COAST Vulnerability Database Reference Guide - Draft Version

Ivan Krsul*
Technical Report 97-12†
COAST Laboratory
Purdue University
West Lafayette, IN 47907–1398
krsul@cs.purdue.edu

February 27, 1998

1 Introduction

The vulnerability collection at the Computer Operations, Audit, and Security Technology (COAST) lab consists of a structured vulnerability database; a collection of several thousand vulnerability related files, mailings and articles; and a directory of vulnerability related tools that include exploit scripts, hacker tools, analysis tools, etc.

This document describes the vulnerability collection, the vulnerability database program, and the vulnerability database WWW interface. It is *strongly encouraged* that you read through the entire document before you use the collection or the database for the first time.

There is a mailing list called `vdb-info@cs.purdue.edu` where all changes to the structure of the database (classifiers, fields, etc.) should be announced. General announcements regarding the database will also be sent to that mailing list.

If you are accessing the collection at the COAST lab, the information should only be accessed by logging into the machine landover and changing your directory to
tt /homes/krsul/vdbase. The information in that directory tree must never be copied to another directory or another machine.

2 Terms for Access to the Database

The information contained in the Vulnerability collection at the COAST laboratory is provided to students and COAST sponsors and collaborators on a case by case basis. To request copies or access to the collection please contact Eugene Spafford at spaf@cs.purdue.edu.

Please read the rest of this document before requesting access to the vulnerability collection.

All the information contained in the collection should be considered sensitive because it contains material that can be misused and has the potential of causing damages to people and/or computer equipment. As a student or researcher accessing this collection, you must take appropriate precautions:

- The information in the collection has been collected from publicly accessible sources (such as mailing lists, newsgroups, etc.). You should **never** enter into the database or collection proprietary information, trade secrets, information for which a non-disclosure agreement has been signed, etc. Help keep our lawyer fees low.

*Portions of this work were supported by sponsors of the COAST Laboratory

†This technical report is updated constantly. The tables, decision trees, procedures, etc., are under constant revision. Hence, the document will remain an official COAST Laboratory draft. The latest version of this technical support is available in `ftp://coast.cs.purdue.edu/pub/COAST/papers/krsulvdbref.{pdf,ps}`. The document can be cited if the date of the draft is clearly indicated.

- The information should never be posted to any mailing lists, newsgroup, or placed on `ftp`, `gopher` or `http` servers. As a researcher you are being given material that is potentially harmful and you must exercise some social responsibility. If you think someone else should have this information **do not give it to them** without contacting Spaf at spaf@cs.purdue.edu.
- The information contained in this collection should **never** be used to crack, hack, or break into machines without the explicit authorization (and we do mean ***explicit***) of the administrator and owner of that machine.
- When in doubt, do the safe thing. Contact Spaf or Ivan Krsul at spaf@cs.purdue.edu or krsul@cs.purdue.edu or call us at (765) 494-9313.

If you are accessing the collection at the COAST lab, the information should only be modified by logging into the machine `landover.cs.purdue.edu` and changing your directory to the vulnerability database directory. The information in that directory tree must never be copied to another directory or another machine.

If you are a student at the COAST lab or the computer science department, and are being given access to the collection, you are expected to contribute to it by adding records to the structured database. There is a Java based interface that you must use and, while modifying information, you must be careful to add your name to the "modifications" field so that there can be a record of what changes you have made to the collection. If you do not contribute to entering and polishing the structured database, your access rights to the database may be removed.

3 Structure of the Collection and Database

The directories relevant to the collection are:

The collection: $\overline{VDBCOLL}$ = */homes/krsul/vdbase*

The structured database: $\overline{VDB}$ = $\overline{VDBCOLL}$ */vdb/*

Index file for vulnerability database: $\overline{VINDEX}$ = $\overline{VDBCOLL}$ */vdb/Vulnerabilities*

Directory for classifiers: $\overline{CLASSD}$ = $\overline{VDBCOLL}$ */vdb_classifiers*

File that contains the schema definition for the database: $\overline{VDBSCH}$ = $\overline{VDBCOLL}$ */vdb_field_list*

Directory where the Java Graphical User Interface (GUI) is: $\overline{JAVAGUI}$ = $\overline{VDBCOLL}$ */vdb/javaprogs*

Directory where the source code for the Java GUI is: $\overline{JAVAGUIDEV}$ = $\overline{VDBCOLL}$ */vdb/javaprogs_dev*

Directory where all the perl based programs are: $\overline{PERLTOOLS}$ = $\overline{VDBCOLL}$ */perl*

Directory for the MIME included files: $\overline{MIMEINCLUDES}$ = $\overline{VDBCOLL}$ */vdb_includes*

Directory for HTTP server and `cgi-scripts`: $\overline{HTTPD}$ = $\overline{VDBCOLL}$ */httpd*

We have a collection of vulnerability-related files, tools, exploit scripts, etc. that is in $\overline{VDBCOLL}$*/related_stuff* . There is no documentation on this part of the collection. You are on your own. Sorry.

The structured database is a collection or records that have a number of fields defined in the file $\overline{VDBSCH}$. The fields can be of various types, including text, list, choice list, matrix classifier and hierarchical classifiers. All fields where you can type text are considered to be multi-valued (i.e. you can add as many lines as you wish) and all the fields where you can type text support the inclusion of arbitrary Multipurpose Internet Mail Extensions (MIME) parts. See section 10 for a detailed description of the format of this file.

The fields of records are stored in individual files, using the file system as a database manager of sorts. The database directory has a subdirectory for each field in the schema, and in those directories we have a directory for each database record ID that has the field defined, and in that directory a file (called `V`) that contains the value for the field. If the field has a confidence rating then the value for that rating will be stored in a file called `R` in the same directory.

If the field is a text field and it has an included MIME part, this part will be stored in the $\overline{MIMEINCLUDES}$ directory in a subdirectory that has the name of the record ID. This allows for multiple fields in the same record to point to a single MIME file that doe not need to be replicated.

So for example, if the record `sunsmailbug` has information on the field description, and this field has a confidence rating, and a MIME part called `MIME123456` then there will be a file called `V`, a file called `R`, and a file called `MIME123456` as shown in figure 1.

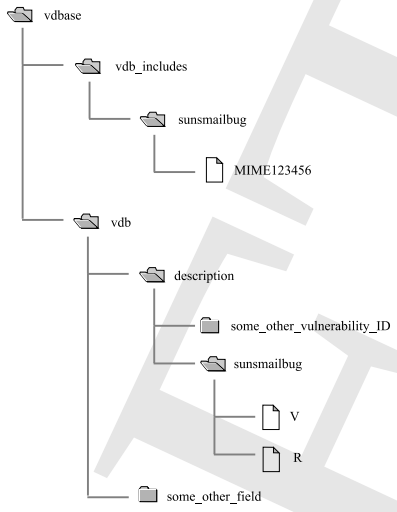


Figure 1: An example of part of the directory structure for the database for a record `sunsmailbug` that has information on the field description, a confidence rating for this field, and a MIME part.

4 Record Index

The list of records defined in the database is in the file *VINDEX* . Each non-comment line has the ID of a record.

5 The WWW Interface

The preferred mechanism for accessing the database is the HTML-based WWW interface for the vulnerability database. This interface provides an easy to use browsing tool that displays records and provides comprehensive search mechanisms.

5.1 Using the WWW Interface

The WWW interface requires that you point your favorite WWW browser to `http://landover.cs.purdue.edu/cgi-bin/vd` Because this service is behind a firewall, you will need to be in a machine that is within the COAST internal network or a machine that is authorized to connect to the service. You will also need a user name and a password to access the database. Contact Ivan Krsul or Eugene Spafford if you do not have a user name and you would like to access the database.

5.2 The Gory Details About the Internals of the Interface

As shown in figure 2, the WWW interface requires four components that must work together to display information. The database is a collection of several thousand files. The search server keeps a copy of the database cached in memory and waits for connections from the `cgi-bin` script. The `cgi-bin` script receives requests from the user via a WWW browser (for example, Netscape), translates the request into something that the search server can understand, and relays the reply from the `cgi-bin` script to the browser.

This complicated setup is necessary because we need a quick response to queries from the browser and the `cgi-bin` script cannot scan through the files in the database fast enough.

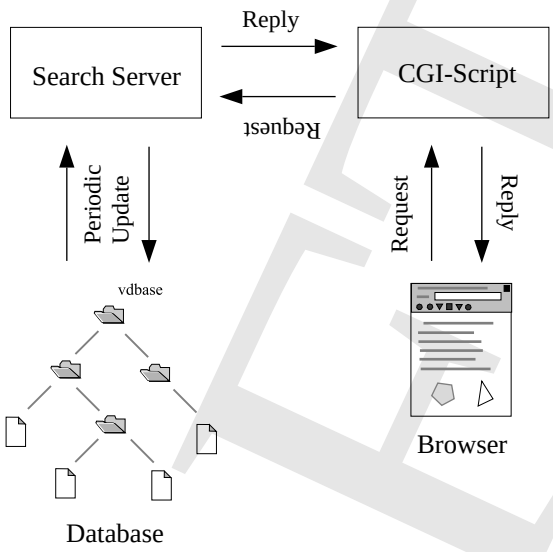


Figure 2: In the WWW interface, four components work together,

The search server is also described in section 6.2 and has two functions. It serves as an intermidiary between the `cgi-bin` script and the database, and it allows UNIX users to quickly search the database by issuing a command in a shell.

5.2.1 `cgi-bin` Script Details

The `cgi-bin` script is located in *HTTP* /cgi-bin/vdb. This is a Perl script that waits for commands from a browser, contacts the search server for requests and returns the data provided by the search server to the browser. The script is responsible for providing the HTML code for the headers and footers of every page, but is not responsible for formatting the data provided by the search server.

5.2.2 Search Server Details (With Respect to the WWW Interface)

The search server waits on a well defined port (hard coded into the server) for connections from clients that are coming from the machine where the `cgi-bin` is located. For security reasons only users in `landover.cs.purdue.edu` are allowed to connect to the search server. **Note:** *The assumption is that DNS cannot be hijacked within the lab because we are behind a firewall. Hence, the authorization check is performed by comparing the name of the machine connecting, as returned by a reverse DNS lookup, to the hard coded name.*

The search server responds to the following commands:

`SEARCH Search String` . The search server does a simple string match search on the database and prints in raw text the search results.

`HTMLSEARCH Search String` . The search server does a simple string match search on the database and prints in formatted HTML the search results.

`HTMLPERLSEARCH Perl Regular Expression` . The search server does a perl regular expression search on the database and prints in formatted HTML the search results. Note that special characters in the regular expression are escaped because they are a problem in the web server. Hence, the search server converts these characters back to normal before using the regular expression (with one notable exception: the back tick.).

`LIST Search String` . The search server lists, in raw text, all the records whose title match the string given, if any.

`HTMLLIST Search String` . The search server lists, formatted in HTML, all the records whose title match the string given, if any.

`DUMP Record_ID` . The search server provides a raw text dump of the entire record.

`HTMLDUMP Record_ID` . The search server provides an HTML dump of the entire record. In this dump all URIs and all references to MIME parts are converted to hyper-links.

`MIMEDECODE Record_ID MIME_Part_Name` . Given a record ID and the name of a MIME part, the search server fetches the MIME part, decodes it and returns the result with an associated MIME type header.

6 The Java Interface

A few words about the Java GUI: It's new and relatively large (8000+ lines of Java) so it is likely to have bugs and problems. Please notify us if there are any problems.

We encourage you to familiarize yourself with the source code of the interface and it's inner workings and, if necessary, to improve it as you see fit. The code is in the *JAVAGUIDEV* directory. The perl programs that are used as support are in the *PERLTOOLS* directory.

One *very important* piece of information that you should be aware of: The Java GUI implements record locking to allow multiple people access to the database at the same time. If the program crashes before it released the locks on the records you were editing (which happens when you save or when you exit), it is possible that the next time you use the program you may have to clean the locks by hand. The program will tell you how to do that.... but be sure that the lock that you clean by had is yours! It is possible that someone else may be editing that record. You can do that by checking the ownership of the lock-file it created.

6.1 Running the GUI Interface

There is a shell script that sets the Java classpath and runs the GUI interface. We recommend that to run the interface you create the following alias: "`alias vdbJava` *JAVAGUI/runvdbgui* ."

Please, don not run the database using another command! The script makes sure that your `umask` is set to 007 and hence the files created by the database will have the correct permissions. The script also makes sure that your classpath contains all the packages needed to run the system. If your `umask` is not 007 then it is possible that the files you create using the Java GUI will not be readable or writable by anyone else. This has the potential for breaking lots of things.

6.2 Searching in the Database from UNIX

The *PERLTOOLS* directory contains a program called `pattern_match.pl` that takes as an argument a series of words and searches the vulnerability database for matches on those keywords. This perl program loads the database index and calls the `fgrep` program to search every file called `V` in the *VDB* directory. Searching the database this way can bring any machine to its knees (every search opens thousands of files) so we do not recommend that you use it unless you have no other choice.

The same directory contains two other programs that are particularly useful for searching the database The first is a program called `searchServer.pl` and it essentially loads the entire database to memory and does a pattern search using perl. The server will check to see if new records have been added or if records have been modified every thirty seconds and will load/reload as needed.

The second program is called `searchClient.ps` and it contacts the server and gives it a string to search for and displays whatever the search server returns.

The match is similar to `fgrep` in that the entire text passed has to be matched. Unlike `fgrep`, however, the search string is altered a little bit before its used in the server:

```
$line = "" if length($line) > 200; # We only want reasonable entries
$line =~ s/^(\s)*(\S*.*\S)(\s)*$/$2/; # Remove leading and trailing spaces
$line = "" if !defined($line);
$line =~ s/[^a-zA-z0-9\@\_\-\!\%\'\:\.]/ /g; # Special characters are not allowed
$line =~ s/\s\s+/ /g; # Replace multiple spaces with single spaces
$line =~ s/^(\s)*(\S*.*\S)(\s)*$/$2/; # Remove leading and trailing spaces
```

In it's output the search server prints the results with some very simple filling so that what you see in the screen is not likely to be in the same format as what you see in the database. If this is a problem then please let me know. After all, this tool is used mainly to find out if an entry we are about to add already exists in the database.

To run the search server `cd` to *PERLTOOLS* and type:

```
% ./searchServer.pl -p 6768
```

where the `-p` option tells it which port to use. To run the client, `cd` to *PERLTOOLS* and type:

```
% ./searchClient.pl -p6768 -k "String to search for"
```

where the `-p` option tells it which port the server is using.

Be warned that the server is not a full-blown daemon and should not be run in the background of some obscure window. Create a separate `xterm` window for it and look at it periodically to make sure that no errors are being ignored. I have not tested this beast completely.

6.3 Using the Database

Using the database is fairly straightforward. Running the Java interface will present you a the main screen as shown in figure 3. Double-clicking on an item on the list of vulnerabilities will display the contents of that record.

