Towards a Faceted Taxonomy of Denial-of-Service Vulnerabilities in the Linux Kernel

Rolando Sánchez-Fraga, Eleazar Aguirre Anaya, Raúl Acosta Bermejo, and Moisés Salinas Rosales

Computer Research Center, National Polytechnic Institute, Federal District, Mexico
rsanchez_b12@sagitario.cic.ipn.mx, eaguirre@cic.ipn.mx, racosta@cic.ipn.mx, msalinasr@cic.ipn.mx

Abstract. The DoS vulnerabilities have become the most common type of vulnerabilities in the Linux kernel. In the last years, researches have focused on the design and implementation of detection and prevention tools of these vulnerabilities; however, despite all efforts they still prevail. This fact reveals the lack of a complete understanding of the causes and characteristics of such vulnerabilities. Because of that, we propose a faceted taxonomy, designed through the analysis and characterization of known DoS vulnerabilities to help developers and researchers organize and understand the records of actual vulnerabilities and then, focus their efforts to remove and prevent the introduction of new vulnerabilities through the development of applicable solutions.

Keywords: Taxonomy, Vulnerability, Denial of Service, Linux Kernel

1 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

Advances in Computer Science
Research in computing Science 81, 2014 pp. 123-133

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.

Because of that, our research focuses on the design of 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. The main contributions of our work are the definition of the taxonomy and the results of the analysis in which it is based.

The paper is organized as follows: we examine previous work on vulnerability taxonomies in section 2, followed by a description of the methodology used in order to develop the taxonomy in section 3. Afterwards, in section 4, we present the current results of our research; the defined taxonomy, the classification of the vulnerabilities we analyze, as well as statistics related to the occurrence of certain vulnerabilities in our vulnerability set. Finally, in section 5, we conclude.

2 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. All previous taxonomies have one main objective; to minimize exploitable software vulnerabilities.

[2,3,4,5,6,7,8,9,10,11] presented general vulnerability taxonomies. Other works, like [12,13,14] focus on overflows and constructed C vulnerabilities taxonomies focusing on C overflow vulnerabilities. [15] and [16] center on DoS and DDoS attacks and defenses taxonomies. Most of these taxonomies were later subsequently reviewed and analyzed further, as was done by [15] and [17].

3 Methodology

Based in [18], we define in Figure 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.

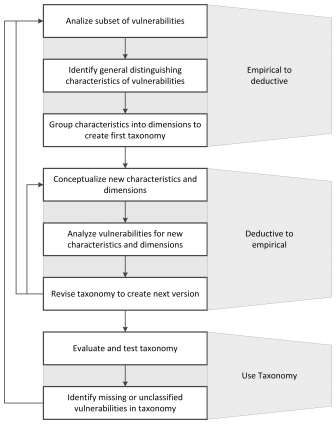


Fig. 1. Investigation Methodology

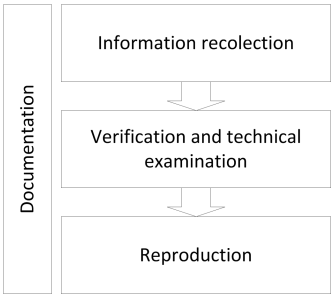


Fig. 2. Vulnerability analysis process

3.1 Analysis of the vulnerabilities

The analysis process consist of four steps [19]: information recollection, verification, reproduction and documentation as presented in Figure 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 recollect 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 showed in Table 1.

Table 1. Finding Summary

Property	Value
ID	Different identifiers founded. For example, CVE and bugtraq IDs.
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.

3.2 Evaluation and Testing

The evaluation of the taxonomy consists in proving that the taxonomy complies with the characteristics of a *well-defined* taxonomy. According to [7], 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.

A taxonomy, however, is an approximation of reality and as such, a satisfactory taxonomy should be expected to fall short in some characteristics. Moreover, 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 set of vulnerabilities analyzed, 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.

