Categorization of Software Errors that led to Security Breaches

Wenliang Du
Email: duw@cs.purdue.edu, Tel: (765)494-9313
COAST Laboratory
1398 Department of Computer Sciences
Purdue University, W. Lafayette, IN 47907,USA
Aditya P. Mathur
Email: apm@cs.purdue.edu, Tel: (765)494-7823, Fax: (765)494-0739
COAST Laboratory and Software Engineering Research Center
1398 Department of Computer Sciences
Purdue University, W. Lafayette, IN 47907, USA

Abstract

A set of errors known to have led to security breaches in computer systems was analyzed. The analysis led to a categorization of these errors. After examining several proposed schemes for the categorization of software errors a new scheme was developed and used. This scheme classifies errors by their cause, the nature of their impact, and the type of change, or fix, made to remove the error. The errors considered in this work are found in a database maintained by the COAST laboratory. The categorization is the first step in the investigation of the effectiveness of various measures of code coverage in revealing software errors that might lead to security breaches.

Keywords: Security, security flaw, fault classfication.

1 Introduction

We report the outcome of an effort to categorize errors in software that are known to have led to security breaches. The set of errors used in this study came from a database of errors developed in the COAST laboratory [10]. Several existing schemes for the categorization of software errors were evaluated for possible use in our effort. However, none was found fully suitable. This led to the development of a new scheme which is reported here.

Our categorization of security errors was motivated by (a) a desire to evaluate the effectiveness of traditional code-based coverage criteria in revealing software errors that lead to security breaches and (b) a perceived need to develop a tool that assists software developers and testers in the assessment of tests of distributed software aimed at detecting and understanding the effects of possible security flaws. We plan to use the categorization reported here in the design and conduct of experiments to fulfill (a). We hope that results from such experiments will lead to information that might be useful in satisfying (b). The

traditional code-based adequacy criteria we are concerned with includes all control flow, data flow, and mutation based criteria considered by Wong [15]. The distributed systems we are concerned with are the ones developed using CORBA as the interface standard between distributed objects and Java as the programming language.

The remainder of this report is organized into three sections. Section 2 briefly discusses current methods for detecting security flaws and their limitations. Section 3 critically examines several existing schemes for the categorization of errors in software. Section 4 presents a new scheme for this purpose and reports the outcome of our categorization effort. Section 6 summarizes this report and discusses how we propose to use the outcome of our categorization in the design of experiments.

2 Detecting security errors

Reports of security breaches due to errors in software are becoming an increasingly common. This has resulted in new security related concerns among software developers and users regarding their product. All stages of software development are effected by the desire to make the product secure and be not vulnerable to malicious intentions of some users. Our work is concerned with the testing of software with the goal of detecting errors that might lead to security breaches. We refer to such errors as security errors or security flaws.

Traditional methods for detecting security errors include penetration analysis, and formal verification of security kernels [11, 12]. Penetration analysis relies on known security flaws in software systems. These flaws are generalized and a team of individuals is given the responsibility of *penetrating* the system using this knowledge. Formal methods use a mathematical description of the security requirements and that of the system that implements the requirements. The goal of these methods is to formally show that indeed the requirements are met by the system.

One weakness of penetration analysis is that it requires one to either know or be able to postulate the nature of flaws that might exist in a system. This may not be a difficult task for systems that such as Operating systems and browsers that implement well known functionality. However, for systems that have one-of-a-kind functionality a lack of experience might render this task difficult. Further, the effectiveness of penetration analysis is as good as that of the team that performs the analysis. A lack of an objective criterion to measure the "goodness" of penetration analysis leads to uncertainty in the reliability of the system for which penetration analysis did not reveal any security flaws.

Attractive due to the precision they provide, formal methods suffer from the inherent difficulty in specifying the requirements, the system, and then applying the process of checking the requirements specification against the system specification. We view penetration analysis and formal methods as adjuncts to the testing of a system for security flaws using traditional methods of software testing.