The file menu, shown in figure 4, has options for saving your changes to the database, exiting the database, printing the record as pure text, and exporting the record as a multi-part MIME file.

The view menu, shown in figure 5, has options for displaying information regarding the field rating system, the classifiers used for the database, etc. Of particular importance is the option for indicating to the GUI interface to eliminate from the record display selected fields. This is particularly useful when fields such as patches and exploit scripts clutter the screen and the user wishes to view records without displaying these fields. When printing records the interface will also only print the fields as indicated by this menu.

The edit menu, shown in figure 4, has options for editing the current record and adding new records to the database. When you create a new record, a dialog is presented to the user requesting a record ID and title for the new vulnerability. Once this information is presented a new blank record is created for that vulnerability.

When editing a record the GUI interface will open a window, shown in figure 6, that contains fields and pop-up menus for entering data. Fields that have classifiers are marked by including the name of the classifier in parenthesis under the field name and you can display the classifier by clicking on the name of the field. If the field has a classifier and is a text field then and the you are not required to enter data that matches the classifiers. However, the GUI will complain about it and we strongly recommend that you do stick with well defined choices.

Some fields in the database have associated confidence ratings that give users an idea of how reliable is the data for that particular field. The rating system is as follows:

Value of 0: Item has not been rated. Users will generally make no assumptions about the information in this field. Items with a rating of 0 should not be trusted or used to justify any results.

Value of 1: Item is likely to be a guess or speculation.

Value of 2: Item is not likely to be correct and limited trust should be put on it.

Value of 3: Item is likely to be only partially correct, may contain errors, may be incomplete, etc.

Value of 4: Item seems to be correct but has not been verified by a trusted party. The operator that entered this information, to the best of his knowledge, believes the information to be accurate.

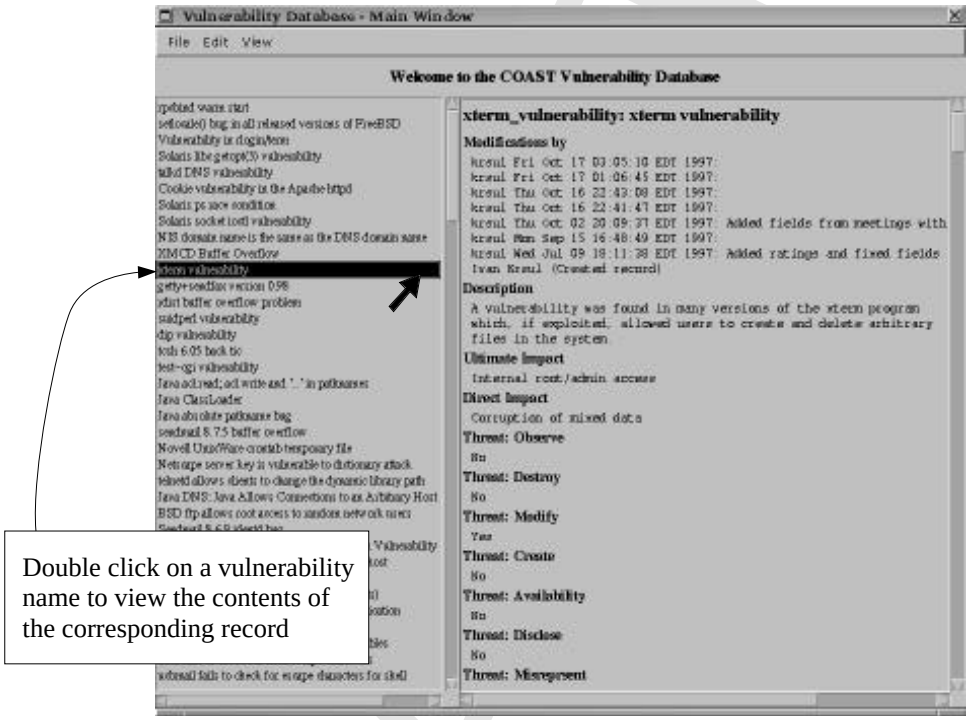


Figure 3: The main screen of the database. Double-clicking on an item on the list of vulnerabilities will display the contents of that record.

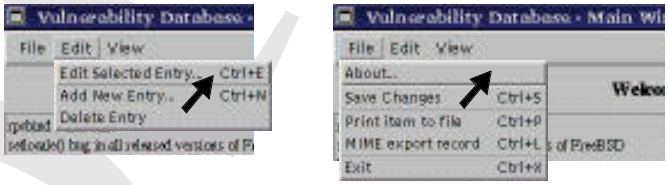


Figure 4: The file menu has options for saving your changes to the database, exiting the database, printing the record as pure text, and exporting the record as a multi-part MIME file. The edit menu has options for editing the current record and adding new records to the database

Figure 5: The view menu has options for displaying information regarding the field rating system, the classifiers used for the database, and for limiting the fields that are displayed or printed for a record.

Values of 5: Item is correct and has been verified by a trusted entity. The operator has evidence that the item is correct and can guarantee, with a high probability, that the item contains accurate and complete information.

When entering data you should be specially careful to enter the appropriate rating for the data that you are entering. Leaving that rating at it's default value of zero will cause the data that you are entering to be ignored in some automatic processes.

All text fields where you can type information can have MIME parts inserted within the text[1] MIME parts are manipulated by using the following keyboard commands while in the text field:

`<control-i>`**:** Insert textual mime part. Opens a dialog, as shown in figure 7, that allows the user to type or paste some text into the field and insert it as a MIME part.
Important Note: *The editor is not smart enough, nor it should be, to notice that you have inserted a MIME part and that it should remove the corresponding file if you decide to discard your changes to the record. Hence, if you add a MIME part and then discard your changes to the record you will have a MIME part file in the MIMEINCLUDES directory that will not be referenced by any record. Hence, delete the MIME parts created manually before discarding your changes to the record if you want the MIME parts to be discarded too!*

`<control-d>`**:** Delete MIME part. This option deletes the MIME part where the cursor is located. The MIME include directive is removed from the text and the MIME part file is deleted from the file system.

`<control-e>`**:** Edit MIME part. If the MIME part is editable then this command allows the user to edit the part in a special MIME part editor as shown in figure 7.

`<control-v>`**:** View the MIME part. Displays the content of the MIME part in a special window.

`<control-x>`**:** Export MIME part. *Not implemented yet!* Allows the user to export this part to a multi-part MIME file that can be viewed with an external viewer or that can be send via email.

`<control-m>`**:** View part with an external viewer. If the MIME part is not a textual part then it cannot be viewed using the `control-v` command. This command saves exports the part as a temporary file and calls an external MIME viewer to display the part.

[1] Fields that have associated classifiers can also have MIME parts. However, we don't recommend that this be done as some utilities will not work correctly in this case.

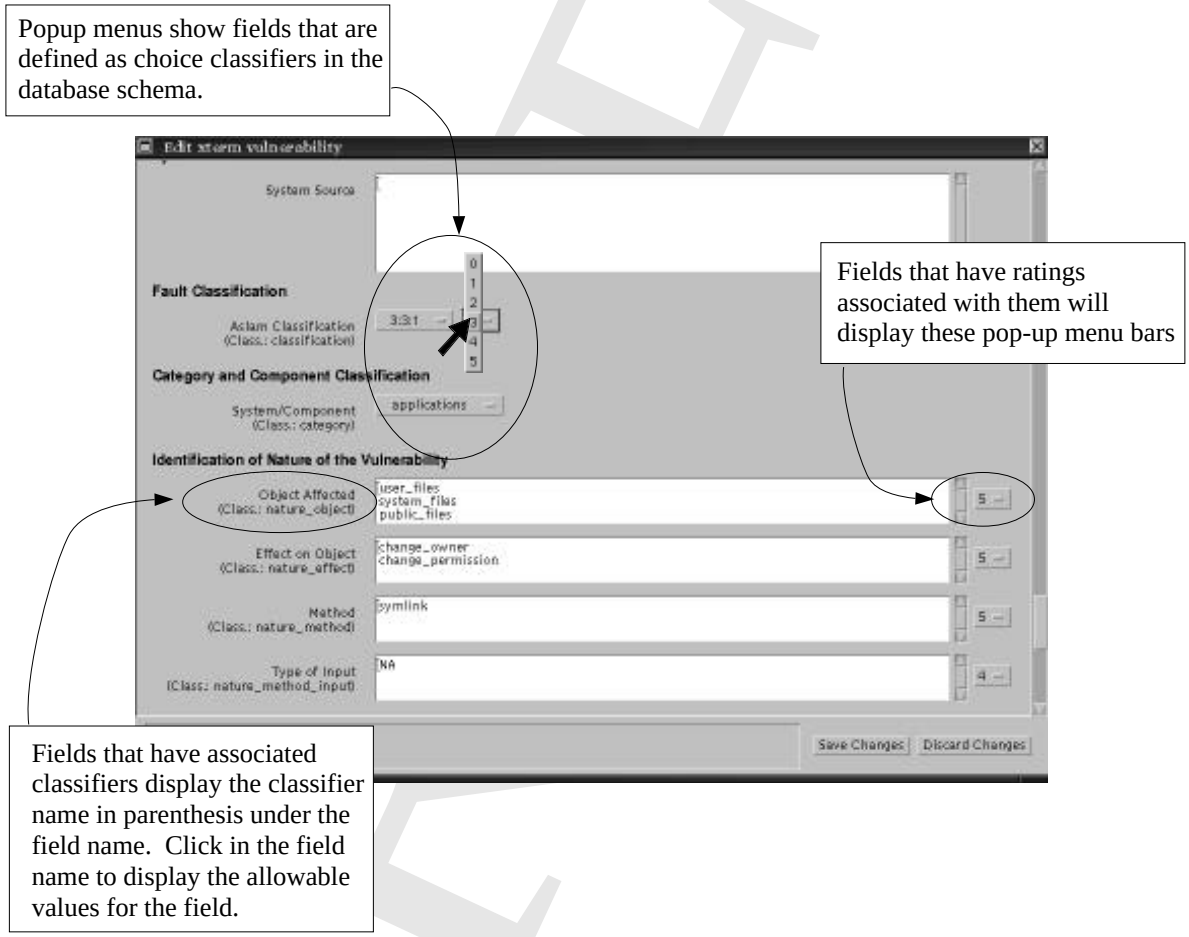


Figure 6:

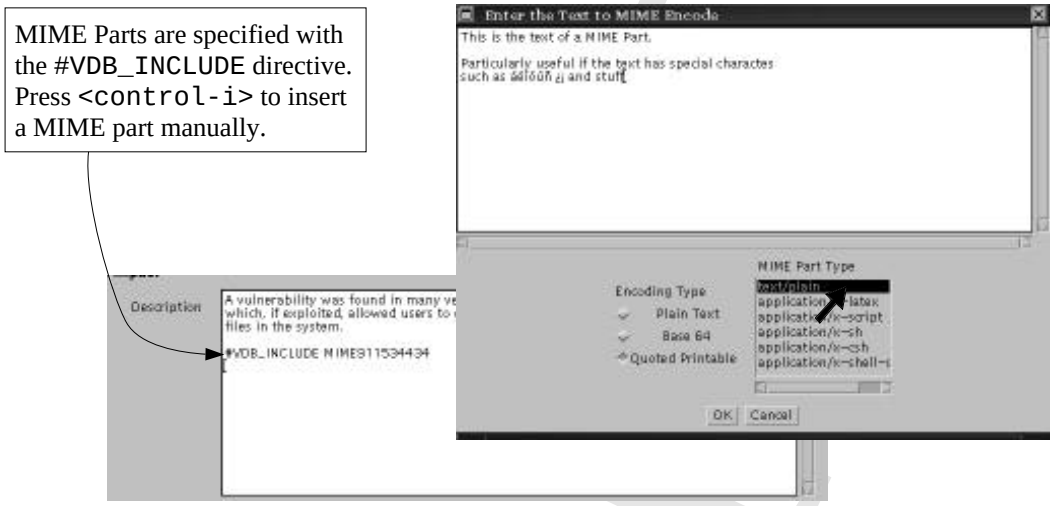


Figure 7:

<control-f>: MIME encode a file. This command opens a file dialog box and lets the user select an external file that must be MIME encoded and saved to a MIME part for the record. Once the file is selected, the interface will attempt to guess the MIME type and will open a dialog box, shown in figure 8, to confirm that the type selected is indeed correct. If it is not, then select the correct type and proceed with the conversion.

Figure 8: When MIME encoding a file, the interface will attempt to guess the MIME type and will ask the user to confirm that the type selected is indeed correct

As shown in figure 9 MIME parts are highlighted in the main window and can be viewed by double clicking on the name of the included part. *Bug Note:* Under some window managers in UNIX, a double click is defined as two successive clicks on the mouse with the *middle* button. With other window managers the double click is done with the left button.

7 Data Entering Procedures

In [KSb] we claim that there are four requirements that must be satisfied by the features (or values of fields that have associated classifiers) we select: **Objectivity** (the features must be identified from the object known and not from the subject knowing), **Determinism** (there must be a clear procedure that can be followed to extract the feature), **Repeatability** (several people extracting the same feature for the object must agree on the value observed), and **Specificity** (the value for the feature must be unique and unambiguous).

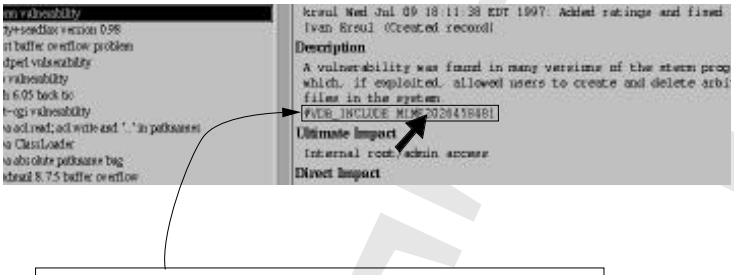


Figure 9: MIME parts are highlighted in the main window and can be viewed by double clicking on the name of the included part.

During the summer of 1997, however, we realized that there were many fundamental problems with the features collected for the vulnerability analysis project at COAST. Many of the features we had collected were ambiguous, many were not repeatable, many were not specific, etc. (See [KSb] for the details).

One possible solution to these problems is to patch these features so they will satisfy as many—if not all—of our requirements as possible. Hence, for example, features that are ambiguous could be fixed by providing instructions that will resolve the ambiguities. Features that are not deterministic could be fixed by providing a procedure that must be followed for the determination of the value of the feature.

The greatest objection to this solution is that we do not have (and sometimes we cannot have) enough information to fix the features objectively. Any fix we provide will be biased to our interpretation of what the author or creator of the feature really meant.

However, we argue that it is desirable to have these *fixed* features rather than the old. Any analysis with the old features is likely to be biased and highly contested because different researchers can come to different conclusions based on the values they choose for their features and there is not procedure that can help resolve such conflicts.