4 Results

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

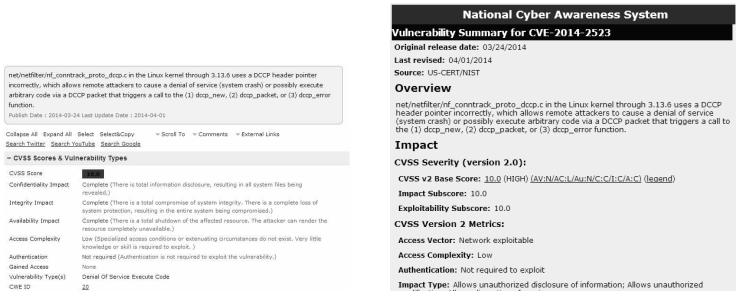
4.1 Analysis of the vulnerabilities

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[20], CVEDetails[1], NVD[22], Openwall[23], SecurityFocus[24], Linux kernel Mailing List[25] and Git Linux kernel source tree[26]. Collected information included a brief description of the vulnerability, severity scores (usually CVSS[21] 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.

As an example, we show part of the analysis of the vulnerability CVE-2014-2523.



(a) CVEDetails (b) NVD

Fig. 3. CVE-2014-2523 information

Figure 3 shows the information recollected from CVEDetails and NVD websites. Both sites present the official description from the CVE database. NVD presents raw CVSS information, while CVEDetails present the same information but in a more descriptive way. Fortunately, the description is good enough to start the code analysis. Now we search any of the three mentioned functions (*dccp_new*, *dccp_packet* and *dccp_error*) and check for any use of a DCCP header pointer.

```
static bool dccp_new(struct nf_conn *ct, const struct sk_buff *skb,
                     unsigned int dataoff, unsigned int *timeouts)
{
    struct net *net = nf_ct_net(ct);
    struct dccp_net *dn;
    struct dccp_hdr _dh, *dh;
    const char *msg;
    u_int8_t state;

    dh = skb_header_pointer(skb, dataoff, sizeof(_dh), &dh);
    BUG_ON(dh == NULL);
```

Fig. 4. dccp_new vulnerable function

Table 2. CVE-2014-2523 Finding Summary

Property	Value
ID	CVE-2014-2523, 66279
Localization	net/netfilter/nf_conntrack_proto_dccp.c dccp_new (line 431), dccp_packet (line 489), dccp_error (line 580)
Description	Incorrect use of a DCCP header pointer, which allows remote attackers to cause a denial of service or possibly execute arbitrary code via a DCCP packet that triggers a call to the dccp_new, dccp_packet, or dccp_error function.
Cause	Buffer overflow trying to copy dccp header in a pointer address.
Prerequisites	Netfilter enabled in kernel configuration (disabled by default). conntrack and dccp_conntrack modules installed and running.
Remediation	Change &dh for &_dh in dccp_new, dccp_packet and dccp_error functions. (commit b22f5126a24b3b2f15448c3f2a254fc10cbc2b92)
Exploit	Non-existent. A PoC was created.
CVSS Severity	High (10.0) (AV:N/AC:L/Au:N/C:C/I:C/A:C)
DREAD Risk	High Priority (8.4) (DP:10/R:9/E:9/AU:6/D:8)

As we can see in Figure 4, *dh is the pointer we are looking for. This pointer is used to store the returned value of the function *skb_header_pointer*. As parameters, we send the buffer, a data offset, the size of the dccp header and the reference of the header; however, we are sending the reference of the pointer instead. If we take a look at the code of the *skb_header_pointer* function, it calls another function that copies the content to the referenced buffer. However, the reference received as a parameter is a pointer not the buffer. That means that if we receive a malformed DCCP packet, we could end overwriting memory on the stack causing a complete denial of service. Worth noticing that if we receive a special crafted packet and have no stack protection, we could end up executing some malicious code.

Finally we document the results of our analysis. As mentioned before we create a *finding summary*. Table 2 presents the finding summary for the vulnerability CVE-2014-2523.

4.2 Taxonomy

With our first analysis, we identified 7 facets:

– Cause of the vulnerability. Reflects the kind of error in the source code that caused the vulnerability.

- Location of the vulnerability. Refers to the location, in the source code, where the vulnerability is present and defined as subsystems of the kernel.
- Availability, integrity and confidentiality impact. Refer to the availability / integrity / confidentiality impact of a successfully exploited vulnerability.
- Access vector. Reflects how the vulnerability is exploited.
- Exploitability. This taxonomy classifies vulnerabilities according to the current state of exploit techniques and code availability.