Several criteria have been proposed to evaluate the adequacy of tests of software. These include criteria that are based on program mutation, control structure, functions, data flows, etc. The cost and utility of applying these criteria differs widely and has not been assessed when applied for detecting security errors. Our primary long-term goal is to evaluate the effectiveness of these adequacy criteria in the detection of security errors. More specifically, we propose to investigate the following questions in this study:

- How do the number and types of security errors that can be found in testing vary over different adequacy criteria ?
- Is there any difference in the effectiveness of the adequacy criteria evaluated over all types of software errors and against only security errors ?
- Can the quality and amount of penetration analysis be improved with the use of adequacy criteria ?

We plan to design several experiments to achieve these goals. The experiments will be conducted by injecting security errors into sample programs and applying various testing techniques to determine how often the techniques reveal the errors.

Before experiments start, there are several things we need to do first: The first thing is to collect security errors, this will provide us with a resource set of appropriate size which we can choose security errors from; the second thing is to categorize security errors. The categorization provides us with the criteria on deciding what errors should be chosen and injected to sample code.

Several people in COAST lab have contributed to the collection of security errors in the last few years, a vulnerability database has been established by Ivan Krusl [10]. It contains abundant security errors for us to conduct the experiment. However, the database does not provide a suitable categorization to help us select security errors in the experiment (we will discuss it later) and conduct the experiment. That's why we need to devise a new categorization scheme that can not only classify security errors, but also provide helpful guidelines to our goal.

Our current focus is on the categorization scheme, we investigated all security errors (about 50 so far, the number is still increasing) stored in vulnerability database, and conducted a deep study on the software error classification and security error classification, and finally we have come up with a new categorization scheme that we are going to use in our experiments.

In addition, the new error categorization scheme will later be used as baseline for evaluating the reliability of a system.

In this paper, we will discuss our new categorization scheme. This new scheme comes up from investigating of about 150 security flaws, one third of them from Krusl's vulnerability database, the second third from Landwehr's paper [6], the other part of them from unorganized data from different resources (this part has not been added to the database yet).

3 Past work

3.1 Characteristics of error classification

It has been suggested that a taxonomy should have classification categories with the following characteristics [1]:

1. The categories be mutually exclusive so that classification of an error is possible into only one category.

2. The categories be exhaustive so that taken together the categories include all possibilities.

3. The categories be unambiguous so that classification is not dependent upon interpretation by multiple classifiers. This property also leads to repeatability of the classification scheme which means that so that repeated applications of the scheme to the same set of errors result in the same classification regardless of the individual(s) involved in the process.

Several researchers have conducted the classification of security errors. In this section we discuss the strengths and weaknesses of the existing classification schemes in.

3.2 Security error classification

Landwehr's scheme

Landwehr [6] proposed a taxonomy of flaws found in different operating systems. The objective of this taxonomy is to provide an understandable record of security flaws that have occurred. It is based on three dimensions: *Genesis*, *Time of Introduction*, and *Location*. Genesis refers to how a security flaw finds its way into a program. Time of Introduction is the point in the software's life cycle where the error was introduced. Location is that part of the operating system or hardware where the error lies.

Landwehr's scheme describes the security flaws more accurately than schemes that use only one dimension. The genesis provides a basis for choosing different strategies to avoid, detect, or compensate for security flaws. One weakness of this scheme is in that the categorization by genesis is ambiguous. In Landwehr's own description of each categorization, the validation errors include the errors of inadequate identification/authentication and boundary condition errors. Another weakness is in that it contains a category called "other exploitable logic errors" into which are placed all errors not classified into any other category. We found that some errors classified into this category by Lendwehr do possess distinguishing features that could be used to categorize them differently.

The time of introduction categorizes errors according to the phase of the system life cycle in which they were introduced. This attribute of an error is important in that it helps in an understanding of weaknesses in the software development process. For example, when one finds that too many security errors are being introduced during the design phase, one may be wise to consider means to improve the design process and the validation of the resulting designs. However, this category is not directly useful in our work which deals with the evaluation of testing strategies applied to the code and not to designs. It is for this reason that we decided not to use this category.

The location of an error classifies errors according to where in the system it is introduced or found. This dimension is primarily related to security flaws in an operating system. Our research does not focus on any specific type of software system and hence this category is again not found to be of direct use in our work.