We chose to fix some of the problems of some of the features listed in [KSb] by providing a decision tree that will both remove ambiguities and provide a deterministic procedure that can be used to determine the value of the feature. The selection of values using these trees does not solve the problem of objectivity and does not remove ambiguities completely because to select a value for a feature a series of questions must be answered and there is no guarantee that different people will not answer these questions differently. This problem is particularly acute in vulnerabilities because the questions asked are bound to be highly technical and can only be answered by a person intimately familiar with the vulnerability. Vulnerability analysis is highly subjective and it seems paradoxical to attempt to extract objective features from a subjective procedure.

In the development of these decision trees we have attempted to choose questions that are as objective as possible and where appropriate we have chosen to annotate the decision procedure to clarify doubts and increase the quality of these features. These annotations are indicated by a number inside a circle in the lower right corner of the corresponding decision.

Feature `indirect_impact`**:** This feature attempts to identify the indirect or ultimate impact of the vulnerability. Indirect impacts are those that ultimately result from the exploitation of the vulnerability. See figure 10

Feature `direct_impact`**:** This feature attempts to identify the direct impact of the vulnerability. Direct impacts are those that are felt immediately after the vulnerability is exploited. See figure 11

Threat features: The original threat feature—extracted from "Current and Future Danger: A CSI Primer on Computer Crime & Information Warfare" by Richard Power [Pow96],–was developed as a classification of hostile actions that an adversary could take against your system.

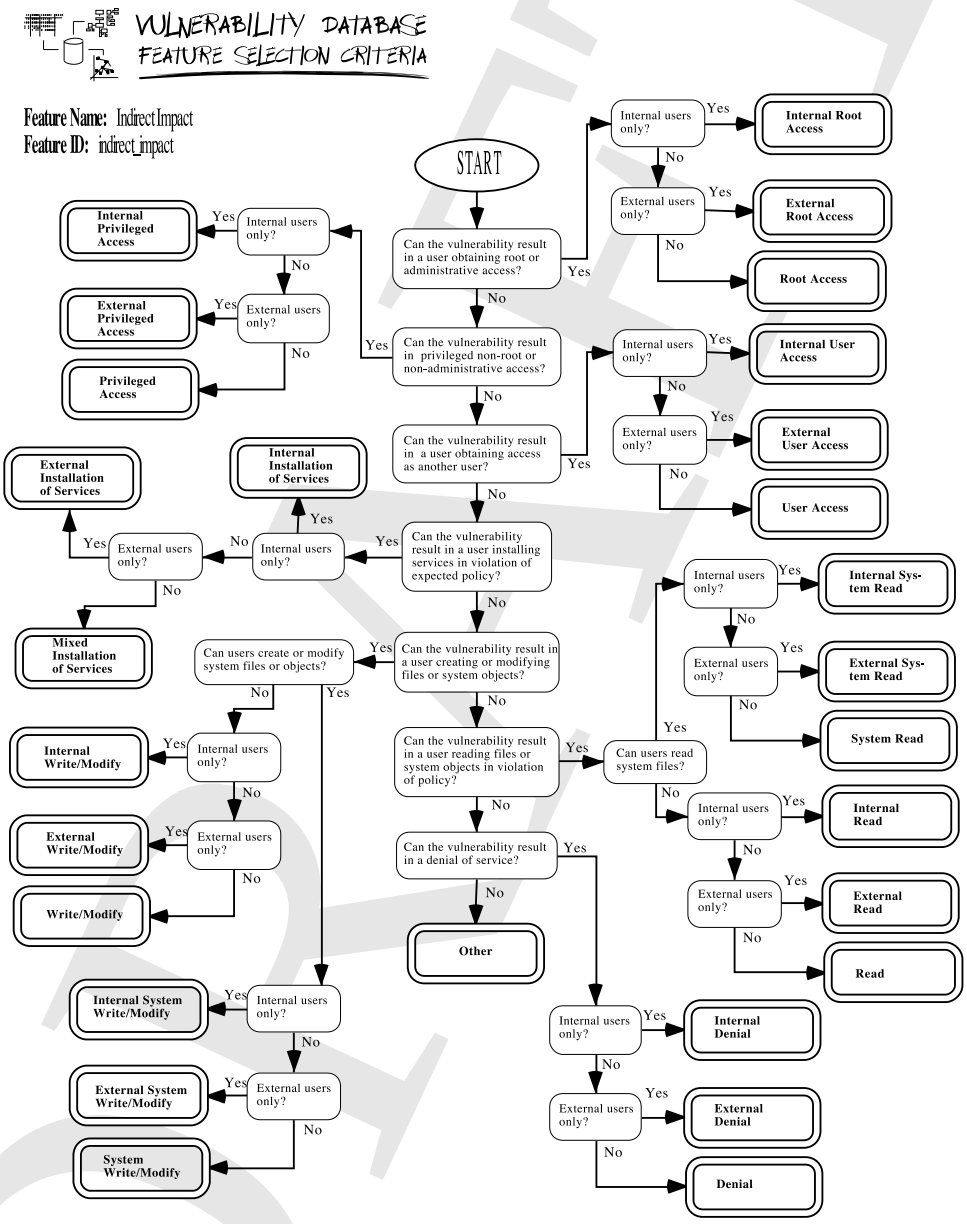


Figure 10: Selection decision tree for the `indirect_impact` feature.

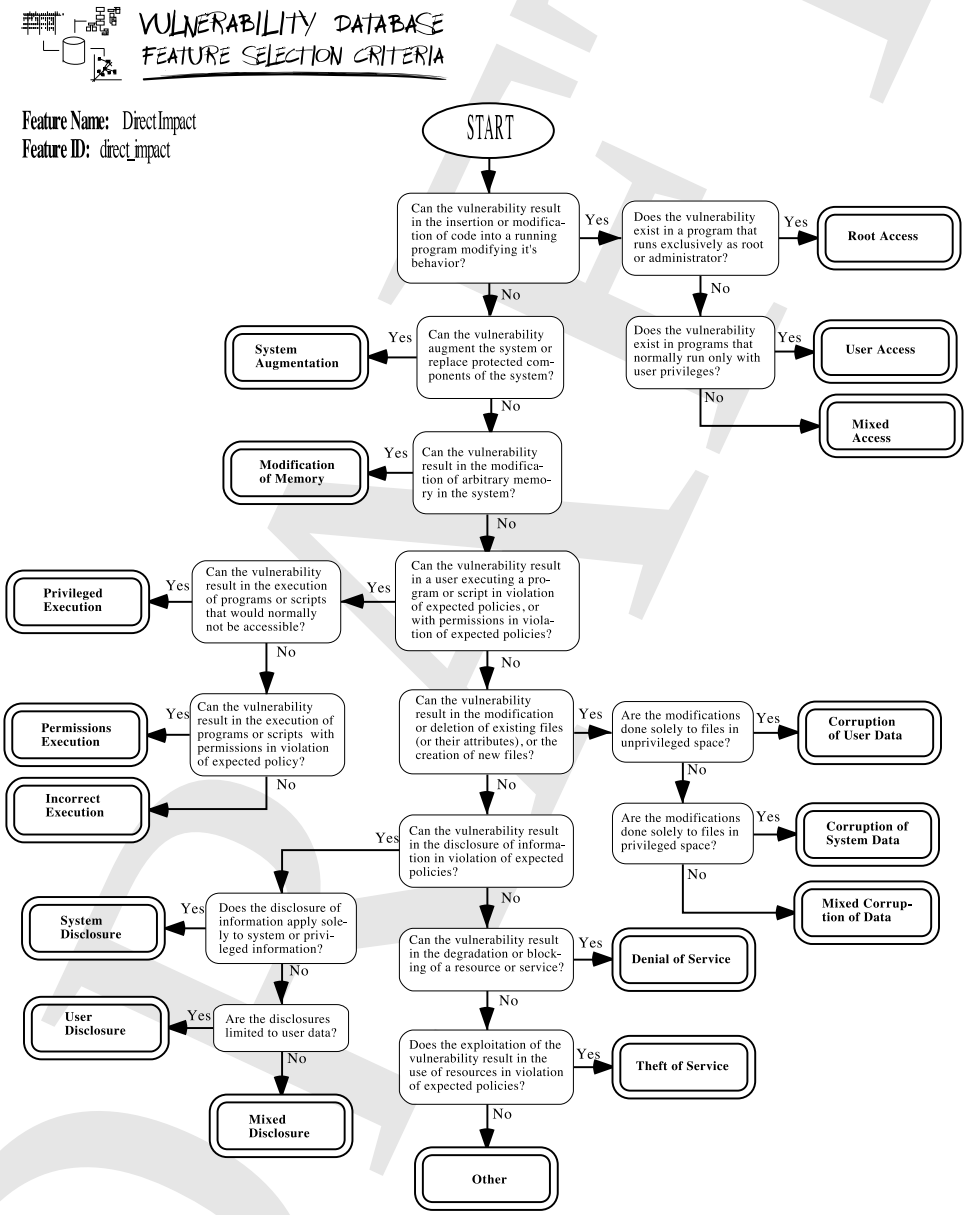


Figure 11: Selection decision tree for the `direct_impact` feature.

This classification is difficult to apply to the vulnerability database because it is essentially ambiguous. In [KSa] we argue that classification trees should use decision nodes that use one *fundamentum divisionis* and as can be seen in figure 12, the threat classifier does not follow this principle.

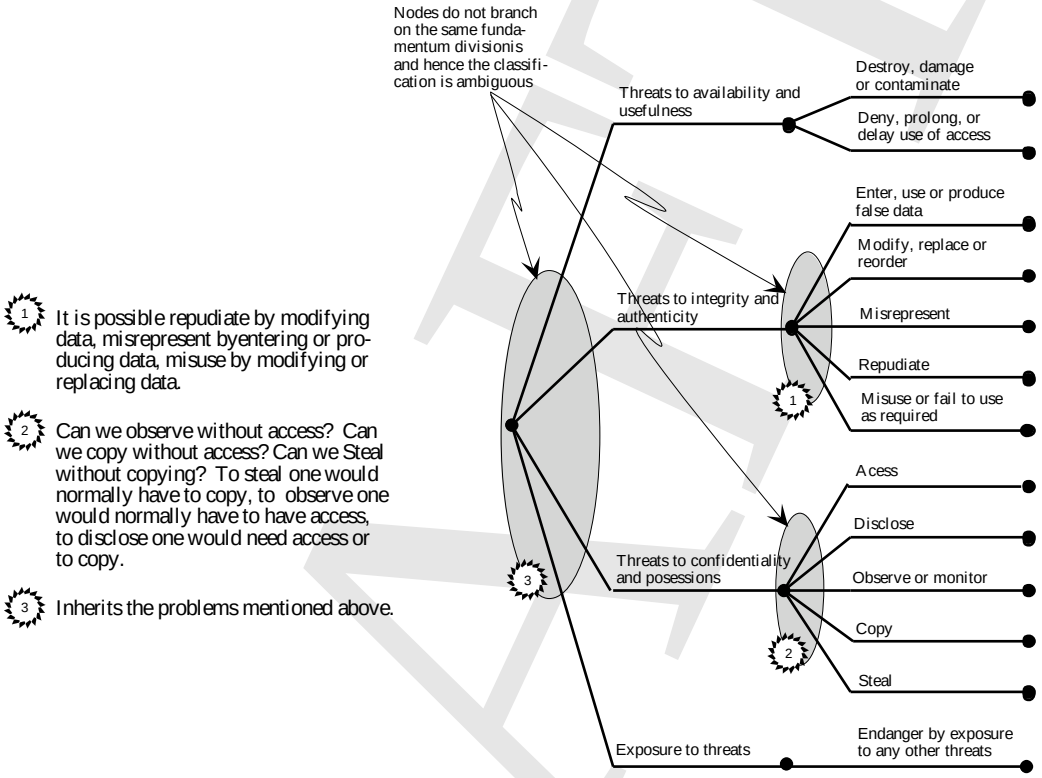


Figure 12: The Threat classifier is ambiguous because it uses nodes that have more than one *fundamentum divisionis*

A *fundamentum divisionis* is a term from Scholastic Logic and Ontology that means “grounds for a distinction” [Aud95]. Ambiguities can arise when the selection criteria for an internal node of the tree has more than one *fundamentum divisionis*. In the threat classifier, a single node in the classification tree branches into *Access*, *Disclose*, *Observe*, *Copy*, and *Steal*. The categories *Observe* and *Access* are concrete actions while the category *Steal* is subjective and requires a value judgment. Hence, it is possible to *Access* and *Steal* simultaneously.

Another of the fundamental problems with features of this kind is that they do not specify an explicit maximum level of indirection that can be used for the determination of the threat of a vulnerability. Hence, a vulnerability that causes the encrypted password of a user to be displayed is a treat to integrity if the password is decrypted, the user account is compromised, an encrypted administrator password can be obtained using this account, this last password can be decrypted, a root shell can be obtained. Because root shells allow any operation to proceed, integrity can be violated. A similar reasoning can be applied to most of the UNIX vulnerabilities we have seen. **Hence, it is important that we clearly specify that we are interested in the immediate threat that is present with the vulnerability**.

We can split the threat classifier into a list of action and consequence binary features that follow:

Action Features

`thac_observe`: The vulnerability[2] can result in a user observing objects, data, etc., in violation of expected policy.

`thac_destroy`: The vulnerability can result in a user destroying objects, data, etc., in violation of expected policy.

`thac_modify`: The vulnerability can result in a user modifying objects, data, etc., in violation of expected policy.

`thac_create`: The vulnerability can result in a user creating objects in violation of expected policy.

`thac_exec`: The vulnerability can result in a user executing a program in violation of expected policy.

Consequence Features

`thac_cavail`: The vulnerability can result in the change of availability of the system.

`thac_disclose`: The vulnerability can result in the disclosure of information in violation of expected policy.

`thac_misrep`: The vulnerability can result in misrepresentation of information.

`thac_repudiate`: The vulnerability can result in repudiation of information.

`thac_integrity`: The vulnerability can result in a change of integrity of the system.

`thac_conf`: The vulnerability can result in the loss of confidentiality of information.

Each of these features can take the values "Yes," "No," "Does Not Apply," and "Unknown". Hence, each feature is a decision tree with a depth of one that has a single *fundamentum divisionis*.

The other categories (steal, copy, contaminate, misuse, fail to use as required, change in usefulness, production of false data) are ambiguous, subjective, abstract, or equivalent to the features shown.

Feature `access_required`: This classifier was originally defined from a talk given by Tom Longstaff [Lon97] and defines the kind of access that is required to exploit the vulnerability. See figure 13

Feature `complexity_of_exploit`: This feature attempts to identify the complexity of the exploitation of a vulnerability, regardless of whether a script or toolkit exists for the exploitation of the vulnerability. See figure 14

1. The notion of a *simple sequence of commands* will, of course, vary from person to person. We will consider a *simple sequence of commands* a linear sequence of commands (i.e. no loops, gotos, etc.) of no more than a dozen commands. Also, these dozen commands must be common commands supported by the operating system, common applications and utilities. Commands that involve scripts and applications that the exploiter must compile, install, etc., do not qualify.
2. Shell scripts, command interpreter source files and macros all qualify. Programs that are implemented in a general purpose programming language (including such languages as Perl) do not qualify.
3. Typically requires a script or application that tries several times and may require slowing down the system.
4. Applications that the exploiter must compile, install, etc.