The cause facet includes the next 5 categories:

- Validation Error: These vulnerabilities arise due to failures in responding to unexpected data or conditions.
- Synchronization and Timing Error: These are caused by the improper serialization of the sequences of processes or an error during a timing window between two operations.
- Resource Management Error: Vulnerabilities in this category are related to improper management of system resources.
- Exception Handling Error: These are caused by the improper response to the occurrence of exceptions.
- General Logic Error: These vulnerabilities are caused by a bad logic in the implementation. Errors like using the wrong operator, uninitialized variable, missing parameter, assigning instead of comparing and wrong operand order belong to this class.

The location facet has the next categories:

- System Call Interface (SCI): Layer that provides the means to perform function calls from user space into the kernel. Founded in /linux/kernel an /linux/arch.
- Process Management (PM). Focused on the execution of processes. Founded in /linux/kernel and /linux/arch.
- Virtual File System (VFS). Provides a common interface abstraction for file systems. Sources are founded in /linux/fs.
- Memory Management (MM). Sources founded in /linux/mm.
- Network Stack (NS). Sources founded in /linux/net.
- Arch-dependent code (Arch). Sources founded in /linux/arch.
- Device Drivers (DD). Sources founded in /linux/drivers. Part of the kernel that makes a particular hardware device usable.

The availability facet includes the next categories:

- Partial: There is reduced performance or interruptions in resources availability.
- Complete: There is a total shutdown of the kernel.

Confidentiality facet includes the following classes:

- None: There is no impact to the confidentiality of the system.

– Partial: There is considerable informational disclosure. Access to some system files is possible, but the attacker does not have control over what is obtained, or the scope of the loss is constrained.
– Complete: There is total information disclosure, resulting in all system files being revealed. The attacker is able to read all of the system's data (memory, files, etc.).

Integrity facet includes the following classes:

– None: There is no impact to the integrity of the system.
– Partial: Modification of some system files or information is possible, but the attacker does not have control over what can be modified, or the scope of what the attacker can affect is limited.
– Complete: There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised. The attacker is able to modify any files on the target system.

The access vector facet includes the next 3 categories:

– Local. Refers to vulnerabilities exploitable only having an interactive, local (shell) account that interfaces directly with the underlying OS.
– Local Network: These vulnerabilities require having access to either the broad-cast or collision domain of the vulnerable software.
– Remote. Refers to those vulnerabilities remotely exploitable.

And the exploitability facet includes the next four categories:

– Unproven: No exploit is publicly available or an exploit is entirely theoretical.
– Proof-of-Concept: Proof-of-concept exploit code or an attack demonstration that is not practical for most systems is available.
– Functional: Functional exploit code is available. The code works in most situations where the vulnerability exists.
– High: Either the vulnerability is exploitable by functional mobile autonomous code, or no exploit is required (manual trigger) and details are widely available.

4.3 Evaluation and Testing

For this phase, we only performed the evaluation and the test with our set of analyzed vulnerabilities. We classify our set of vulnerabilities on each of the described dimensions of our taxonomy. Tables 3 to 9 show the results of the classification.

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

1. The facets' hierarchies need to be expanded. Subclasses are needed for a better description.
2. The location and access vector dimensions are ambiguous.

Because of that, we decided that at least another iteration of our process is necessary. The location dimension could be divided according to the source files and directories structure, rather than by subsystem. With the cause and access vector dimensions, hierarchical classes could be defined to avoid ambiguity.

Table 3. Vulnerabilities based in their cause

Type	Number	%
Validation Error	22	61.11
Synchronization and Timing Error	7	19.44
Resource Management Error	3	8.33
Exception Handling Error	2	5.56
General Logic Error	2	5.56

Table 4. Vulnerabilities based in their location

Location	Number	%
System Call Interface	3	8.33
Process Management	1	2.78
Virtual File System	3	8.33
Memory Management	3	8.33
Network Stack	14	38.89
Arch-dependent code	6	16.67
Device Driver	6	16.67

Table 5. Vulnerabilities by availability impact

Availability Impact	Number	%
Partial	1	2.78
Complete	35	97.22

Table 6. Vulnerabilities by confidentiality impact

Confidentiality Impact	Number	%
None	27	75
Partial	2	5.56
Complete	7	19.44

Table 7. Vulnerabilities by integrity impact

Integrity Impact	Number	%
None	27	75
Partial	2	5.56
Complete	7	19.44

Table 8. Vulnerabilities by existence of exploits

Existence of exploits	Number	%
Unproven	23	63.89
Proof-of-Concept	11	30.56
Functional	1	2.78
Complete	1	2.78