Aslam's scheme

Aslam [2] has proposed a taxonomy with the objective of providing a basis for data organization in a vulnerability database that facilitates different queries to be performed on the stored data. It considers classifying coding errors into two categories, namely synchronization and condition validation. It attributes the cause of all non-synchronization security errors to the improper evaluation of condition. This viewpoint appears a narrow viewpoint in that some errors are not caused by the improper evaluation of any condition. Stated differently, we may correct the error without even changing any condition in the program. Figure 3 illustrates this point. Further, Aslam's taxonomy was derived from security errors found in the UNIX operating system and is not exhaustive. For example, the use of a relatively weak algorithm in the encryption of a password would not be categorizable according to Aslam's scheme.

Bishop's scheme

Matt Bishop [4] proposed a six-axis taxonomy for software vulnerability. Each vulnerability is classified on each of the six axes. The axes are: the *nature* of the flaw, the *time of introduction*, the *exploitation domain* of the vulnerability, the *effect domain*, the *minimum number* of components needed to exploit the vulnerability and the *source* of the identification of the vulnerability. The objective of this taxonomy is to describe the vulnerabilities in a form useful for the intrusion detection mechanisms. This goal is different from ours. Except for the the nature of the flaw, the other axes are not directly relevant to our goal.

Other works, such as the Protection Analysis Project [14], the RISOS Project [8], and the Flaw Hypothesis Methodology [11] also proposed their own classification scheme. All these schemes share the weaknesses pointed out above in our discussion of Aslam's and Lendwehr's schemes.

3.3 Classification of software errors

In addition to the classification schemes discussed above, several schemes for the categorization of software errors have been proposed by researchers. These categorization schemes have been published in non-security community. In this section we review these schemes and discuss their strengths and weaknesses.

Orthogonal defect classification

Chillarege and Bhandari proposed an orthogonal defect classification scheme [7]. The objective of this scheme is to provide fast and effective feedback to developers by using in-process measurement, i.e., the defect distribution can be used to measure the development process and highlight that part of the process that needs attention. The defect types are general enough to be applicable to any phase of the software development process. The classification they choose actually provides the capability to reflect the characteristic of defect distribution change over development process. As our study focuses on the implementation and testing phase, and not the entire software development cycle, the ODC is overly complex for our use. Further, it does not provide sufficient data to help determine the exact nature of the error in the code.

Goodenough and Gerhart's scheme

The goal of Goodenough and Gerhart's scheme [9] is to give an insight into test reliability. Their classification scheme is used to evaluate the reliability of various test data selection criterion. Many of data selection criterion are based on program's internal structure, such as path testing, branch testing, and statement testing. The paper attempts to show that testing based solely on a knowledge of a program's internal structure can not lead to reliable tests. To achieve this goal, the classification should reflect program's internal structure, which is the control flow of the program.

One problem with this scheme is in its ambiguity. As an example of this ambiguity, consider the incorrect statement `if (A) ...` which should actually be `if (A.AND.B) ...`. Clearly there is a failure on the part of the programmer to test for condition `B`. Thus, this fault is of type *missing control flow path*. However, this is also of type *inappropriate path selection* because the condition has been expressed incorrectly. Furthermore, this classification scheme is too narrow to classify security errors. We found there are some security errors that do not fit into any of the proposed categories. For example, security errors caused by race condition cannot be classified according to their scheme.

Ostrand and Weyuker's scheme

Ostrand and Weyuker's scheme [13] was designed with the goal of evaluating the effectiveness of current and proposed software development, validation, and maintenance techniques. The key feature of this scheme is that it attempts to identify the fault characteristics in several distinct areas. Within each area, one of several possible values can be chosen to describe the fault. As we have discussed before, the more accurately we categorize faults, the more easy it is to avoid a strong bias in selecting security errors. We decided not to use this scheme because the attributes used do not reflect key features related to security errors.