Aslam Classification: The Aslam classification has been expanded and a decision tree has been introduced to eliminate ambiguities and resolve some conflicts. See figures 15 and 16.

[2] In this list of features, for brevity, "The exploitation of the vulnerability" has been shortened to "The vulnerability"

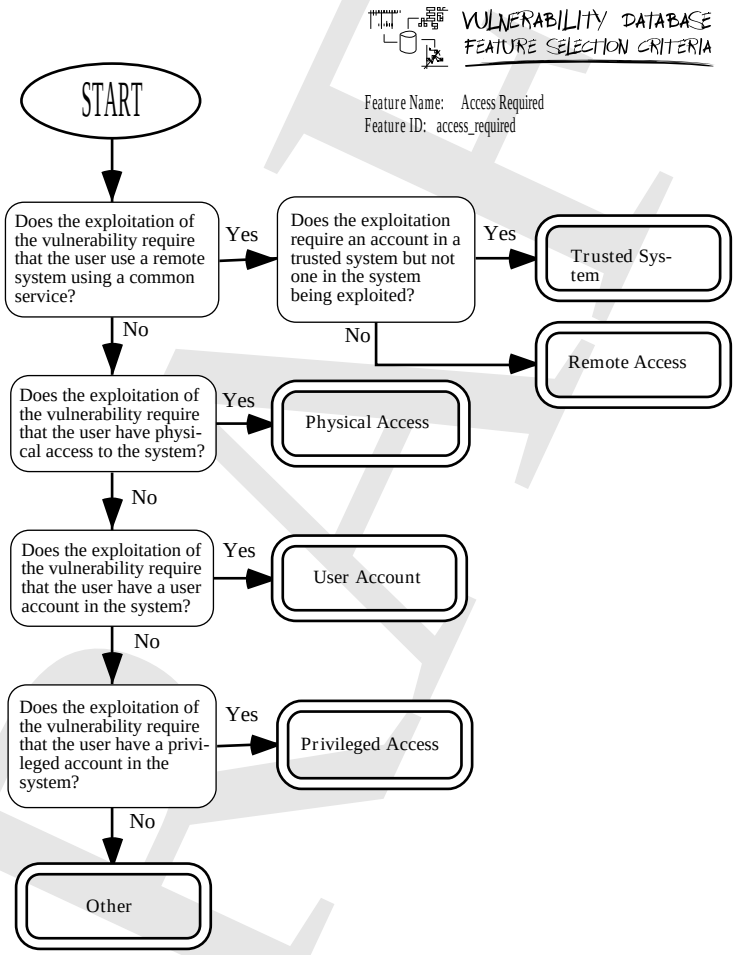


Figure 13: Selection decision tree for the `access_required` feature.

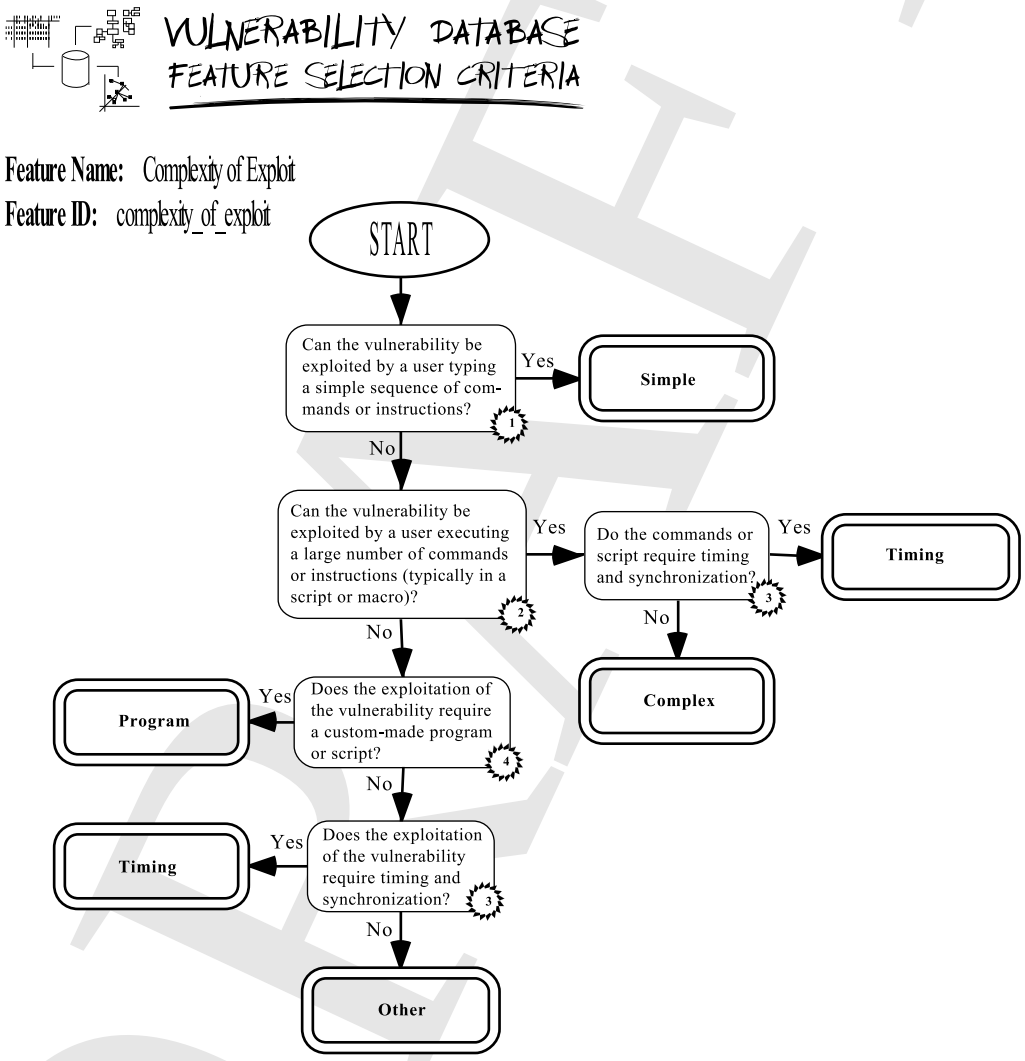


Figure 14: Selection decision tree for the `complexity_of_exploit` feature.

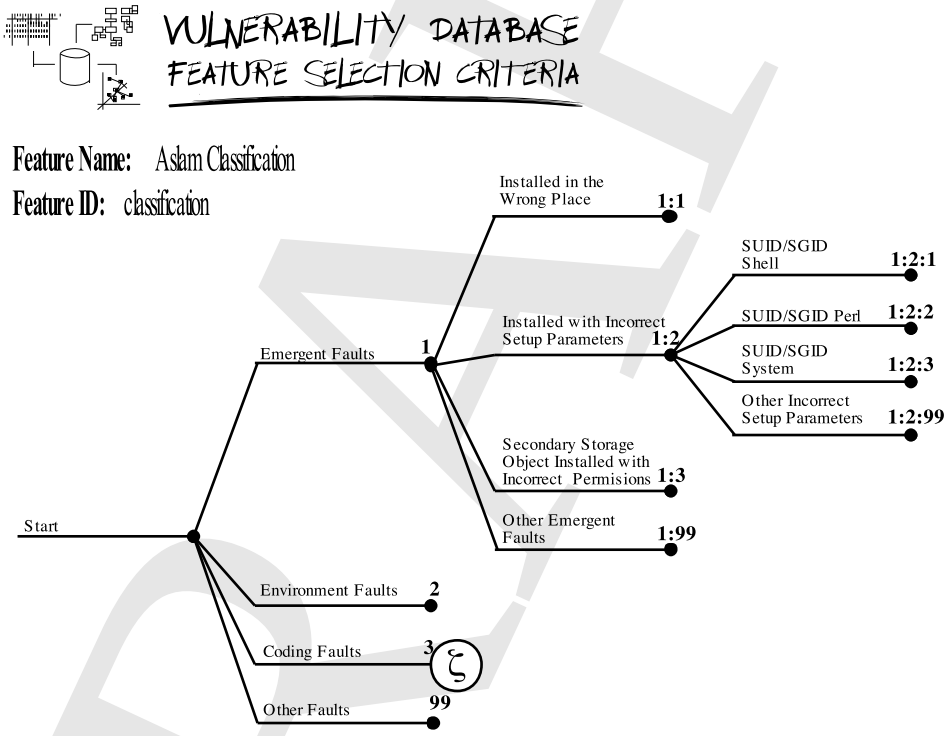


Figure 15: Aslam Classification decision tree (part 1 of 2) for the `classification` feature.

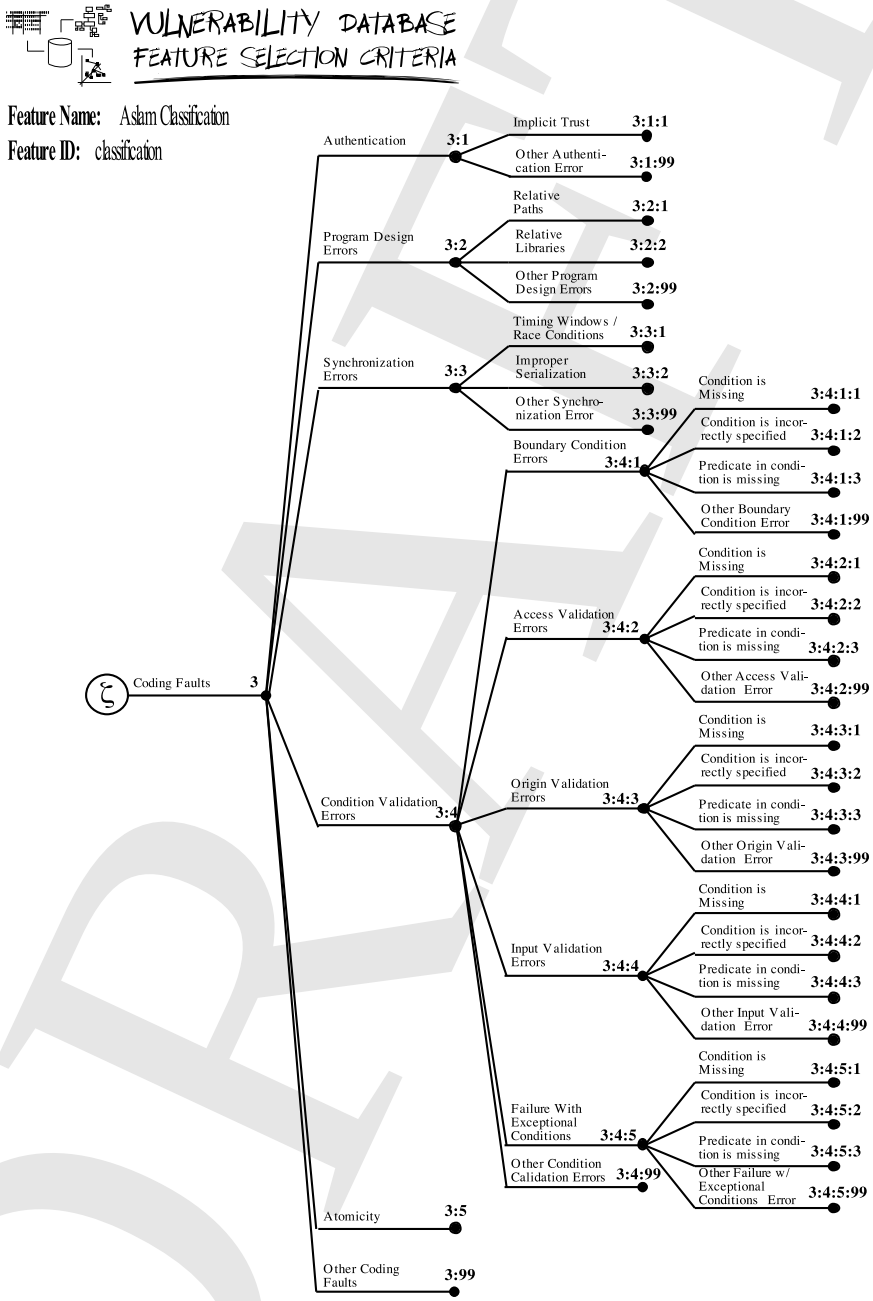


Figure 16: Aslam Classification decision tree (part 2 of 2) for the `classification` feature.

Aslam Classification	Description
1:2:3	SUID/SGID routines that use the `system()`, `popen()`, `execlp()`, or `execvp()` calls to run something else.
2	Environment faults are introduced when specifications are translated to code but sufficient attention is not paid to the run-time environment. Environmental faults can also occur when different modules interact in an unanticipated manner. Independently the modules may function according to specifications but an error occurs when they are subjected to a specific set of inputs in a particular configuration environment.
3:1	Failure of software to authenticate that it is really communicating with the desired software or hardware module it wants to be accessing.
3:1:1	For example, routine B assumes routine A's parameters are correct because routine A is a system process.
3:3:1	A fault can be exploited because of a timing window between two operations.
3:3:2	A fault results from improper serialization of operations.
3:5	Did the error occur when partially-modified data structures were observed by another process? Did the error occur because the code terminated with data only partially modified as part of some operation that should have been atomic?

Feature `envass`**:** This feature attempts to identify the environmental assumptions that were made by programmers or designers and that, if correct, would make the program correct.

