

# Security Testing using a Susceptibility Matrix

**Kanta Jiwnani**

Department of Computer Science  
University of Maryland  
College Park, Maryland 20742.  
kanta@cs.umd.edu

**Marvin Zelkowitz**

Department of Computer Science, University of Maryland  
and Fraunhofer Center for Experimental Software Engineering  
College Park, Maryland 20740.  
marv@zelkowitz.com

## 1. Introduction

Software testing is a cost effective method to detect faults in software [1]. Similarly, Security testing is intended to assess the trustworthiness of the security mechanisms and is often regarded as a special case of system testing [2]. The emphasis of Security testing is not to establish the functional correctness of the software but to establish some degree of confidence in the security mechanisms [2]. It is the single most common technique for gaining assurance that a system operates within the constraints of a given set of policies and mechanisms. Presently, there is no systematic approach to security testing. Our goal has been to devise a classification scheme to increase testing effort in high-risk areas and help the software community to get feedback to improve continuously.

## 2. Security Background

Organizations test the security of their systems, firewalls and networks either by using commercially available vulnerability tools, penetration testing, or by using formal methods. Other methods for security testing have been developed, including syntax testing, property-based testing, fault injection, mutation testing and Gligor's testing method. These techniques are limited to finding specific security flaws. Also, there are the general testing techniques like path testing, domain testing, and data flow testing. However these techniques are not specifically adapted for security issues. Another approach to assess the security of the system is test for specific security vulnerabilities. Taxonomy of vulnerabilities helps us understand their distribution in the system. A number of flaw

taxonomies have been developed including the Protection Analysis Taxonomy (1978), the Research in Secured Operating Systems security taxonomy (1976), Spafford's taxonomy (1992), Landwehr's taxonomy (1994), Aslam's taxonomy (1995), Bishop's taxonomy (1995), Du and Mathur's taxonomy (1997), Brian Marick Survey (1990) and Chillarege's Orthogonal Defect Classification. Since our goal was to look at the impact that security flaws have on an evolving product, how a flaw occurs, when it occurs, and its impact appeared to be the right mix of criteria. We centered on Landwehr's model [1] as the basis for our work.

## 3. Susceptibility Matrix

We developed a 3-dimensional taxonomy (shown in Figure 1). Cause and Location are simplified versions of Genesis and Location dimensions of Landwehr's model. We included a third dimension, impact, to be able to prioritize testing effort [2]. We classified 853 flaws found in all versions of Windows and 160 flaws found in Red Hat Linux using this classification scheme. Each vulnerability is associated with a triple: <cause, location, and impact> of flaw. Using the taxonomy we construct a Susceptibility matrix. Each entry in this matrix has a vector of impacts. The result of this construction for Windows is shown in Figure 4. Susceptibility matrix provides the system developers and testers with a view of the system's vulnerable areas showing the impact an exploit of these vulnerable areas would result in. We constructed a similar Susceptibility matrix for Linux. These two matrices are combined and shown graphically in Figure 2 with the data given in Figure 3. The left semicircle represents Windows

Cause	Location	Impact
Validation Errors	System Initialization	Unauthorized Access
Domain Errors	Memory Management	Root or System Access
Serialization or aliasing errors	Process Management or Scheduling	Denial of Service
Inadequate Identification or Authentication	Device Management	Integrity Failure
Boundary and Condition Errors	File Management	Termination
Trojan Horse	Identification or Authentication	Failure
Covert Channel		Invalid State
Exploitable Logic Errors		File Manipulations
		Errors due to clock changes

**Figure 1. Security Flaw Taxonomy from a Security Testing Perspective.**

	Validation	Domain	Serial / Aliasing	Identifn / Authen	Boundary Violation	Exploitable Logic	Trojan Horse	Covert Channel
System Initialization	●	○	∩	●	∩	●	∩	
Memory Mgmt	●	○		○	○	○	∩	
Process Mgmt	○	∩	○	○		∩	∩	
Device Mgmt	○			○		∩	∩	
File Mgmt	○	∩	∩	○	○	○	∩	
Identification / Authen	◐	∩	∩	●		◐	◐	

Figure 2. Comparison of Windows and Linux flaws.