Basili and Perricone's scheme

Basili and Perricone [3] has a goal to analyze the relationship between the frequency and distribution of errors during software development. The classification they use is ambiguous, as mentioned by the authors. This might cause different analysts to interpret the categories differently.

4 Proposed scheme for the classification of security errors

4.1 Characteristics of our scheme

The primary goal of this research is to devise a usable and practical scheme for categorizing security errors. Such a scheme is essential for understanding and controlling the factors which affect software security. Therefore, the categorization should reflect key features of security errors. In addition, this scheme will serve as a basis for evaluating the effectiveness of software testing techniques for revealing errors that lead to security breaches.

We propose using a multiple attribute approach similar to the one proposed by Ostrand and Weyuker [13]. Another approach to classification is to place a given error into a single

category which most closely matches some feature of the error. By doing so, we risk losing information. Classifying an error into a single category leads to abstraction at the risk of throwing away several features of the error while retaining only the feature represented by the category into which the error is classified. This is not desirable from our point of view.

Consider, for example, using a single-category classification approach which places an error into one of the four categories A, B, C, and D. Now, suppose we find that testing technique T could help detect, on an average, 95% of security errors. However, the reason for this high percentage might be that most of the security error we have selected in our experiment might have attribute E which we decided to abstract away as it is implicitly included in categories A to D. Thus, classification to one category might to results from our experiments that are not trustworthy as they are biased towards the categories we have selected and ignore the categories that have been abstracted away.

Keeping more information in categorization is a way to avoid the bias. Towards this end, we decided not to assign an error to a single category. Instead, we attempt to identify the error's characteristics in several distinct areas. Within each area one of several possible values can be chosen to describe the error. Together all areas provide more detailed information about the security errors than would be available with a single category scheme.

From an operational viewpoint, a security error is due to some reason, has an impact that violates a security policy, and may be fixed eventually. The sequence is shown in Figure 1 This operational sequence will be expanded in this section to provide a taxonomy that will then be used to classify security errors.

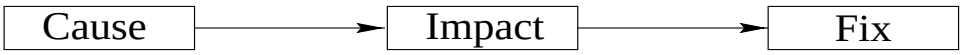

Figure 1: Life cycle of a security error.

4.2 Classification scheme

By cause

Landwehr's categorization by genesis describes the cause of the security errors in more detail than others. As we focus on *inadvertent* errors, we do not use the *intentional* part of Landwehr's taxonomy. Also, we have modified the definition of some categories to get rid of its ambiguity.

1. *Validation error:* Validation is used when a program interacts with the environment. There are three items that need validation: *input, origin, and target.* An error occurs when the assumptions about at least one of these three items does not conform to the reality. *Input validation* ensures that the input is what is expected. It includes the number, type and format of each input field. It should forbid any bad input to enter the software system. *Origin validation* ensures that the origin is actually the one it claims to be. It includes checking the identity of the origin. *Target validation* ensures that the information goes to the place it is supposed to. This includes checking the identity of the target so that protected information does not go to an untrusted target.

2. *Authentication error* is one that permits a protected operation to be invoked without sufficient checking of the authority of the invoking agent.

3. *Serialization/aliasing error:* A serialization flaw permits the asynchronous behavior of different system components to be exploited to cause a security violation. Many time-of-check-to-time-of-use (TOCTTOU) flaws fall in to this category. We also include *aliasing flaws* in this category. These flaws occur when two names for the same object can cause its contents to change unexpectedly and, consequently, invalidate checks already applied to it.

4. *Boundary checking error* is one that is caused by the failure to check boundary to ensure constraints. Not checking against excessive values associated with table size, file allocation, or other resource consumption leads to boundary checking error. Buffer overflow is a result of a boundary checking error.

5. *Domain error* occurs when the intended boundaries between protection environments have holes. This causes information to leak out implicitly.

6. *Weak or incorrect design error* occurs when the error is traced to the design phase. For example, weak encryption algorithm falls in to this category.

7. *Other exploitable logic error* is one that does not fall into the above categories.

By direct impact

1. *Execution of code* is any unauthorized execution of code on a target computer.

2. *Change of target resource* is any unauthorized alteration of resource on a target computer, the resource include files, environment, processes and etc.