The features can have the following possible values:

envass Possible Values	
Item Value	**Item Description**
nameinv	Assumes that a name (i.e. a path) is strongly bound to a specific system object.
objinv	Assumes the invariance of an object during the execution of a program (i.e. the program assumes that no other subject can change the object while the program is running)
objne	A program assumes that an object does not exist at the time of execution (i.e. a program assumes that a file with a specific name does not exist)
tempdel	A program assumes that a temporary item it created cannot be deleted by any other subject while the program is running.
memavail	A program assumes that sufficient memory for it's execution will always exist.
netdata	A program assumes that data from a network service will always be reliable.
envdata	A program assumes that the data in environment variables is valid and bounded.
userdata	A program assumes that user-provided input is valid and bounded
filedata	A program assumes that the input from a file is valid and bounded
reassembly	A program assumes that the re-assembly of a data object form fragments will not affect the essential properties of the original object.
execpath	A program assumes a specific execution path.
objatt	A program assumes that certain attributes of certain objects have predefined values.
perstore	A program assumes that persistent store is immutable (i.e. assumes that a file it writes cannot be modified by any other subject in between program runs)
dataexec	A program assumes that the modification of program data (by external subjects) will not affect the semantics of the program.
nameover	A program assumes that, while creating a file, any existing file that has the same name can be overwritten.
falseconst	A program falsely assumes that a constraint or property holds in the system.
insufverif	A program falsely assumes that a set of operations are suffient for the verification of the property of an object
namepurpose	A program assumes that there is a strong binding between the name and purpose of an object
reservedobject	A program assumes that an object with a specific name will not be used by any other entity in the system by virtue of its name alone.
other	Other
NA	Does not apply
?	Unknown

Feature `category`**:** This feature attempts to identify the system component that a vulnerability belongs to. See figure 17

1. The notion of Operating System (OS) is ambiguous as some consider the OS to be the bare bones kernel and others include all the files that were shipped with the distribution of the OS. We take the view that

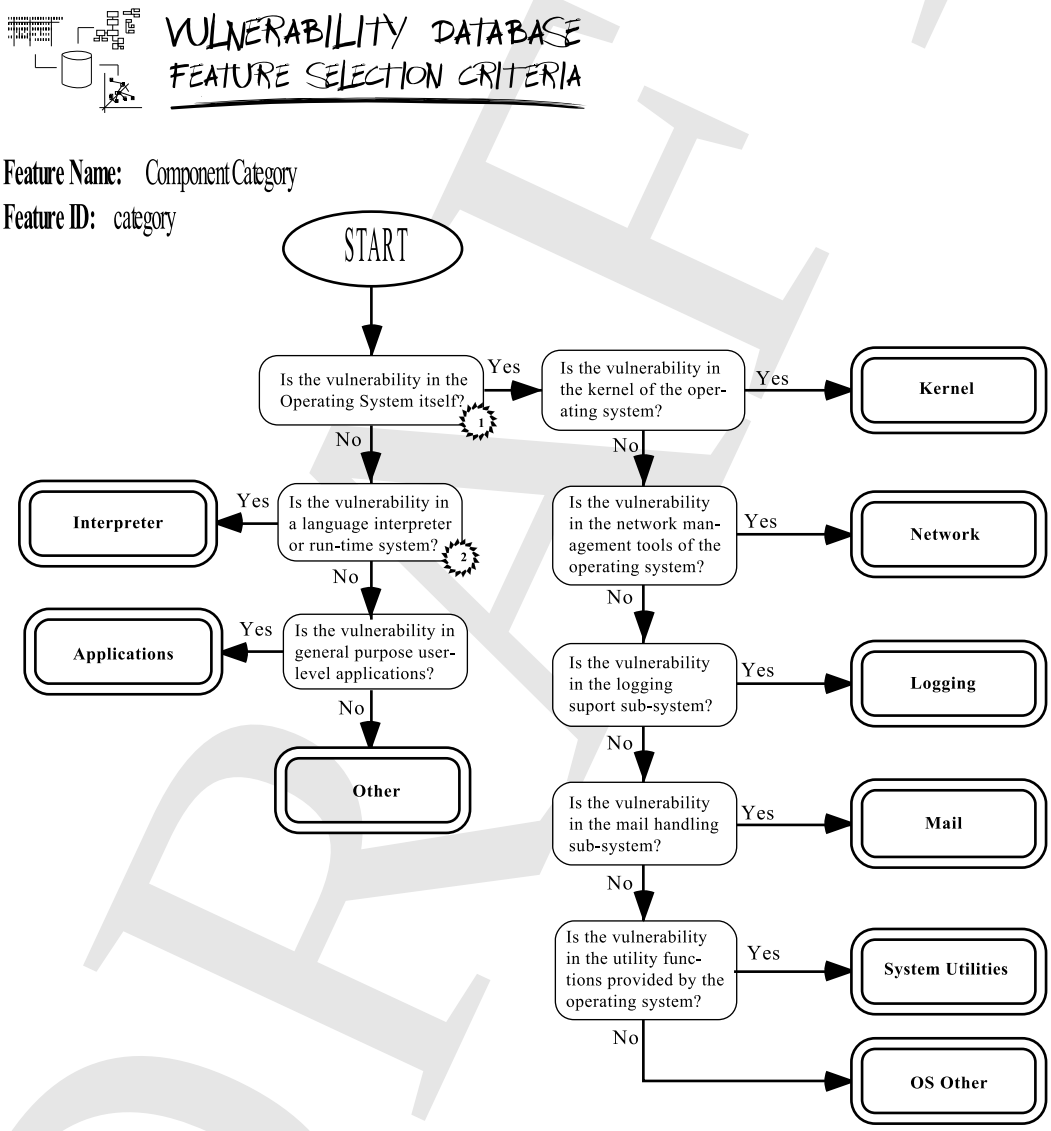


Figure 17: Selection decision tree for the `category` feature.

the OS is the kernel and all the utilities that are common to all distributions of that OS that are minimally required for it's operation. Hence, in the Windows NT 3.5 OS the Internet Explorer is not a part of the OS, even if it is included in all distributions because it can be deleted without affecting the operation of the OS. In versions of Windows NT where the Internet Explorer replaces the file system NT Explorer we would consider it a part of the OS.

Feature `os_type`**:** This classifier attempts to identify the class of operating systems that is affected by the vulnerability. See figure 18

Features `nature_object`**,** `nature_effect`**,** `nature_method`**, and** `nature_method_input`**:** These features attempt to capture the fundamental consequences of vulnerabilities by looking at the object that is essentially affected by the vulnerability, the effect that the vulnerability has on that object, the method or means that lead to the effect on the object, and, if appropriate, the type of input that leads to the effect on the object.

The features can have the following possible values:

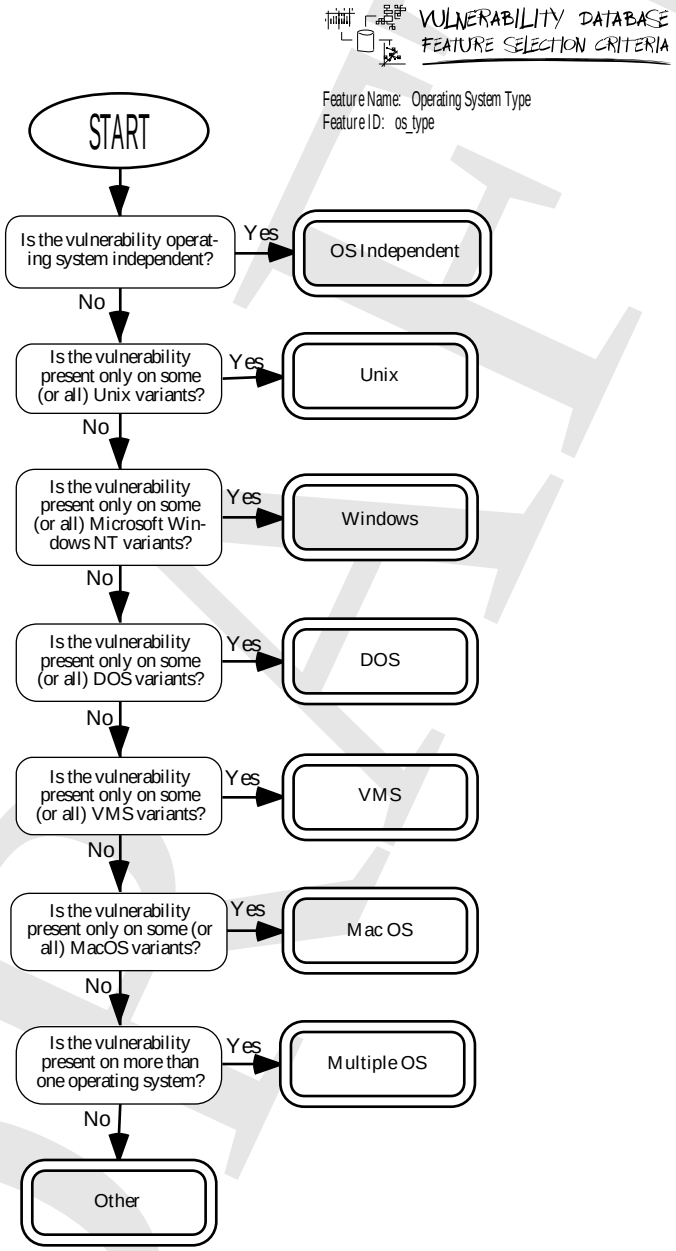


Figure 18: Selection decision tree for the `os_type` feature.

nature_object Possible Values	
Item Value	**Item Description**
user_files	User files in the system.
system_files	System-related or administrative files in the system.
public_files	Publicly available files in the system.
directory	Directories in the system
partition	A file system partition
heap_data	Data in the heap of running program.
heap_code	Executable code in the heap of running program.
stack_data	Data in the stack of a running program.
static_data	Data that is statically allocated in a running program
stack_return	Return address of a function in the stack of a running program.
stack_code	Executable code in the stack of a running program.
password	Password or access token. Can also be a pass-phrase.
shell_command	Shell command.
system_program	System program.
user_program	User installed or owned program.
system_info	Information regarding the system.
outfiles	Files outside a restricted space. Describes files that should not be visible/accessible outside a chrooted environment, virtual environment, sandbox, etc.
classloader	A ClassLoader object in Java or any object responsible for loading dynamic classes in any object oriented programming language.
library	System function or service library.
a_net_connection	Network connections to arbitrary hosts.
web_pages	WWW page.
names	User names, domain names, work-group names, etc.
pass_known	Well-known nonce encrypted with user password.
o_attributes	System-managed object attributes. Attributes the object itself (or entities other than the system) does not manage.
cpu	CPU time.
os	Operating System.
email	Electronic Mail
netport	Network Port
packets	Network Packets
system_names	Internal system names in control of the system.
device	A device in the system
addr_mapping	Address mapping maintained by the system. i.e. an ARP cache
command_prompt	A command prompt presented to the user
other	Other
NA	Does not apply
?	Unknown

nature_effect Possible Values	
Item Value	**Item Description**
replaced	Contents are completely replaced.
changed	Can be written to or can be changed.
read	Can be read.
append	Information can be appended.
created	Can be created.
displayed	Displayed or revealed.
change_owner	Ownership can be changed.
change_permission	Permissions can be changed.
predictable	Is predictable or can be guessed.
executed	Can be executed in violation of expected policy.
loaded	Can be dynamically loaded and linked.
clear_text	Is transmitted or stored in clear text.
exhausted	Is exhausted.
crash	Crashes
bound	Can be bound to in violation of expected policy
exported	Can be exported for mounting
mounted	Is mounted or attached
locked	Can be locked
debugged	Can be debugged or attached to with a debugger
presented	Presented to the user in a console or terminal
other	Other
NA	Does not apply
?	Unknown

nature_method Possible Values	
Item Value	**Item Description**
symlink	Program follows symbolic link or late binding link without verifying that the object being pointed to is correct.
memcpy	Program uses strcpy, sprintf, or bcopy to copy data of arbitrary length to a stack buffer.
config	Configuration error.
back_ticks	Back ticks in parameter or input string.
special_chars	Special characters in input string, including file completion characters, special shell characters, etc.
dotdot	Uses ".." to climb up a directory tree past allowable bounds.
verify_fail	Bytecode or code verifier allows code that catches a security exception when creating an object loader.
mod_name	Modifying compiled code to alter the name of objects.
mod_env	Modifying environment variables.
NTML_auth	NTML authentication process requires action.
inherit_privs	Program inherits unnecessary privileges.
capability	System provides inappropriate capability.
hidden_mount	System provides hidden system mount point
syscall_disclose	System call discloses sensitive information.
incorr_imp	Incorrect implementation given current environment (mistaken environmental assumption)
rel_paths	Program refers to relative paths
incprot	System fails to implement the protection mechanisms correctly
proxy	Program uses a trusted intermediary or proxy to bypass protection mechanisms
coresymlink	A program dumps a core file that follows symbolic links or late binding link.
infloop	Program uses an infinite and tight loop that consumes resources
criticalsect	Program fails to protect or isolate a critical section
other	Other
NA	Does not apply
?	Unknown

nature_method_input Possible Values	
Item Value	**Item Description**
env	Environment variable.
command	User command line option.
netdata	Network data.
store	Persistent store.
tempfile	Temporary file.
conffile	Configuration file.
datafile	Data file.
gecos	System User information (Name, phone number, etc.)
parameter	Parameter to a system call
libparameter	Parameter to a library call
floppy	Removeable media
other	Other
NA	Does not apply
?	Unknown

8 Classifiers

Classifiers are stored as files in the directory *CLASSD*. Depending on whether the classifier is a straight list or a choice list,the first line of the classifier includes the type. Classifiers can have comments and all blank lines in classifiers are ignored.

For example, a standard choice list for yes/no fields is the `yes_no` classifier. It typically has the following contents:

```
#cclass

# Choice list for a yes/no choice.
# This choice list also allows the
# user to specify that a yes/no answer
# is not appropriate or that a value is
# not know for the record.

yes#Yes
no#No
NA#Does not apply
?#Unknown
```

An example of a list classifier is the `system` classifier that defines the kinds of operating systems that are allowed. A reduced example of this classifier is:

```
#list
# This classifier is used to define
# operating systems and has the
# following form:
#
#  Operating system code#Operating system name
#
Solaris#SUN Solaris
SunOS#SUN OS
DOS#Microsoft DOS
Windows95#Microsoft Windows 95
Caldera#Caldera
Goah#NEC's Goah
NA#Does not apply
```

A complete list of the classifiers defined for the database is given in section 14.

9 An Annotated Example

This section presents an annotated example of a vulnerability record in the database. Annotations are either given as footnotes or endnotes denoted by the symbols ①, ②, etc.

VULNERABILITY ID:
`solaris_getopt`

TITLE:
Solaris libc `getopt(3)` vulnerability

DESCRIPTION:
A buffer overflow condition exists in the `getopt(3)` routine. By supplying an invalid option and replacing argv[0] of a Set-User-ID (SETUID) program that uses the `getopt(3)` function with the appropriate address and machine

code instructions, it is possible to overwrite the saved stack frame and upon return(s) force the processor to execute user supplied instructions with elevated permissions.

ULTIMATE IMPACT:
`internal_root_access`

DIRECT IMPACT:
`mixed_access`

IMPACT TEXT:
Non-privileged users can exploit a vulnerability in the getopt(3) routine inside libc. As most SUID programs in Solaris are dynamically linked, users can gain root privileges.

