A Classification of Malicious Software Attacks

Hanno Langweg and Einar Snekkenes
Norwegian Information Security Laboratory – NISlab
Department of Computer Science and Media Technology, Gjøvik University College
P.O. Box 191, 2802 Gjøvik, Norway
hanno.langweg@hig.no, einar.snekkenes@hig.no

Abstract

This paper presents a classification of attacks by malicious software. Unlike previous schemes, it focuses on application software instead of operating systems. We classify attacks pertaining to input, processing, and output of an application. Our scheme can be used to adapt testing strategies and is intended as a step towards developing a security metric for assessing robustness of programs.

1. Introduction

Many classifications of software vulnerabilities exist in the literature. Most of them stress the kinds of design and programming errors that can be exploited by attackers. Often the focus rests on operating systems. However, applications take over increasingly more of the business processes and increasingly define their own security demands. Attacking applications will become more rewarding from an attacker's perspective. Making programs more robust against attacks should be guided by an analysis of exposure and severity of vulnerabilities.

We present a classification of attacks by malicious software. Unlike previous schemes, it focuses on application software instead of operating systems. We consider application interfaces rather than functional components, and claim impact of an attack as a measure of its importance. The major difference between operating systems and applications is that operating systems offer a general functionality whereas applications offer a special functionality making use of the operating system infrastructure. We classify attacks pertaining to input to a program (location), exploitable logic in processing by a program (cause), and output of a program (impact).

Our scheme furthers a better understanding of preferential ways to exploit vulnerabilities in software. It helps in searching for vulnerabilities in a systematic and automatic procedure, e.g. with vulnerability scanners. Software developers profit from a classification as it helps them to identify vulnerable interfaces of and processing by a program. This classification is also a sound basis to develop a security metric, i.e., here, to quantitatively evaluate the robustness of an application. It could also be used when assessing the effectiveness of countermeasures.

In the next section, we explore features of malicious software and outline our attack scenario. This is followed by a review of the state of the art. Section four presents our classification of malicious software attacks.

2. Features of malicious software

We concentrate on attacks by malicious software on the same machine as the attacked application. Of all possible attack scenarios we limit ourselves with respect to attacker capabilities and target objects. Attacks are carried out by a program running on the same machine as the target program, i.e., same hardware, same operating system instance. Both programs are executed with a similar set of privileges and possibly on behalf of the same user account. This is shown in the figures below. We acknowledge that programs often are not distinguished in existing implementations of access control models, i.e., users and programs are treated as the same subjects. We further assume that there are numerous paths of attack and that an attacker will initially succeed in executing the malicious program one way or another.

Malicious software defies traditional node perimeter security because it is executed within the machine perimeter. An application perimeter is more difficult to define and difficult to defend when flexibility is also a design goal. Application security demands may not be known to the operating system. Hence, hostile applications are difficult to defend against by operating systems alone because security demands may be difficult to express and handle centrally for a variety of applications. There may not be a reference monitor for interacting with programs. Malicious programs, when executed, may interact with the operating system to violate operating system security. They may as well interact with other applications to violate application security demands.

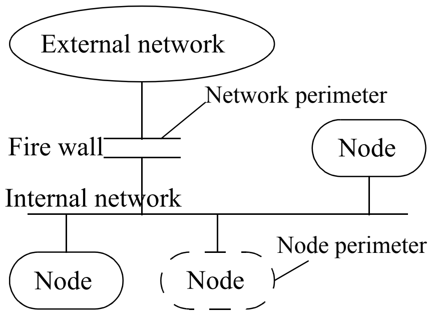

Fig. 1 Network and node perimeter

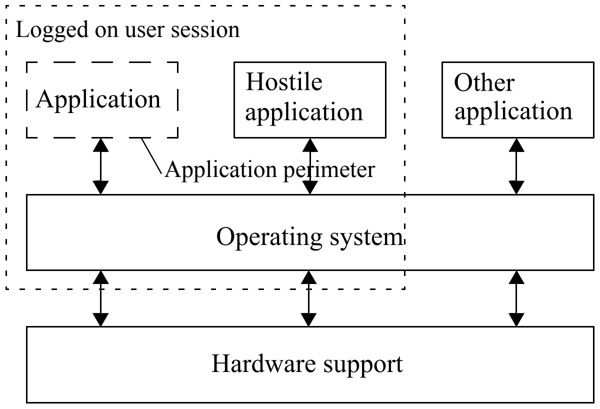

Fig. 2 Application perimeter

2.1. Differences between modelling applications and operating systems

Operating systems have a general model stating which functional components they should have, e.g., memory management, file management, scheduling, security kernel etc. Applications do not have such a general architecture. They do not offer general functionality like operating systems do, but can in principle be designed to offer arbitrary functionality. A selection of functional components could comprise application logic, persistent and volatile data, operating system interaction, security enforcement. However, application logic is a very general term. Data storage and operating system interaction are included in our location dimension, flaws in security enforcement are covered by the cause dimension. Hence, we regard interfaces as more appropriate than functional components with respect to location.