3. *Access the target resource* is any unauthorized access of resource on a target computer. The access does not include alteration and the resource is the same as above.

4. *Denial of service* is any action that degrades or blocks the computer or network.

We prioritize the above four categories according to their severity. The *execution of code* has the highest priority and the *denial of service* has the least priority. If a security error causes more than two kinds of impact, we categorize it according to the highest priority, i.e., according to the most serious impact it causes.

By fix

Demillo and Mathur's fault classification scheme [5] is the best candidate for categorizing security faults by fix because it is unambiguous and capable of automation. More importantly as it is based on the original source code and the code after correction, it fits well for categorization by fix.

The scheme classifies an error into one of is the following categories.

1. *Spurious entity:* A fault whose correction requires the removal of its characteristic substring falls under this category.

2. *Missing entity:* A fault whose correction requires the insertion of a syntactic entity into the incorrect program falls under this category. A missing entity could be, for example, a sequence of statements, a single statement, an expression, or a unary operator. These four syntactic entities form the sub-categories of the missing entity category.

3. *Misplaced entity:* If the correction of a fault requires a change in its position within the code, it is classified under this category.

4. *Incorrect entity:* When a fault cannot be classified as missing entity, misplaced entity, or spurious entity, then it is classified into this category.

In summary, our scheme is presented in Figure 2.

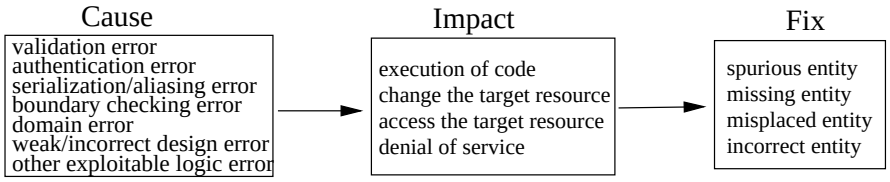

Figure 2: A scheme for the categorization of security errors.

5 Sample categorization

To illustrate the categorization scheme, we give two examples of problem descriptions and show the resulting classifications. Both examples are from Vulnerability Database maintained by Ivan Krusl in the COAST laboratory [10].

Problem 1: Security hole in guestbook script for web servers using SSI.

Description of problem symptoms: Guestbook applications allow a person browsing a web site to "sign" an electronic guestbook and leave an appropriate message. All versions of this program have a vulnerability that under certain conditions allows a remote user to execute arbitrary commands on the server as the `user id` of the `httpd` daemon. These conditions are: 1) the server allow Server Side Includes (SSI) on the directory in which the guestbook is located, 2) the guestbook application allows the remote user to write HTML tags into the Comment field of the guestbook, and 3) the guestbook application does not filter appropriate HTML tags.

Description of fix: See Figure 3.

Problem 2: `bind`() allows binding to sockets in use.

Description of problem symptoms: On most systems, a combination of setting the SO_REUSEADDR socket option and a call to bind() allows any process to bind to a port to which a previous process has bound width INADDR_ANY. This allows a user to bind to the specific address of a server bound to INADDR_ANY on an unprivileged port and steal its udp packets/tcp connection.

Description of fix: See Figure 4 .

Table 1 shows the three fault categorizations that resulted from these two problem reports.