IDENTIFICATION OF THE NATURE OF THREAT:

Threat Observe	Threat Destroy	Threat Modify	Threat Create
No	No	Yes	No

Threat Availability	Threat Disclose	Threat Misrepresent	Threat Repudiate	Threat Integrity	Threat Confidentiality
No	No	No	No	No	No

SOURCE DETAIL:
L0pht Advisory

SYSTEMS:
Solaris

SYSTEM VERSION:
Solaris 2.5

SYSTEM VENDOR:
SUN

TYPE OF OS:
UNIX

APPLICATION:
NA
APPLICATION VERSION:
NA
VERBOSE APPLICATION:
NA

ADVISORY/IES:
L0pht Advisory: `Solaris libc - getopt(3)`

ANALYSIS:
From: mudge@l0pht.com

While evaluating programs in the Solaris Operating System environment it became apparent that changing many programs trust `argv[0]` to never exceed a certain length. In addition it seemed as though `getopt` was simply copying `argv[0]` into a fixed size character array.

```
./test >>& ccc
```

```
Illegal instruction (core dumped)
```

Knowing that the code in `./test` was overflow free it seemed that the problem must exist in one of the functions dynamically linked in at runtime through `ld.so`. A quick gander through the namelist showed a very limited range of choices for the problem to exist in.

```
00020890 B _end
0002088c B _environ
00010782 R _etext
         U _exit
00010760 ? _fini
0001074c ? _init
00010778 R _lib_version
000105ac T _start
         U atexit
0002088c W environ
         U exit
0001067c t fini_dummy
0002087c d force_to_data
0002087c d force_to_data
000106e4 t gcc2_compiled.
00010620 t gcc2_compiled.
         U getopt
00010740 t init_dummy
00010688 T main
```

Next we checked out `getopt()` — as it looked like the most likely suspect.

```
#include <stdio.h>

main(int argc, char **argv)
{
  int opt;

  while ((opt = getopt(argc, argv, "a")) != EOF) {
    switch (opt) {
    }
  }
}

>gcc -o test test.c
>./test -z
./test: illegal option -- z
```

Note the name it threw back at the beginning of the error message. It was quite obvious that they are just yanking `argv[0]`. Changing `argv[0]` in the test program confirms this.

```
for (i=0; i< 4096; i++)
  buffer[i] = 0x41;

 argv[0] = buffer;
```

With the above in place we see the following result:

```
>./test -z
[lot's of A's removed]AAAAAAAAA: illegal option -- z
Bus error (core dumped)
```

By yanking out the object file from the static archive `libc` that is supplied with Solaris our culprit was spotted (note - we assumed that `libc.a` was built from the same code base that `libc.so` was.)

```
> nm getopt.o
         U _dgettext
00000000 T _getopt
00000000 D _sp
         U _write
00000000 W getopt
         U optarg
         U opterr
         U optind
         U optopt
         U sprintf
         U strchr
         U strcmp
         U strlen
```

Here we see one of the infamous non-bounds-checking routines: `sprintf()` More than likely the code inside `getopt.c` looks something like the following:

```
getopt.c:
  char opterr[SOMESIZE];
  ...
  sprintf(opterr, argv[0]...);
```

Thus, whenever you pass in a non-existent option to a program that uses getopt you run into the potential problem with trusting that `argv[0]` is smaller than the space that has been allocated for `opterr[]`.

This is interesting on the Sparc architecture as `getopt()` is usually called out of `main()` and you need two returns (note - there are certain situations in code on Sparc architectures that allow you to switch execution to your own code without needing two returns. Take a look at the TBR for some enjoyable hacking) due to the sliding register windows. Some quick analysis of SUID programs on a standard Solaris 2.5 box show that most of these programs `exit()` or more likely call some form of `usage()-exit()` in the default case for `getopt` and thus are not exploitable. However, at least two of these programs provide the necessary returns to throw your address into the PC:

```
     passwd(1)
     login(1)
\begin{verbatim}

On Solaris X86 you do not need these double returns and thus a whole
world of SUID programs allow unprivileged users to gain root access:

\newchar{Core Vulnerability}

Excerpt from actual rpcbind code from {\tt
/p/src/sun5.3/solaris2.3/os-net/src_ws/usr/src/cmd/rpcbind.c}
Full disclosure may violate NDA.  For {\tt getopt}:

\begin{verbatim}
#define ERR(s, argv0, c) if (opterr){\
  char errbuf[256]; \
  (void)
    sprintf(errbuf, s, argv0, c); \
  (void)
    write(2, errbuf, strlen(errbuf));}
```

```
main() {
  ...
  ERR("%s\n",argv[0]."");
}
```

FIX:
For those with source: If you are one of the few people who have a source code license the fix should be fairly simple. Replace the `sprintf()` routine in `getopt.c` with `snprintf()` and rebuild `libc`.

Super Ugly kludge fix: If you don't have the source code available (like most of us), one solution is to use `adb` to change the name for `getopt` with something like `getopz`, yank a publicly available `getopt.c`, and put it in place of `getopt`. If anyone can tell me how to yank the object files out of dynamically linked libraries it would be appreciated as you suffer performance hits among larger problems by doing this from the static library Sun provides as, of course, it is not PIC code.

ACCESS REQUIRED:
`user_account`

ASLAM CLASSIFICATION:
3:4:4:1

SYSTEM/COMPONENT:
`os_other`

OBJECT AFFECTED:
`stack_code`
`stack_return`

EFFECT ON OBJECT:
`changed`

METHOD:
`memcpy`

TYPE OF INPUT:
other

EXPECTED POLICY VIOLATED:

- System libraries should always check that their inputs are of the appropriate size
- The `getopt()` system library should only check the arguments of the program that called it and return.
- `#VDB_INCLUDE povide_buffer_overflow.MIME`[1] .

ENVIRONMENT FEATURES:

- Requires a stack based machine
- Stack must be executable

CHARACTERISTICS:

- Library call.
- Used by administrative programs that run SETUID root.

- Copies data to buffer without regards to bounds checking.
- Vulnerability involves the `sprintf` routine.
- Programmed in "C".
- Processes user input directly
- Routine does not validate parameters passed to it.
- Requires that the program that calls the library does not check the validity of the inputs either.

ANNOTATIONS AND ENDNOTES::
① The document that is referenced by this MIME tag is included in this document in section 15.

10 Detailed Description of the Schema File

The vulnerability database schema file (*VDBSCH*) describes the fields allowable in the database. It is a text file where comments are indicated by starting a line with the character #. Every non-comment line defines a field or a separating header.

Every field specification is composed of the following parts separated by the # character: Field ID, short name, long name, field type, field height. Field IDs should be unique. The short name is used to display identify the field in editors and reports. The long name is used to provide descriptive information about the field to the user on data entry and reports. The field type and number of lines tells the Java interface how to represent this field in the data entry editor.

The field types are one of `text`, `secsep`, `cclass`, and `list`[3]:

`text:` Fields of type `text` are general text fields that can contain any textual information. These fields can have included MIME parts that are specified by putting in a line an include directive of the form `#VBD_INCLUDE file_name` where the file name corresponds to a MIME part file in the appropriate directory under the *MIMEINCLUDES* directory.

`secsep:` Fields of this type are not real data fields but used to specify that the line in the schema is a section separator.

`cclass:` These are choice lists. The user is presented with a list of choices and can select a single item from the list.

`list:` These fields are lists where the user can select multiple elements from the list.

Fields can have options that are indicated by following the field type by a question mark and the options desired. As of this release, the options defined are:

`c:` Every field can have a *confidence interval*. These are ratings that are given to the field that indicate the level of confidence that the data entry operator has on the correctness of the value.

The following text is an example of the schema file for the database. If represents the database as it was defined on October 19, 1997.

IMPORTANT NOTE: The schema file does not support continuation characters. In this example, however, we have added two backslashes (\\) to split lines that were to long to display in this document.

```
# One line per field. Each line has the vulnerability field ID,
# a title, a long [er] description of what the description
# is, a field type, and the recommended editor height
#
# Field types can be one of:
# text (A text field.)
# class:classifier_file_name (Choice list classifier)
# list:list_file_name (List of allowed values)
#
```

[3]The database supports fields of type `hclass` and `matrix` but these are not used anymore. Look at the code to find out how to use these!

```
# Please be ware that the field title is hard-coded
# in all the programs and it should never change or be removed.
#
# March/97 - Ivan Krsul - krsul@cs.purdue.edu
#

# Identification
*#Identification#*#secsep#1
title#Title#Title of the vulnerability#text#2

# Information about modification
*#Modification History#*#secsep#1
modifications#Modifications by#Person(s) that have modified this record, \\
  the date of modification, and the modifications made#text#4

#Description and impact
*#Description and impact#*#secsep#1
desc#Description#Description of the vulnerability#text#15
indirect_impact#Ultimate Impact#Ultimate consequences of an attack \\
  exploiting the vulnerabilty by a threat agent#cclass:indirect_impact#1
direct_impact#Direct Impact#Rather the the ultimate impact of the \\
  vulnerability, the direct or immediate impact#cclass:direc_impact#1
impact_verbatim#Impact Text#Textual description of the impact of exploiting \\
  the vulnerability#text#7

#Threat
*#Identification of the Nature of Threat#*#secsep#1
thac_observe#Threat: Observe#The vulnerability can result in a user observing \\
  objects, data, etc., in violation of expected policy#cclass:yes_no#1
thac_destroy#Threat: Destroy#The vulnerability can result in a user destroying \\
  objects, data, etc., in violation of expected policy#cclass:yes_no#1
thac_modify#Threat: Modify#The vulnerability can result in a user modifying \\
  objects, data, etc., in violation of expected policy#cclass:yes_no#1
thac_create#Threat: Create#The vulnerability can result in a user creating \\
  objects in violation of expected policy#cclass:yes_no#1
thac_cavail#Threat: Availability#The vulnerability can result in the change of \\
  availability of  the system#cclass:yes_no#1
thac_disclose#Threat: Disclose#The vulnerability can result in the disclosure of \\
  information in violation of expected policy#cclass:yes_no#1
thac_misrep#Threat: Misreprsent#The vulnerability can result in \\
  misrepresentation of information#cclass:yes_no#1
thac_repudiate#Threat: Repudiate#The vulnerability can result in repudiation of \\
  information#cclass:yes_no#1
thac_integrity#Threat: integrity#The vulnerability can result in change of \\
  integrity of the system#cclass:yes_no#1
thac_conf#Threat: Confidentiality#The vulnerability can result in the loss of \\
  confidentiality of information#cclass:yes_no#1

# Information about the source of the information
*#Information Regarding the Source of the Information#*#secsep#1
source_addres#Source Detail#Detailed information on the source of the \\
  information. The WWW address, email address, books, etc. where the \\
  information was gathered from.#text#4

# System identification
*#System Identification#*#secsep#1
system#System(s)#System(s) vulnerable#list:system?c#4
system_version#System Version#System Version#text?c#4
```

```
system_vendor#System Vendor#System Vendor#list:vendor?c#4
system_verbatim#Misc System#Additional textual description of system#text?c#4
os_type#Type of OS#Type of operating systems affected#cclass:os_type#1

# Application information
*#Application Information#*#secsep#1
app#Application#Application that contains the vulnerability#list:application?c#4
app_version#Application Version#Application Version#text?c#4
app_verbatim#Verbose application#Long description of applications that contain \\
  vulnerabilities#text?c#5

# References
*#References#*#secsep#1
advisory#Advisory/ies#Advisory/ies that warn/describe about the \\
  vulnerability.#text#4
reference#References#References to the vulnerability in literature or in the \\
  net#text#4
related_docs#Related Docs.#Documents that describe the vulnerability, related \\
  to the vulnerability or that are useful in the analysis of the \\
  vulnerability#text#10

# Detailed analysis, detection techniques and fixes
*#Detailed Analysis, Detection Techniques, and Fixes#*#secsep#1
analysis#Analysis#A detailed analysis of the vulnerability#text?c#15
core_vulner#Core Vulnerability#If the vulnerability is in a piece of code, the \\
  smallest piece of code that still has the vulnerability#text?c#15
detection#Detection#Method of detecting that the vulnerability is being \\
  exploited#text?c#10
fix#Fix#A fix that can be used to eliminate the vulnerability.#text?c#10
test#Test#Method that can be used to detect whether the vulnerability is present \\
  in a system#text?c#10
workaround#Workaround#A temporary workaround for the vulnerability.  Used until \\
  a patch can be applied.#text?c#10
patch#Patch(es)#A patch or a series of patches that can be used to eliminate the \\
  vulnerability.#text?c#15

# Detailed information about exploitation
*#Detailed Information About Exploitation#*#secsep#1
exploit#Exploit Scripts#Reference to exploit scripts or programs#text?c#15
ease_of_exploit#Ease of Exploit#How easy is it to exploit the \\
  vulnerability#list:ease_of_exploit#3
idiot#IDIOT Pattern#IDIOT Pattern used to detect the exploitation of the \\
  vulnerability.#text#15
access_required#Access Required#What access is required for the \\
  exploitation#cclass:access_required#1
complexity_of_exploit#Complexity of Exploit#How complex is the exploitation of the \\
  vulnerability#cclass:complexity_of_exploit#1

# Source code and pointers to source code for the systems that contain the \\
  vulnerabilities.
*#System Sources#*#secsep#1
system_source#System Source#Source code or a pointer to the source code for the \\
  system that contains the vulnerability#text#7

# Classifications and features
*#Fault Classification#*#secsep#1
class#Aslam Classification#Aslam Classification. See the documentation for the \\
  possible values and an explanation.#cclass:classification?c#3
```

```
*#Category and Component Classification#*#secsep#1
category#System/Component#To what system or component does the vulnerability belong \\
  to#cclass:category#1

# Nature of vulnerability
*#Identification of Nature of the Vulnerability#*#secsep#1
nature_object#Object Affected#The object fundamentally affected by the \\
  vulnerability#list:nature_object?c#3
nature_effect#Effect on Object#The effect that the vulnerability has on the \\
  object#list:nature_effect?c#3
nature_method#Method#The method or means by which the object is \\
  affected#list:nature_method?c#3
nature_method_input#Type of Input#The type of input, if any, that leads to the \\
  effect#list:nature_method_input?c#3

# Verification of vulnerabilty.
# Although the verif field accepts any value, it is unlikely that it
# will ever be used as an enumeration.  Rather it is likely to be used
# as a boolean that indicates if this vulnerability was verified.
# However, it may be useful for humans reading the database to add the
# full name of the person or persons that verified the vulnerability.
*#Verification of Vulnerability#*#secsep#1
verif#Verified by#Person or entity that verified the vulnerability.  Verification \\
  should imply that the vulnerability is know to exist and had been exploited or \\
  verified by the person named.#text#3

#
# Policy features
*#Identification of Policy Violation#*#secsep#1
policyvio#Expected Policy Violated#Expected Policy Violated by the Vulnerability.  \\
  These policies need not be formally specified and are the expectation that users \\
  feel have been violated.#text?c#5

#
# The following fields are used to indicate future features that would be desirable
# for this record of the database
*#Identification of Environmental Factors#*#secsep#1
environment#Environment Features#What environmental conditions contribute to the \\
  vulnerability?  What assumptions are made about the environment that don't hold?  \\
  What about the environment makes this vulnerability possible?#text#10
features#Other Features#What other characteristics and features are relevant for the \\
  understanding of the vulnerability?#text#10
```

11 MIME Support in the Database

To support the inclusion of all kinds of data in the database, all of the text fields that do not contain classifiers can contain pointers to MIME encoded fields.