Cohen [6] uses a Turing machine in his pioneering work on computer viruses. It may be tempting to discuss malicious software attacks based on Turing machines. However, this model could be too simple when transferring results to existing systems. An approach could be to have Turing machines with multiple tapes and multiple finite controls, or two machines using the same tape.

Model–View–Controller [8] is a design pattern for interactive applications. It divides functionality among objects involved in maintaining and presenting data to minimize the degree of coupling between the objects. The architecture maps traditional application tasks – input, processing, and output – to the graphical user interaction model.

3. Previous and related work

All of the past classifications we surveyed focus on operating systems, half of them take applications into account. All classify vulnerabilities corresponding to the kind of design or programming flaw in varying detail. Differences are mostly visible as regards location of a flaw, the impact of an attack, and the importance of flaw remediation. Location is available for operating systems and based on [13]. Impact revolves around the traditional categories confidentiality, integrity, availability, with some authors ([7], [10]) adding more categories. The fix dimension appears to be of little value.

Table 1 Features of existing classifications

Work	Focus	Cause	Location	Impact	Fix
RISOS [1]	OS	yes	–	–	–
PA [3]	OS	yes	–	–	–
Hogan [9]	OS	yes	–	–	–
Landwehr [13]	OS	yes (extension of [1])	OS functional components	–	–
Aslam [2]	OS, app.	yes (reduction of [13])	–	–	–
Bishop [4]	OS, app.	depends on point of view	depends on point of view	–	–

Table 1 Features of existing classifications

Work	Focus	Cause	Location	Impact	Fix
Lindqvist [14]	OS, app.	yes	–	14 CIA	–
Du [7]	OS, app.	yes (based on [13])	–	Exec+CIA	mostly "?"
Krsul [12]	OS, app.	yes	–	–	–
Piessens [16]	OS, app.	yes	–	–	–
Jiwnani [10]	OS	yes (based on [13])	OS fct. cmp. ([13])	9	–

Classification of the cause of a vulnerability in many cases builds on early work done in [1] and [3] that was later extended by [13]. We give an overview and comparison in the table below. In the Cause column the values are shown according to [7] which is a recent modification of [13]. Each column shows a classification and how it relates to the Cause column. A '+' sign indicates that the category is supported by the particular classification. Sometimes a classification has a smaller number of categories or splits a category into subcategories. A '–' sign indicates that the category is not supported.

Table 2 Cause (exploitable logic in processing) as used in earlier work

Cause	RISOS [1]	PA [3]	Hogan [9]	Landwehr [13]	Aslam [2]	Bishop [4]	Lindqvist [14]	Du [7]	Krsul [12]	Piessens [16]	Jiwnani [10]
Validation error	Incomplete ~	+	–	+	Coding Faults – Condition Validation Errors	Depends on level of abstraction	–	+	Coding Flaws	Insufficiently defensive input checking	+
	Inconsistent parameter validation										
Boundary checking error	+		+	+			Resource exhaustion	+			+
Authentication error	+	+	+	+			Bypassing intended controls	+		–	+
Domain error	+		–	+	+		Inadvertent write permission	+		Non-secure abstractions, Feature interaction	+
Serialization/ aliasing error	+	+	–	+	+		–	+		Non-atomic check and use	+
Other exploitable logic error	+	–	–	+	–		–	+		Programming bugs	+
Weak or incorrect design error	–	–	Imperfect Protection	Intention-al–Non-Malicious –Other	+		–	+		Reuse in hostile environment, Trading off security, Insecure defaults	–

In our opinion, Du [7] is based on established earlier work and has the most complete set of categories for the cause dimension.

Previous work on the location dimension is inadequate for our purposes as it is not suitable for applications. Approaches to classifying impact of an attack focus on the classic security goals of confidentiality, integrity, and availability. We deem the addition of code execution useful as this is more specific than the very general notion of a violation of code integrity.

3.1. Vulnerability data bases

We also surveyed a number of vulnerability data bases that contain instances of security problems. They are usually employed to determine vulnerabilities in a specific product and to determine how an installation can be repaired or protected against exploitation. Unfortunately, most of the information is rather descriptive and intended to be read. Query interfaces often allow a search only for keywords. In one case, Coop VDB, expressiveness was so high that it lead to vulnerabilities not being classified according to categories but in natural language.

We restricted our survey to data bases that were publicly available without requiring a paid subscription.