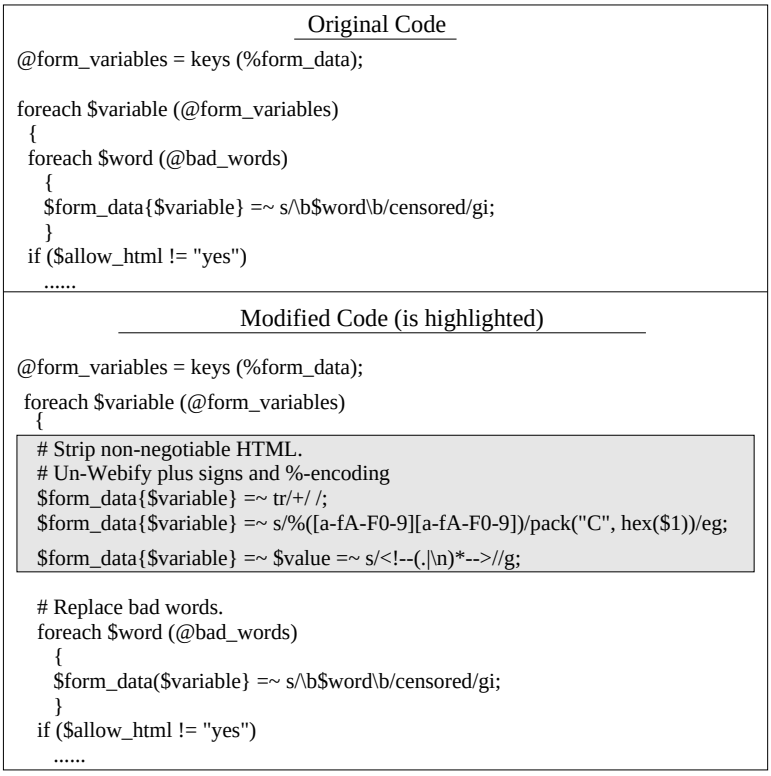

Figure 3: Original and modified code for problem 1.

Table 1: Categorization of sample security errors in VDD.

Error ID	Cause	Impact	Fix
Problem 1	validation error	execution of code	missing entity
Problem 2	domain error	access target resource	missing entity

6 Summary and future work

Work reported here focuses on reviewing the existing schemes for the categorization of security errors and other of general software errors. Based on this review we have devised a new scheme that suits our goal. proposed an security error classification scheme in this paper. Our classification scheme allows distinct categorization of each security error according to the specified criteria from three points of view. These three viewpoints not only reflect the feature of security error but also provide the basis for our future work which is to experimentally determine the effectiveness of testing techniques in revealing security errors.

In the experiment, we plan to inject security errors into some sample programs and apply various testing techniques to determine how often the techniques reveal the errors. The data we collect from this experiment will help us estimate the effectiveness.

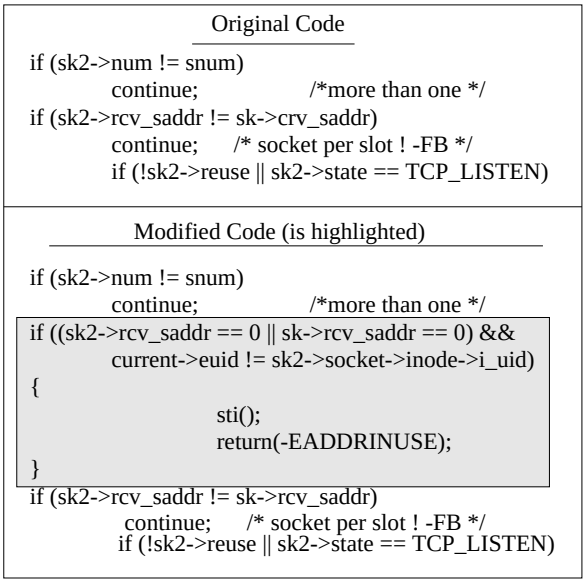

Figure 4: Original and modified code for problem 2.

References

[1] E. G. Amoroso. *Fundamentals of Computer Security Technology.* Prentice-Hall PTR, Upper Saddle River, NJ, 1994.

[2] T. Aslam. A taxonomy of security faults in the unix operation system. Master's thesis, Purdue University, August 1995.

[3] V. R. Basili and B. T. Perricone. Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27(1):42–52, January 1984.

[4] M. Bishop. A taxonomy of unix system and network vulnerabilities. Technical Report CSE-95-10, Department of Computer Science, University of California at Davis, May 1995.

[5] R. A. Demillo and A. P. Mathur. A grammar based fault classification scheme and its application to the classification of the errors of TEX. Technical Report SERC-TR-165-P, Purdue University, 1995.

[6] C. E. Landwehr, et al. A taxonomy of computer program security flaws. *ACM Computing Surveys*, 26(3), September 1994.