The complete list of MIME types possible is listed in section 12

12 MIME Types Defined for the Database

The MIME types listed in this section were compiled from the `exmh` MIME types file, a Java package that implemented some of the MIME functionality (The original author of the package is unknown), and the SUN `metamail` and `exmh` distributions.

MIME Type	File Extension		MIME Type	File Extension
application/activemessage			application/andrew-inset	
application/applefile			application/atomicmail	
application/dca-rft			application/dec-dx	
application/mac-binhex40			application/macwriteii	
application/msword			application/news-message-id	
application/news-transmission			application/octet-stream	.a
application/octet-stream	.arc		application/octet-stream	.bin
application/octet-stream	.dump		application/octet-stream	.exe
application/octet-stream	.gz		application/octet-stream	.hqx
application/octet-stream	.o		application/octet-stream	.saveme
application/octet-stream	.uu		application/octet-stream	.z
application/octet-stream	.bin		application/oda	.oda
application/pdf	.pdf		application/postscript	.ai
application/postscript	.eps		application/postscript	.ps
application/remote-printing			application/rtf	.rtf
application/rtf	.rtx		application/slate	
application/wita			application/wordperfect5.1	
application/x-IslandDraw			application/x-IslandWrite	
application/x-answerbook			application/x-bcpio	.bcpio
application/x-colorchooser			application/x-compress	
application/x-cpio	.cpio		application/x-csh	.csh
application/x-default-app			application/x-dos	
application/x-dvi	.dvi		application/x-fontedit	
application/x-frame			application/x-gtar	.gtar
application/x-guide			application/x-hdf	.hdf
application/x-latex	.latex		application/x-lotus-123	
application/x-makefile			application/x-mif	.mif
application/x-netcdf	.cdf		application/x-netcdf	.nc
application/x-patch	.patch		application/x-script	
application/x-sh	sh		application/x-shar	.sh
application/x-shar	.shar		application/x-shell-script	
application/x-sun-executable			application/x-sun-prog	
application/x-suncalendar			application/x-sundraw	
application/x-sunwrite			application/x-sv4cpio	.sv4cpio
application/x-sv4crc	.sv4crc		application/x-tar	.tar
application/x-tcl	.tcl		application/x-tex	.tex
application/x-texinfo	.texi		application/x-texinfo	.texinfo
application/x-troff	.roff		application/x-troff	.t
application/x-troff	.tr		application/x-troff-man	.man
application/x-troff-me	.me		application/x-troff-ms	.ms
application/x-troff-msvideo	.avi		application/x-ustar	.ustar
application/x-wais-source	.src		application/x-wais-source	.wsrc
application/zip	.zip		audio/basic	.au
audio/basic	.snd		audio/x-aiff	.aif
audio/x-aiff	.aifc		audio/x-aiff	.aiff
audio/x-sunaudio			audio/x-wav	.wav
content/unknown			image/gif	.gif
image/ief	.ief		image/jpeg	.jfif

MIME Type	File Extension		MIME Type	File Extension
image/jpeg	.jfif-tbnl		image/jpeg	.jpe
image/jpeg	.jpeg		image/jpeg	.jpg
image/tiff	.tif		image/tiff	.tiff
image/x-atk			image/x-brush	
image/x-cmu			image/x-cmu-rast	.ras
image/x-fits			image/x-fs	
image/x-g3			image/x-gould	
image/x-hips			image/x-ilbm	
image/x-img			image/x-imgw	
image/x-lispm			image/x-mac	
image/x-mgr			image/x-mtv	
image/x-pbm			image/x-pcx	
image/x-pgm			image/x-photocd	
image/x-pi1			image/x-pi3	
image/x-pict			image/x-pj	
image/x-portable-anymap	.pnm		image/x-portable-bitmap	.pbm
image/x-portable-graymap	.pgm		image/x-portable-pixmap	.ppm
image/x-ppm			image/x-qrt	
image/x-raw			image/x-rawg	
image/x-rgb	.rgb		image/x-rgb3	
image/x-sld			image/x-spc	
image/x-spu			image/x-sun-icon	
image/x-sun-raster			image/x-tga	
image/x-tiff			image/x-xbitmap	.xbm
image/x-xbm			image/x-xim	
image/x-xpixmap	.xpm		image/x-xpm	
image/x-xwd			image/x-xwindowdump	.xwd
image/x-ybm			image/x-yuv	
message/external-body			message/news	
message/partial			message/rfc822	.mime
multipart/alternative			multipart/appledouble	
multipart/digest			multipart/mixed	
multipart/parallel			text/html	.htm
text/plain	.c		text/plain	.c++
text/plain	.cc		text/plain	.h
text/plain	.java		text/plain	.pl
text/plain	.text		text/plain	.txt
text/richtext	.rtx		text/tab-separated-values	.tsv
text/x-setext	.etx		video/mpeg	.mpe
video/mpeg	.mpeg		video/mpeg	.mpg
video/quicktime	.mov		video/quicktime	.qt
video/x-msvideo	.avi		video/x-sgi-movie	.movie
video/x-sgi-movie	.mv			

13 Considerations on the Inner Workings of the Java Interface

- The source code for the database is has sufficient comments to be easily understood. The `docs` directory in the *JAVAGUIDEV* contains the Javadocs documentation for all the classes of the database GUI. The main Java file is `VulnerabilityDatabase.java`.
- In addition to all the source code developed at COAST, the database uses the PerlTools classes for implementing

pattern matching[4]. The MIME routines were developed on top of the Cryptix-Java V2.2 sources[5], however, we only used the MIME packages of that distribution and hence there is no crypto code in our program.

- All the site-dependencies have been centralized in a single class called `vdbGlobals.java`.
- The Java GUI uses two perl programs to MIME encode large files and export records to a multipart MIME file. These are the `encodeMIMEFile.pl` and `generateMIMEMultiPart.pl`. These perl programs are not system specific and hence should be completely portable. They receive from the Java GUI all the information they need to perform their task.

14 Classifiers Defined for the Database

In addition to the classifiers described in section 7, the following classifiers are defined for the database:
Feature: ***Yes/No Classifier***

- `yes` → Yes
- `no` → No
- `NA` → Does not apply
- `?` → Unknown

Feature: ***Application***

This feature defines the application that has the vulnerability. This classifier is relevant for those vulnerabilities that are present in user-level programs, daemons, servers, etc. that are not a part of the operating system itself. This feature can take on many values and here we give a small subset as examples.

- `Netscape` → Netscape WWW Browser
- `HotJava` → SUN's HotJava WWW Browser
- `JDK_appviewer` → Java Developer Kit's appler viewer
- `Ora_pbrow` → Oracle PowerBrowser
- `XMCD` → CD digital audio player utility for X11/Motif
- `NIS` → Network Information System
- `Apache` → Apache WWW `httpd`
- `FrontPage` → Microsoft FrontPage
- `InternetExplorer` → Microsoft Internet Explorer
- `NetscapeNewsServer` → Netscape's News Server
- `Minicom` → Linux free `telecom` program
- `NTHTTPServer` → HTTP Server included in Windows NT
- `rpcbind` → Universal addresses to RPC program number mapper
- `rlogin` → Remote login
- `stat` → File status
- `ftpd` → Internet File Transfer Protocol server (`ftpd`)
- `talkd` → Server for talk program (`talkd`)
- `ps` → Report process status: `ps`
- `rmail` → Read mail program in Unix
- `lpr` → Unix `lpr` - Send a job to the printer
- `ircd` → IRC Server
- `NCSAhttpd` → NCSA WWW httpd
- `pkgtool` → PKGTOOL Linux Software Management Utility
- `sysdiag` → HP System Diagnostics tool
- `majordomo` → Majordomo mailer
- `passwd` → Unix password change utility
- `binmail` → `/usr/bin/mail` on Unix
- `rdist` → Remote file distribution client
- `ppp` → Implementation of the Point-to-point protocol for TCP/IP
- `sperl` → SetUID Perl
- `xterm` → Terminal emulator for X
- `cxterm` → Chinese Terminal emulator for X
- `admintool` → Sun administration tool
- `inperson` → InPerson desktop video conferencing package
- `lynx` → Lynx text web browser
- `swinstall` → HPUX software installation utility
- `glance` → HPUX Glance software
- `workman` → Workman CD digital audio player
- `lpd` → Line printer daemon

[4] See `http://www.oroinc.com/`
[5] See `http://www.systemics.com/docs/cryptix/`

- `sendmail` → Un*x program for sending email over the Internet
- `lmgrd` → FLEXlm license manager daemon
- `expreserve` → `vi` and `ex` file preservation utility
- `ld.so` → run-time linker used by dynamically linked executables (`a.out`)
- `fm_fls` → FrameMaker license server
- `vold` → Solaris volume mounting daemon
- `kcms` → Kodak Color Management System
- `wuftpd` → Washington University ftpd
- `rpcmountd` → `rpc` mount daemon
- `df` → Disk space reporting command
- `ordist` → IRIX version of `rdist`
- `pset` → IRIX processor set modification utility
- `chkey` → RPC change key utility
- `cdplayer` → SGI CD digital audio player
- `fpkg2swpkg` → HPUX product spec. conv. utility
- `test-cgi` → A script that returns the status of the `cgi` systems on `http` daemons
- `crontab` → System clock daemon manager for users
- `website` → Website Commercial NT/95 web server
- `telnetd` → DARPA TELNET protocol server
- `norton` → Norton Utilities
- `usrmgr` → NT user manager
- `mstimeserv` → NT Time Server
- `msaccess` → Microsoft Access
- `msoffice` → Microsoft Office
- `innd` → Internet News daemon
- `abuse` → Abuse game
- `at` → Unix Job Scheduler command
- `dip` → Dial up IP program for Linux
- `newgrp` → Program to create a new group
- `bash` → GNU Project's Bourne Again SHell
- `BIND` → Berkeley Internet Name Domain
- `elm` → The Elm Mail System
- `NA` → Does not apply

Feature: ***Ease of Exploit***

This classifier was originally defined from a talk given by Tom Longstaff [Lon97] and attempts to identify how easy (or how hard) it is to exploit the vulnerability.

- `simple` → Simple command
- `toolkit` → Toolkit available
- `expertise` → Expertise required
- `user` → Must convince a user to take an action
- `administrator` → Must convince an administrator to take an action

Feature: ***Vendor***

This classifier is used to identify the vendor of the systems or that the vulnerability is present on.

- `SUN` → Sun Microsystems, Inc.
- `Microsoft` → Microsoft
- `SGI` → Sillicon Graphics Inc.
- `Netscape` → Netscape Corporation
- `BSDI` → Berkeley Software Design, Inc.
- `Slackware` → Walnut Creek CDROM
- `Redhat` → Redhat Software, Inc.
- `Debian` → Software in the Public Interest (SPI)
- `MkLinux` → Apple Computer
- `DGC` → Data General Corporation
- `FreeBSD` → FreeBSD, Inc
- `HP` → Hewlett-Packard Company
- `IBM` → IBM Corporation
- `NEC` → NEC Corporation
- `SCO` → The Santa Cruz Operation, Inc.
- `NeXT` → NeXT Software, Inc.
- `OpenGroup` → The Open Group
- `SantaCruz` → The Santa Cruz Operation (SCO)
- `Caldera` → Caldera
- `DEC` → Digital Equipment Corporation
- `Apple` → Apple Computer
- `OSF` → Open Software Foundation
- `CRAY` → Cray
- `NetBSD` → The NetBSD Project
- `OpenBSD` → The OpenBSD Project
- `Novell` → Novell
- `NA` → Does not apply

Feature: ***System***

This classifier is used to indicate the systems that are known (to us!) to have the vulnerability. To date we have recorded vulnerabilities for the following operating systems.

- `Solaris` → SUN Solaris
- `SunOS` → SUN OS
- `DOS` → Microsoft DOS
- `Windows95` → Microsoft Windows 95
- `WindowsNT` → Microsoft Windows NT
- `WindowsWG` → Microsoft Windows (pre-95)
- `Slackware` → Linux Slackware Distribution
- `Redhat` → Linux Redhat Distribution
- `Debian` → Linux Debian Distribution
- `MkLinux` → Linux Apple Distribution
- `OpenLinux` → Linux Caldera Distribution
- `OtherLinux` → Unknown, unsuported, or uncommon Linux Distribution
- `BSDI` → BSDI Unix
- `NovellUnix` → Novel UnixWare
- `NetBSD` → NetBSD Unix
- `FreeBSD` → FreeBSD Unix
- `Athena` → MIT-distributed Athena
- `Cygnus` → Cygnus Network Security
- `OpenVision` → openVision
- `SGIRIX` → SGI IRIX
- `DECOSF1` → Digital OSF/1
- `NECUX` → NEC XX-UX
- `HP-UX` → Hewlett-Packard Unix
- `AIX` → IBM's AIX
- `OpenStep` → OpenStep
- `OSF` → OSF
- `Caldera` → Caldera
- `Goah` → NEC's Goah
- `Ultrix` → Ultrix
- `DEC_UNIX` → Digital Unix
- `AUX` → Apple's Unix
- `DG` → Data General
- `unicos` → Cray's UNICOS
- `OpenBSD` → OpenBSD Unix
- `MacOS` → Macintosh OS (MacOS)
- `Netware` → Novell Netware
- `VMS` → DEC VMS
- `NA` → Does not apply

15 MIME Inclusion: `povide_buffer_overflow.MIME`

.

The text of this function corresponds to the text that would be included as a MIME file called `povide_buffer_overflow.MIME`.

There is a class of computer vulnerabilities that is commonly called "buffer overflows" that is difficult to characterize and define. There are many variations of these but they essentially have one of the forms shown below. A program tries to copy some data from one object into another, does not check that the destination object is large enough to contain the source object, and uses a routine such as `sprintf` to do the copying.

Line	Form 1	Form 2	Form 3
1	`main(int ac,`	`main() {`	`main() {`
2	`  char *av[]) {`	`  p();`	`  p();`
3	`  p(av[1]);`	`}`	`}`
4	`}`		
5		`void p(){`	`void p(){`
6	`void p(char *a){`	`  char b[30];`	`  struct hostent *h;`
7	`  char b[30];`	`  char *p;`	`  sockaddr_in s;`
8			
9	`  strcpy(b,a);`	`  p = getenv("TERM"));`	`  h = gethostbyname(*host);`
10	`}`	`  sprintf(b,"%s",p);`	`  bzero(&s, sizeof s);`
11		`}`	`  s.sin_family =`
12			`  h->h_addrtype;`
13			`  s.sin_port = 25;`
14			`  bcopy(h->h_addr_list[0],`
15			`  &s.sin_addr,`
16			`  h->h_length);`[6]
17			`}`

Normally `h->h_length` would be the same size as `h->h_addr_list[0]`. However, it is possible to create a (possibly fake) DNS reply that will violate this assumption.