Table 9. Vulnerabilities by access vector

Access Vector	Number	%
Local	22	61.11
Local Network	6	16.67
Remote	8	22.22

5 Conclusion

The results of an analysis of DoS vulnerabilities on Linux, a new DoS vulnerabilities taxonomy and the classification of the analyzed vulnerabilities with the new taxonomy were presented. The analysis results will enable Linux kernel maintainers, developers and researchers to better understand denial-of-service vulnerabilities and prioritize their efforts according to the results presented. That way, better tools, techniques and defenses will be created. Differently from previous work, the scope of our taxonomy is DoS vulnerabilities in order to highlight the characteristics and provide more useful information with the classification.

References

1. CVE Details. Linux kernel, http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33
2. Krsul, I.V.: Software vulnerability analysis. pp. 171. Purdue University (1998)
3. Lough, D.L.: A taxonomy of computer attacks with applications to wireless networks. pp. 348. Virginia Polytechnic Institute and State University (2001)
4. Aslam, T.: A Taxonomy of Security Faults in the UNIX Operating System. (1995)
5. Bazaz, A., Arthur, J.D.: Towards a Taxonomy of Vulnerabilities. ;In HICSS (2007)
6. Gegick, M., Williams, L.: Matching attack patterns to security vulnerabilities in software-intensive system designs. ;ACM SIGSOFT Software Engineering Notes(2005)1-7
7. Howard, J.D., Longstaff, T.A.: A common language for computer security incidents (1998)
8. Tsipenyuk, K., Chess, B., McGraw, G.: Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors. ;IEEE Security & Privacy(2005)81-84
9. Hansman, S., Hunt, R.: A taxonomy of network and computer attacks. ;Computers & Security(2005)31-43
10. Killourhy, K.S., Maxion, R.A., Tan, K.M.C.: A Defense-Centric Taxonomy Based on Attack Manifestations. ;In DSN(2004)102-102
11. Landwehr, C.E.: Formal Models for Computer Security. ;ACM Comput. Surv.(1981)247-278
12. Kratkiewicz, K.: Evaluating Static Analysis Tools for Detecting Buffer Overflows in C Code. (2005)
13. Ahmad, N., Aljunid, S., Ab Manan, J.-l.: Taxonomy of C Overflow Vulnerabilities Attack. In: Zain, J., Wan Mohd, W., El-Qawasmeh, E. (eds.) Software Engineering and Computer Systems, vol. 180, pp. 376-390. Springer Berlin Heidelberg (2011)
14. Kratkiewicz, K., Lippmann, R.: A taxonomy of buffer overflows for evaluating static and dynamic software testing tools. Proceedings of Workshop on Software Security Assurance Tools, Techniques, and Metrics 500, 44 (2006)
15. Igure, V., Williams, R.: Taxonomies of attacks and vulnerabilities in computer systems. Communications Surveys & Tutorials, IEEE 10, 6-19 (2008)
16. Mirkovic, J., Reiher, P.: A taxonomy of DDoS attack and DDoS defense mechanisms. SIGCOMM Comput. Commun. Rev. 34, 39-53 (2004)
17. Ahmad, N.H., Aljunid, S.A., lail Ab Manan, J.: Understanding vulnerabilities by refining taxonomy. IAS, pp. 25-29. IEEE (2011)
18. Nickerson, R. C., Muntermann, J., Varshney, U., Isaac, H.: Taxonomy Development in information systems: Developing a taxonomy of mobile applications (2009)
19. Dowd, M., McDonald, J., Schuh, J.: The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities. Addison-Wesley Professional (2006)
20. Common Vulnerabilities and Exposures, https://www.cve.mitre.org
21. Common Vulnerability Scoring System v2.0, http://www.first.org/cvss/cvss-guide
22. National Vulnerability Database, http://nvd.nist.gov
23. Openwall, http://www.openwall.com/
24. SecurityFocus, http://www.securityfocus.com
25. Linux kernel mailing list, https://lkml.org/
26. Linux kernel source tree, https://github.com/torvalds/linux