[7] R. Chillarege, et al. Orthogonal defect classification – a concept for in-process measurements. *IEEE Transactions on Software Engineering*, 18(11):943–956, November 1992.

[8] R. P. Abbott, et al. Security analysis and enhancements of computer operating systems. Technical Report NBSIR 76-1041, Institute for Computer Science and Technology, National Bureau of Standards, 1976.

[9] J. B. Goodenough and S. L. Gerhart. Toward a theory of test data selection. *IEEE Transactions on Software Engineering*, SE-1(2):156–173, June 1975.

[10] I. Krsul. Computer vulnerability analysis thesis proposal. Technical Report CSD-TR-97-026, Computer Science Department, Purdue University, 1997.

[11] R. R. Linde. Operating system penetration. In *AFIPS National Computer Conference*, pages pp. 361–368, 1975.

[12] E. J. McCauley and P. J. Drongowski. The design of a secure operating system. In *National Computer Conference*, 1979.

[13] T. J. Ostrand and E. J. Weyuker. Collecting and categorizing software error data in an industrial environment. *The Journal of Systems and Software*, pages 289–300, 1984.

[14] R. Bibsey, G. Popek and J. Carlstead. Inconsistency of single data value over time. Technical report, Information Sciences Institute, University of Southern California, December 1975.

[15] W. E. Wong and A. P. Mathur. Effectiveness of mutation and data flow testing. *Software Quality Journal*, 4:69–83, 1995.

Table 2: Categorization of security errors in VDD by their Cause, Impact and Fix. The appearance of a "?" in the column marked Fix indicates that sufficient code was not available to categorize the error.

Error ID	Cause	Impact	Fix
rpcbind_warm	authentication error	change target resource	?
freebsd_setlocale	boundary checking error	execution of code	missing
rlogin_term	boundary checking error	execution of code	?
solaris_getopt	boundary checking error	execution of code	?
talkd_dns	boundary checking error	execution of code	?
apache_cookie	boundary checking error	execution of code	missing
solaris_ps	serialization/aliasing error	execution of code	incorrect
socket_ioctl	domain error	denial of service	?
NIS_DNS	validation error	execution of code	?
xmcd_cddpath_overflow	boundary checking error	execution of code	?
xterm_vulnerability	serialization/aliasing error	execution of code	?
mgetty_sendfax	validation error	execution of code	?
rdist	boundary checking error	execution of code	?
dip	boundary checking error	execution of code	?
tcsh_back_tic	validation	access target resource	?
test_cgi	validation error	access target resource	?
java_acl_path	validation error	access target resource	?
java_abs_path	validation error	execution of code	?
sendmail_875b	boundary checking error	execution of code	?
crontab_tempfile_guess	authentication error	change target resource	?
netscape_servkey_guess	design error	execution of code	incorrect
telnet_dynlib	authentication error	execution of code	?
java_dns	validation	access target resource	?
sendmail_MIME_overrun	boundary checking error	execution of code	?
aix_gethostbyname	bounary checking error	execution of code	?
dns_bind	boundary checking error	execution of code	?
ie_ntlm_auth	authentication error	access target resource	?

Table 3: Categorization of security errors in VDD by Cause, Impact, and Fix.

Error ID	Cause	Impact	Fix
winnt_rpc_vulnerability	validation error	denial of service	?
ibm_nls_environment	boundar checking error	execution of code	?
innd_shell_escape	validation error	execution of code	?
inn_ucbmail_shellEscape	validation error	execution of code	?
imap_buffer_overflow	boundary checking error	execution of code	?
sysinstall_freebsd	authentication error	access target resource	incorrect
lpd_buffer_overflow	boundary checking error	execution of code	incorrect
shockwave-security-hole	authentication error	access target resource	?
winNT_sampermission	design error	execution of code	?
remote_NT_password_cracking	authentication error	access target resource	?
bind_packet_stealing	domain error	access target resource	missing
winNT-longfilenames-subst-bug	boundary checking error	denial of service	?
selena_sl_guest	validation error	execution of code	missing
nat_lang_buf_overflow	boundary checking error	execution of code	?