However, not all program that share this characteristic are vulnerable. The programs shown below all have buffer overflows but are not vulnerable because either the function never returns–in which case the program never has the opportunity to jump to the code inserted–or the program's buffer is declared static–in which case the program overruns the heap and not the stack.

Line	Form 1	Form 2
1	`main(int ac,`	`main() {`
2	`  char *av[]) {`	`  p();`
3	`  p(av[1]);`	`}`
4	`}`	
5		`void p(){`
6	`void p(char *a){`	`  static char b[30];`
7	`  char b[30];`	`  char *p;`
8		
9	`  strcpy(b,a);`	`  p = getenv("TERM"));`
10	`  exit(1);`	`  sprintf(b,"%s",p);`
11	`}`	`}`

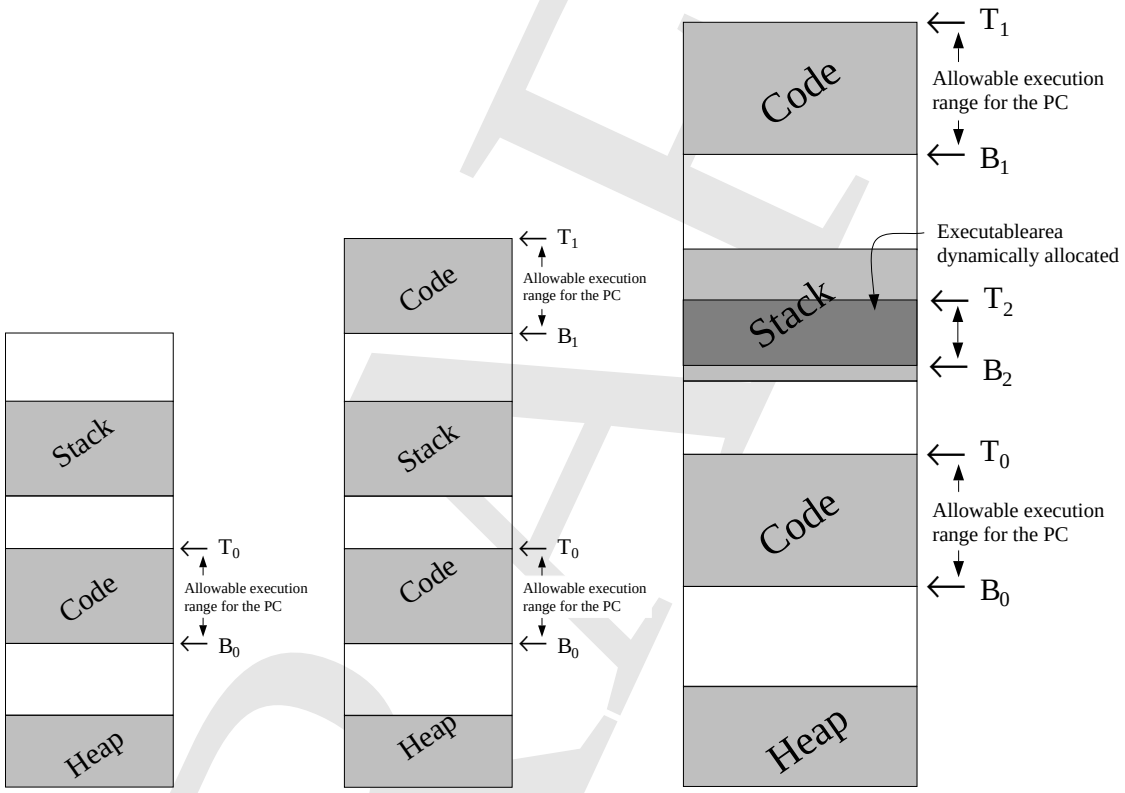
Programs and programmers implicitly make assumptions about the environment the code will execute under. For most architectures, programmers make the implicit assumption that programming counter (PC) will execute the code provided and not anything else. Figure 19 illustrates the ranges where the PC can execute in the cases of non-fragmented code segments, fragmented code segments, and fragmented code segments with dynamic loading of code[7].

The so-called "buffer overflow" vulnerabilities are instances of programs where, as shown in figure 20, the programming counter jumps from an allowable executable region to a region in memory, normally the stack, where it executed some arbitrary code. Because there are instances of buffer overflows that do cannot be characterized as vulnerabilities, the real issue behind these vulnerabilities is not the buffer overflow but rather what happens when a user can cause the stack pointer to change so that it points back at the stack.

The program shown below illustrates how a program that can provide an attacker an index into arbitrary stack memory can be vulnerable to the same problem without overwriting the program's local memory or altering anything else than the return address in the stack and the portion of memory that will be used to store the code to be executed. The program segment was extracted from a project for graduate operating system course and it's function is to allow the programmer to change the value of debugging flags without having to recompile the code.

```
main() {
```

[7]This simplified model does not take into account the system area of memory and this can be incorporated by adding an additional set of segment market to signal that it is OK for the PC to execute system memory.



A program normally contains three segments that can be distinguished: heap, code, and stack. We expect the PC to remain in the area between the B_0 and T_0 markers.

If the code segment is fragmented, the program counter is expected to remain in the areas between the B_0, T_0, B_1, and T_1 markers.

Dynamic code can be loaded into the stack and the program explicitly (by calling a function that dynamically links the code) creates another area where the PC is allowed to execute: the are between the B_2 and T_2 markers.

Figure 19: Programs normally execute code from well defined regions in memory, even if the memory is fragmented or the program contains dynamic executable code.

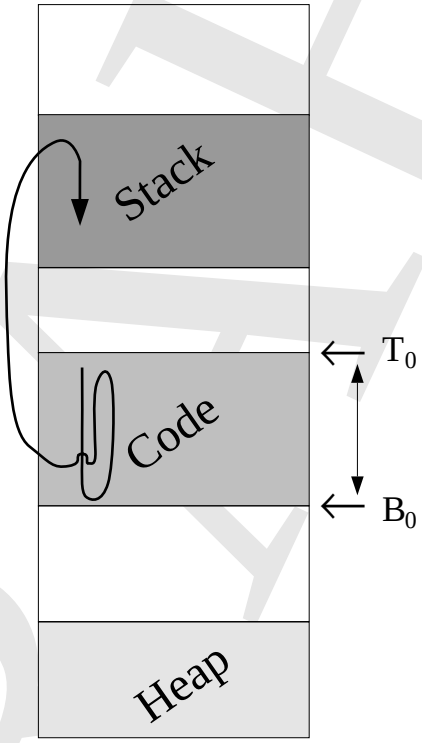


Figure 20: Buffer overflows are among the vulnerabilities where the programming counter (PC) jumps from an allowable region to the stack to execute arbitrary code.

```
  int  dbg1, dbg2, dbg3, dbg4;
  int  numiter,index, i, j;
  FILE *fp;

  /* Read from a file the values
     of the debugging variables
     as a series of (position, value)
     pairs: (1,5) would set dbg1 to 5
     and (4,0) would clear the dbg4
     flag. */
  if((fp = fopen("conf","r"))!=NULL){
    /* How many flags to change? */
    fscanf(fp,"%d",&numiter);
    printf("numiter = %d\n",numiter);
    for(i=0;i<numiter;i++) {
      fscanf(fp,"%d%d",&index,&j);
      *(&dbg1-index+1) = j;
    }
  }
}
```

The expected policy violated by this sample code, and all of the "buffer-overflow" vulnerabilities we have seen to date, is that the program's PC should remain within the allowable range. A policy specification for this case can now be generated with the the model and the notation described in [KT].

For simplicity, an atomic operation will be axiomatically defined as the execution of any instruction that causes the program counter (PC) to move. Note that, however, an alternative definition could consider only those operations that cause the program counter to jump or we could consider only those operations that cause a program to return from a subroutine. Although less general, these definitions are just as effective and much more practical.

Or policy function takes as arguments a system value function, an object value function, and two sets of interest (before and after the execution of an instruction). The function returns `true` if the policy has not been violated and `false` otherwise.

$$
\begin{array}{l}
\textit{Policy} : \textit{System Value function} \times \textit{Object Value function} \times \\
\qquad \textbf{set of interest} \times \textbf{set of interest} \rightarrow \textbf{boolean} \\
\textbf{fun} \quad \textit{Policy}\,(\textit{Value}, v, I_i, I_{i+1}) ::= \\
\qquad \textbf{if } \textit{Value}\,(I_i, v) \leq \textit{Value}\,(I_{i+1}, v) \quad \textbf{then} \\
\qquad\qquad \textit{Policy} := \text{true}; \\
\qquad \textbf{else} \\
\qquad\qquad \textit{Policy} := \text{false}; \\
\qquad \textbf{fi} \\
\textbf{nuf}
\end{array}
\tag{1}
$$

The system value function is an aggregation of the values of the objects in the system:

$$
\begin{array}{l}
\textit{Value} : \textbf{set of interest} \times \textit{Object Value function} \rightarrow \textbf{integer} \\
\textbf{fun} \quad \textit{Value}\,(S, v) ::= \\
\qquad \textit{Value} := \sum_{x \in S} v\,(x, S - x) \qquad \forall x \in S; \\
\textbf{nuf}
\end{array}
\tag{2}
$$

The policy we will specify requires that applications only execute instructions within the bounds defined. The set of interest consists of programs, program counters, and boundaries:

Programs:

- ♢ Set of boundaries: b.
- ♢ Set of program counter locations: pc.

Boundary:

$\diamondsuit$ Top of allowed segment: T.
$\diamondsuit$ Bottom of allowed segment: B.

Program Counter:
$\diamondsuit$ Location: l.

The value function that can be used to implement the desired policy is:

v : **object of interest** $\times$ **set of object of interest** $\rightarrow$ **integer**
fun $v\,(o, S)$ ::=
$v := 0;$
if o is a program **then**
$\forall x \in o.pc$ **do**
$m := 0;$
$\Rightarrow$ *Check to see if the PC is is a correct range*
$\forall y \in o.b$ **do** (3)
$m := 1$ **if** $x.l \geq y.B \wedge x.l \leq y.T;$
od
$\Rightarrow$ *Violation if we did not find a valid range for the PC*
$v := v - 1$ **if** $m = 0;$
od
fi
nuf

ADDENDUM:

There are a class of vulnerabilities that result from buffer overflows that cannot be caught by the violation of the policy specified in this document. We will show next two such vulnerabilities—at this point theoretical because we have no evidence that these actually exist in released systems

The first program declares all it's variables to be `static` and hence cannot have a buffer overflow that overwrite the stack. However, the execution path of the program can be changed to execute code that would not be executed under normal circumstances:

```
main() {
  /* Static variables. Can't inject anything into the stack */
  static char name[10];
  static char term[5];
  static char userID[10];

  /* Do something that will determine the name and userID
    of the user */
  strcpy(userID,"krsul");
  strcpy(name,"ivan");

  /* The program needs to know the terminal type... so lets read
     it from the environment variable */
  /* BUFFER OVERFLOW HAPPENS HERE! */
  strcpy(term,getenv("TERM"));

  /* Now that we know the terminal... */
  if(strcmp(userID,"root")==0) {
     /* Do something super restricted or secret */
     do_secret(term);
  } else {
     /* Print error message telling the user that
        he/she does not have access */
     print_error(term);
  }
}
```

If the environment variable "TERM" is set to the value "vt100root" then the program will execute the function `do_secret` regardless of the original value of the variable `userID`. The program counter remains in the area allowed by design but violates the semantic of the program specification.

The second program[8] is an example of a program that can modify only the return address on the stack to return to a different part of the program than expected:

```
main() {
  f(1);
  printf("1");
  printf("2");
  printf("3");
  printf("4\n");
}

int f(int i1) {
  int i;

  for(i=0;i<2;i++)
    *(&i1-i) += 13;
}
```

The execution of the program results in the string "234" rather than the expected string "1234". The program has *skipped* a statement by changing the return address of the function `f()`. If the attacker can calculate the offset from the original return of the function to an arbitrary point in the code, and the offset change (the value 13), can be provided by the attacker, then the program will effectively *jump* to that portion of code.

We argue that, although these vulnerabilities are the result of buffer overflows, they belong to a different class of vulnerabilities because the attacker cannot inject arbitrary code that will be executed by the program. Rather, the attacker can just cause existing code to be executed in a different order that specified in the program design.

16 Acronyms

COAST Computer Operations, Audit, and Security Technology. COAST is a multiple project, multiple investigator laboratory in computer security research in the Computer Sciences Department at Purdue University.

OS Operating System. In a computer system, the software that controls processing, manages resources, and communicates with external devices like disks and printers is sometimes referred to the *executive*, *monitor*, *task manager*, or *kernel*. For all of these we can use the broader term Operating System (OS) [Com84].

SETUID Set-User-ID. Processes in UNIX can assume the identity of the user that owns an executable file that has the Set-User-ID (SETUID) bit set. Such executable files are said to be SETUID programs.

MIME Multipurpose Internet Mail Extensions. MIME defines a format and general framework for the representation of a wide variety of data types in Internet mail so that the body of messages, encoded as flat US ASCII, can include textual messages in character sets other that US ASCII and non-textual messages [FB96].

ASCII American Standard Code for Information Interchange A standard data transmission code that was introduced to achieve compatibility between devices [LS90].

GUI Graphical User Interface. A program user interface that includes non textual-only graphical information such as windows, menus, buttons, etc.

[8] Thanks to Wenliang Kevin Du of the COAST laboratory for this example

17 Acknowledgments

Many people at COAST contributed to the development of the COAST vulnerability database. Tom Daniels and Adam Wilson contributed in the development and data entering phases; Wenliang Du, Tugkan Tuglular, and Diego Zamboni contributed with comments and feedback. Eugene Spafford, the director of the COAST laboratory, gave the project perspective and direction.

References

[Aud95] Robert Audi, editor. *The Cambridge Dictionary of Philosophy*. Cabridge University Press, 1995.

[Com84] Douglas Comer. *Operating System Design: The XINU Approach*. Prentice Hall, 1984.

[FB96] N. Freed and N. Borenstein. RFC 2045: Multipurpose Internet Mail Extensions. (MIME) Part One: Format of Internet Mssage Bodies, November 1996.

[KSa] Ivan Krsul and Eugene Spafford. On taxonomies and classifications for computer security applications. Under review by Spaf.

[KSb] Ivan Krsul and Eugene Spafford. The Feature Selection Problem for the Applicaton of Machine Learning Algorithms to the Analysis of Computer Vulnerabilities. Under review by Spaf.

[KT] Ivan Krsul and Tugkan Tuglular. An economic model for modeling computer policies. Under review by Spaf.

[Lon97] Tom Longstaff. Update: CERT/CC Vulnerability Knowledgebase. Technical presentation at a DARPA workshop in Savannah, Georgia, February 1997.

[LS90] Dennis Longley and Michael Shain. The Data and Computer Security Dictionary of Standards, Concepts, and Terms, 1990.

[Pow96] Richar Power. Current And Future Danger: A CSI Primer of Computer Crime & Information Warfare. CSI Bulletin, 1996.