Density	Windows % Of Flaws	Linux % Of Flaws
Total (all)	100%	100%
Common High (black circle)	67%	62.5%
High (black semi circle)	85.5%	68%
Common Low (white circle)	9%	24%
Low (white semi circle)	16%	31%
Common Null (empty)	0%	0%

Figure 3. Percentage of total flaws.

while the right semicircle represents Linux. Black indicated  $\geq 40$  Windows ( $\geq 8$  Linux) flaws while white indicates fewer flaws using a ratio of 5:1 for relative number of Windows to Linux flaws. Emphasizing testing on only the 5 black circles identifies about two-thirds (67% of Windows and 63% of Linux) of the vulnerabilities in both systems. Looking at each system independently, the black semicircles represent 85.5% of Windows and 68% of Linux flaws.

#### 4. Conclusions

We have shown that using a Susceptibility Matrix, as a driver for testing would help us identify most of the security vulnerabilities found in real systems. The information captured by a Susceptibility Matrix reflects the organization's environment and hence accurately identifies the problem areas in its software.

#### 5. References

- [1] G. Myers, "The Art of Software Testing", Wiley, 1979.
- [2] B. Beizer, "Software Testing Techniques", Van Nostrand Reinhold, New York, 1990.
- [3] C. E. Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi, "A taxonomy of computer program security flaws," ACM Computing Surveys, Vol. 26 (3), pp. 211-254, 1994.
- [4] K. Jiwnani and M. Zelkowitz, "Maintaining Software with a Security Perspective", IEEE International Conference on Software Maintenance, Montreal Canada, October 2002.

CAUSE	→								Sum	Ranks
	Validation	Domain	Serial / Aliasing	Identification / Authentication	Boundary Violation	Exploitable Logic	Trojan Horse	Covert Channel		
System Initialization	48 DoS (2 Med), 19 Unauthorized Access (1 High, 6 Med), 1 Crash, 1 Root (1 High), 3 Invalid, 6 Failure, 5 Integrity Failure, 1 clock	11 Unauthorized Access (1 Med), 3 Integrity Failure, 1 DoS		109 Unauthorized Access (6 High, 35 Med), 2 Root Access (1 High, 4 Med), 10 DoS (2 Med), 2 Invalid (1 Med), 5 Integrity Failure (1 Med), 1 File Manipulation (1 Med)	5 DoS, 1 Integrity Failure	115 Unauthorized Access (6 High, 31 Med), 11 Root (2 High, 3 Med), 82 DoS (12 Med), 25 Integrity Failure, 4 Failure (1 Med), 1 invalid, 1 clock, 3 File Manipulation	22 Unauthorized access (12 Med), 2 Root, 3 DoS (1 Med)		505	1
Memory Mgmt	34 Unauthorized Access (8 High, 18 Med), 16 DoS (2 Med), 6 Root (2 High, 2 Med)	1 Unauthorized Access, 2 DoS		1 Unauthorized Access, 1 DoS	1 Crash, 2 DoS, 1 Failure	11 DoS (2 Med)	2 Unauthorized Access (1 Med)		77	3
Process Mgmt	1 DoS (1 Med)	1 DoS	1 DoS	1 Unauthorized Access (1 High), 1 DoS (1 Med)		5 DoS			10	6
Device Mgmt	2 DoS			10 Unauthorized Access (1 Med), 2 DoS		1 Root Access (1 High)	Unauthorized Access (1 Med)		15	5
File Mgmt	1 DoS, 1 Failure, 4 Unauthorized Access (1 Med), 1 File Manipulation (1 Med)	2 DoS		3 Unauthorized Access (1 Med), 3 DoS, 4 File Manipulation (1 High, 1 Med)	1 DoS, 1 Crash	3 Unauthorized Access (1 High), 1 Invalid, 2 DoS	1 Invalid (1 High), 3 Unauthorized Access (2 High)		31	4
Identification / Authen	3 Unauthorized Access, 1 File Manipulation (1 Med), 3 DoS (1 Med)			13 Root Access (6 High, 6 Med), 42 Unauthorized Access (5 High, 18 Med), 2 DoS, 1 Crash (1 Med), 1 Integrity Failure, 2 Invalid (2 Med)		13 Root access (4 High, 7 Med), 40 Unauthorized access (4 High, 11 Med), 6 DoS (2 Med), 1 File Manipulation	Med), 76 Unauthorized access (11 High, 27 Med), 2 Invalid, 5 DoS (2 Med), 1 Hang		215	2
Sum		155	21	1	218	12	326	120	0	
Ranks		3	5	7	2	6	1	4	8	

Figure 4. Susceptibility Matrix for Windows flaws.