2] R. P. Abbott, J. S. Chin, J. E. Donnelley, W. L. Konigsford, S. Tokubo, and D. A. Webb, "Security analysis and enhancements of computer operating systems," DTIC Document, Tech. Rep., 1976.

[3] P. G. Neumann and D. B. Parker, "A summary of computer misuse techniques," in *Proceedings of the 12th National Computer Security Conference*. Baltimore, MD, USA, 1989, pp. 396–407.

[4] U. Lindqvist and E. Jonsson, "How to systematically classify computer security intrusions," in *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*. IEEE, 1997, pp. 154–163.

[5] T. Aslam, "A taxonomy of security faults in the unix operating system," Ph.D. dissertation, Purdue University, 1995.

[6] I. V. Krsul, "Software vulnerability analysis," Ph.D. dissertation, West Lafayette, IN, USA, 1998, aAI9900214.

[7] V. Igure and R. Williams, "Taxonomies of attacks and vulnerabilities in computer systems," *Communications Surveys Tutorials, IEEE*, vol. 10, no. 1, pp. 6–19, First 2008.

[8] N. Ahmad, S. Aljunid, and J. Manan, "Understanding vulnerabilities by refining taxonomy," in *Information Assurance and Security (IAS), 2011 7th International Conference on*, Dec 2011, pp. 25–29.

[9] K. J. Kratkiewicz, "Evaluating Static Analysis Tools for Detecting Buffer Overflows in C Code," Master's thesis, 2005.

[10] N. Ahmad, S. Aljunid, and J.-l. Ab Manan, "Taxonomy of c overflow vulnerabilities attack," in *Software Engineering and Computer Systems*, ser. Communications in Computer and Information Science, J. Zain, W. Wan Mohd, and E. El-Qawasmeh, Eds. Springer Berlin Heidelberg, 2011, vol. 180, pp. 376–390.

[11] K. Kratkiewicz, "A taxonomy of buffer overflow for evaluating static and dynamic software testing tools," in *In Proceedings of Workshop on Software Security Assurance Tools, Techniques and Metrics. NIST*, 2006.

[12] D. Larochelle and D. Evans, "Statically detecting likely buffer overflow vulnerabilities," in *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, ser. SSYM'01. Berkeley, CA, USA: USENIX Association, 2001.

[13] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 39–53, Apr. 2004.

[14] R. C. Nickerson, U. Varshney, J. Muntermann, and H. Isaac, "Taxonomy development in information systems: Developing a taxonomy of mobile applications," in *17th European Conference on Information Systems, ECIS 2009, Verona, Italy, 2009*, S. Newell, E. A. Whitley, N. Pouloudi, J. Wareham, and L. Mathiassen, Eds., 2009, pp. 1138–1149.

[15] M. Dowd, J. McDonald, and J. Schuh, *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley Professional, 2006.

[16] J. D. Howard and T. A. Longstaff, "A common language for computer security incidents," Sandia National Labs., Albuquerque, NM (US); Sandia National Labs., Livermore, CA (US), Tech. Rep., 1998.

[17] The MITRE Corporation. (2014, june) CVE - Common Vulnerabilities and Exposures (CVE). [Online]. Available: https://cve.mitre.org/

[18] Forum of Incident Response and Security Teams (FIRST). (2014, june) Common Vulnerability Scoring System (CVSS-SIG). [Online]. Available: http://www.first.org/cvss

[19] National Institute of Standards and Technology (NIST). (2014, june) NVD - Home. [Online]. Available: http://nvd.nist.gov/

[20] A. Peslyak. (2014, june) oss-security mailing list. [Online]. Available: http://www.openwall.com/lists/oss-security/

[21] SecurityFocus. (2014, june) Vulnerabilities. [Online]. Available: http://www.securityfocus.com/vulnerabilities

[22] L. Torvalds. (2014, june) Linux kernel source tree. [Online]. Available: https://github.com/torvalds/linux