The cause and impact dimensions yield similar results compared with our survey of vulnerability classifications. Location often contains a concrete program or operating system utility in which the flaw occurred. Fix usually refers to whether or not a patch is available or how the vulnerability can be remedied by changing the configuration of the system.

Information gathered from vulnerability data bases is overly descriptive and helps only partly in developing a classification.

Table 3 Features of existing vulnerability data bases

Data base	Cause	Location	Impact	Fix
CERT/CC Vulnerability Notes Database www.kb.cert.org/vuls	description	description	description	description
Coop VDB https://cirdb.cerias.purdue.edu/coopvdb/public/	description (193.200)	description	17 (8)	description
ICAT Metabase icat.nist.gov	10	8	7	link
ISS X-Force http://www.iss.net/security_center/search.php	description	description	description	description
OSVDB www.osvdb.org	description	description	description	description
Scip Verletzbarkeitsdatenbank www.scip.ch/cgi-bin/smss/showadvf.pl	12	–	12	description
Security Focus www.securityfocus.com/bid	10	2	description	description

4. Arranging attacks in a classification

Our classification is based on three dimensions: location, cause, and impact. Location of a vulnerability differs in applications compared to operating systems. Operating systems have components according to a generally accepted model and offer specified functionality to a variety of applications. Operating systems are part of the infrastructure. Applications do not have a similar general model (cf. also 2.1.). They fulfil arbitrary tasks and can be structured in many different ways. Hence, it is not useful to classify vulnerabilities by location the same way as Landwehr [13] does, defined as the place where the vulnerability manifests in the system, e.g., memory management. It is more promising to classify vulnerability location by fault introduction location, i.e., where the fault crosses the application's boundary. This is done in accordance with Jonsson [11] where a vulnerability is defined as the place where a fault can be introduced.

Cause of a vulnerability is a traditional category found in every classification. We do not make modifications here.

Impact refers to how the result of an attack manifests in the system. The established categories of security properties are confidentiality, integrity, and availability. We use some of Lindqvist's [14] subcategories, and extend Du's [7] list. This allows us to further refine the outcome and to put more emphasis on the application compared with common protection objects.

With these design decisions, we tailor our approach to take the nature of arbitrary applications into account. This classification reminds one of the Model–View–Controller pattern. Attack location refers to input, Cause refers to processing, and Impact refers to output. We will elaborate on the three dimensions in the following subsections.

4.1. Location

As long as there is no input to a system (i.e., the application), there is no attack. Faults can only be introduced by crossing the system boundary with some input. Hence, all attacks happen at the application's perimeter. This perimeter partitions between the objects and connections controlled by the application on the one side and the environment on the other. The attack location is the part of the application perimeter where an input can be introduced. For instance, this could be a buffer expecting data from another program via a named pipe.

Location can be characterised by its nature (interactive, active, or passive) and the origin of data transmitted over the interface (user input, operating system, other programs, hardware, storage). Although not every venue can be used by malicious software directly, it is often possible for an attacker to use it as an attack path indirectly, depending on the overall system architecture. Examples for this include simulation of user input, extension of operating system functions by user mode programs etc.

Table 4 Location (input)

Nature of communication	Origin of data		Example
Interactive	User input		• Simulated local user input • Simulated remote user input
Active (Initiated by application, Polling)	Operating system		• Invoking API function/system call
	Other application		• Inter-process communication • Exerting a network service • Executing other process
	Hardware		• Smart Card • Card Terminal • Sensor • Device Driver
	Storage	Configuration data	• Stored in file • Stored in registry • Stored in network directory • Stored in external token • Stored in secure OS storage area
		File	• Data file • Data containing active content • Run-time library containing code
Passive (Initiated from outside the application, Events)	Operating system		• Call-back handler • System event notification
	Other application		• Inter-process communication • Service used over a network • Execution initiated by another process • Automation interface
	Hardware		• Interrupt handler

4.2. Cause

The type of invalidated assumption or programming flaw is "the" traditional category in vulnerability classifications since the RISOS study [1]. Different authors introduce subcategories or group some of the classes together. We stick with Du's [7] modification of Landwehr's [13] scheme.

Table 5 Cause (exploitable logic in processing)

Design/Implementation/ Configuration error	Example
Validation error	• Input validation • Origin validation • Target validation
Authentication error	• Protected operation invokable by unauthorised agent
Serialization/aliasing error	• Time-of-check-to-time-of-use flaw • Same name for different objects
Boundary checking error	• Buffer overflow
Domain error	• Access to implementation details
Weak or incorrect design error	• Weak encryption algorithm
Other exploitable logic error	–

4.3. Impact

Impact is a failure according to Jonsson's [11] terminology. A failure is the event at which deviation first occurs between the service delivered by the system and the expected service, as defined in the system specification. The classes shown here are based on Du [7], adding "Modified execution path", with subclasses borrowed from Lindqvist [14].

Table 6 Impact (output)

Manifestation of security breach		Example
Execution of code		• Executing arbitrary code
Modified execution path (access to application functionality)		• Execution of privileged code • Change of internal state
Change of resources	Selective	• Targeted modification
	Unselective	• Vandalism
Non-modifying access to resources		• Disclosure of Information
Denial of service	Selective	• Interrupting execution for specific user
	Unselective	• Barring application start-up

5. Conclusions

We have presented a classification for malicious software attacks. In contrast to existing classifications, we strongly focus on applications instead of operating systems. Attacks are classified by the location of flaw introduction (location), the exploitable logic causing the vulnerability (cause), and the manifestation of security property violation (impact).

Applications take over increasingly more of the business processes and increasingly define their own security demands. Hence, attacking applications will become more rewarding from an attacker's perspective. With our classification, software developers and evaluators can direct their testing efforts to reduce exposure and severity of vulnerabilities. Our scheme helps in searching for vulnerabilities in a systematic and automatic procedure, e.g., with vulnerability scanners. This classification is also a sound basis to develop a security metric, i.e., here, to quantitatively evaluate the robustness of an application. It could as well be used when assessing the effectiveness of countermeasures.

We intend to examine vulnerability scanners for applications against this classification. This could show the areas in which scanners help in an automatised search for program vulnerabilities with respect to malicious software.

6. References

[1] Abbott, R., Chin, J., Donnelley, J., Konigsford, W., Tokubo, S. and Webb, D. (1976). *Security Analysis and Enhancements of Computer Operating Systems.* Technical Report NBSIR 76-1041, ICET, National Bureau of Standards. Presented and reference given in [5] pp. 662-665.

[2] Aslam, T., Krsul, I. and Spafford, E.H. (1996). 'Use of A Taxonomy of Security Faults'. *Proceedings of 19th National Information Systems Security Conference.* Pp. 551-560.

[3] Bisbey II, R. and Hollingworth, D. (1978). *Protection Analysis: Final Report.* Technical Report ISI/SR-78-13, Information Sciences Institute, University of Southern California. Presented and reference given in [5] pp. 665-670.

[4] Bishop, M. and Bailey, D. (1996). *A Critical Analysis of Vulnerability Taxonomies.* Technical Report CSE-96-11, Department of Computer Science, University of California at Davis.

[5] Bishop, M. (2003). Computer security: art and science.

[6] Cohen, F. (1985). *Computer Viruses.* PhD thesis, University of Southern California.

[7] Du, W. and Mathur, A.P. (1998). 'Categorization of Software Errors that led to Security Breaches'. *Proceedings of 21st National Information Systems Security Conference.* Pp. 392-407.

[8] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns.*

[9] Hogan, C.B. (1988). 'Protection Imperfect: The Security of Some Computing Environments'. *Operating Systems Review 22.3(1988)*:7-27.

[10] Jiwnani, K. and Zelkowitz, M. (2002). 'Maintaining Software with a Security Perspective'. *Proceedings of 18th International Conference on Software Maintenance.* Pp. 194-203.

[11] Jonsson, E., Strömberg, L., and Lindskog, S. (1999). 'On the Functional Relation Between Security and Dependability Impairments'. *Proceedings of New Security Paradigms Workshop.* Pp. 104-111.

[12] Krsul, I.V. (1998). *Software Vulnerability Analysis.* PhD thesis, Purdue University.

[13] Landwehr, C.E., Bull, A.R., McDermott, J.P. and Choi, W.S. (1994). 'A Taxonomy of Computer Program Security Flaws'. *ACM Computing Surveys 26.3(1994)*:211-254.

[14] Lindqvist, U. and Jonsson, E. (1997). 'How to Systematically Classify Computer Security Intrusions'. *Proceedings of 1997 IEEE Symposium on Security and Privacy.* Pp. 154-163.

[15] Neumann, P.G. and Parker, D.B. (1989). 'A summary of computer misuse techniques'. *Proceedings of the 12th National Computer Security Conference.* Pp. 396-407. Reference given in [14].

[16] Piessens, F., De Decker, B. and De Win, B. (2001). 'Developing Secure Software. A survey and classification of common software vulnerabilities'. *Proceedings of 4th International Conference on Integrity and Internal Control in Information Systems.* Pp. 27-40.

[17] Saltzer, J.H. and Schroeder, M.D. (1975). 'The protection of information in computer systems'. *Proceedings of the IEEE 63.9(1975)*:1278-1308.
http://web.mit.edu/Saltzer/www/publications/protection/