SECURATE—Security evaluation and analysis using fuzzy metrics*

LANCE J. HOFFMAN
The George Washington University
Washington, DC

and

ERIC H. MICHELMAN
Massachusetts Institute of Technology
Cambridge, Massachusetts

and

DON CLEMENTS
Teknekron, Inc.
Berkeley, California

INTRODUCTION

In the last several years, methods for controlling security in computer systems have become widely known. While in 1972 there were only one or two books and three bibliographies on computer security and privacy, in 1977 there were at least 15 books and six bibliographies on the topic. Expanding public concern with the problem is evidenced by a great deal of federal, state and local legislation. 1,2 Governmental agencies such as the National Bureau of Standards, the Defense Department, and the National Science Foundation are all sponsoring research efforts in the area, as have some private manufacturers. 3-5

One consequence of this work is that a significant part of the computing community is now aware of techniques for maintaining security in computer systems. Unfortunately, the question of how to measure the costs and effectiveness of the various security methods is still largely unexplored. Only recently has there been any reliable work done on costs of privacy transformations or authentication methods, and researchers have only scratched the surface in investigating metrics for security systems. 7 While some more formal work has begun, 8-13 there has so far been very little useful application of this theoretical work to practical security decisions. One exception is a privacy cost model which has been recently introduced. 14 This work assesses the impact of various privacy regulations upon data processing installations which handle personal information. Goldstein discusses conversion costs to meet privacy requirements and the "privacy increment" to installation operational costs. He does not, however, deal directly with the effectiveness of security techniques.

We will address the question of security effectiveness. We here describe SECURATE, a computer installation security evaluation and analysis system, which is based upon a model of a computer installation as a set of triples composed of objects, threats, and security features. SECURATE uses security rating functions based on fuzzy set theory to provide security ratings for an installation as a whole as well as for subsections. It helps to determine weak and strong points and facilitates the comparison of alternative security designs. SECURATE is not meant to be a substitute for a human decision maker.

After the system description is complete, the user is asked to specify his or her outlook on security; some view total system security as an all-or-nothing thing, while others see it as dependent in various ways on a system's individual components. Based upon the user's outlook and the system description, SECURATE supplies evaluation functions which return natural-language security ratings of the system or its subsystems. The fifth section describes these evaluation functions.

After the system was implemented in APL, it was used at

---

* Work supported in part by Grant MCS76-09214 from the National Science Foundation and by the University Committee on Research, The George Washington University.
seven installations by students who were doing risk analyses of the installations. Feedback from this initial group of users is discussed in the seventh section.

TECHNICAL BASIS

The technical basis for SECURATE is in earlier work\(^{16}\) which defines a security system as a set of objects, each with a loss value; a set of threats, each with a likelihood; and a set of features, each with a resistance. This work addresses the problem of imprecision in the approximation of values, likelihoods, and resistances by using linguistic variables\(^{16}\) to combine the values of these variables into security ratings.

Clements’ model groups resources within computing systems which are vulnerable to some security threat as the set \(O\) of security objects. Each object in the set possesses a loss value to its owner.

Associated with each security object \(O_i\) is a number of potential intrusion activities; all of these together form the set \(T\) of security threats. Each threat \(T_i\) has associated with it a likelihood of occurrence.

The object-threat relations form a bipartite directed graph (Figure 1) in which edge \((T_i, O_j)\) exists only if threat \(T_i\) is a viable means of compromising object \(O_j\). The relations of threats to objects is not one to one; a threat may compromise any number of objects and an object may be vulnerable to more than one threat.

Finally, there is the set \(F\) of security features. A security feature performs a “firewall function” by presenting some degree of resistance to a penetration attempt. The set of security features transforms the bipartite graph of Figure 1 into the tripartite graph of Figure 2. In a “protected” system all edges of the form \((T_i, F_k, O_j)\) exist and \((T_i, F_k, O_j)\) identifies an unprotected object. Each triplet \((T_i, F_k, O_j)\) in \(T \times F \times O\) contributes to the overall security rating of a system, as we shall see below.

In attempting to specify the object values, threat likelihoods, and feature resistances, one is confronted with the problem of imprecision. In evaluating a computer system’s security, one generally relies on imprecise human judgment to provide approximate measures of the effectiveness of security features. The problem is aggravated when one attempts to produce security ratings for entire systems from these measures. The assignment of a numerical security rating is inconsistent with the complexity of data processing installations and the imprecision of the underlying measures.

Clements suggested that it is possible to make meaningful measurements of the security of a computer system through the use of linguistic variables—variables which assume values which are words rather than numbers. Using this approach, the object values, threat likelihoods, and feature resistances (as well as the resultant security rating) are specified in terms such as high, low, and medium. Appropriate modifiers provide finer resolution by allowing terms such as very high, somewhat low, etc.

The user assigns linguistic values (high, medium, very high, etc.) to the component variables (loss value, likelihood, resistance) for each (object, threat, feature) triplet in the system. These measures determine the contribution of that particular triplet to total system security. How this is done is discussed in a later section.

Each linguistic variable is a fuzzy set whose members are real numbers in the interval \([0,1]\). These values comprise the compatibility function, \(\mu\), for the specific linguistic variable. For example, if \(\mu_{\text{high}}(0.8) = 0.9\), then 0.9 represents the degree to which a non-fuzzy rating of 0.8 on a base scale agrees with a fuzzy rating of high. Figure 3 illustrates what the complete compatibility functions for typical high and very high terms might be. More detail on linguistic variables, base scales, and compatibility functions can be found in Reference 16.

OBJECT HIERARCHY AND LINGUISTIC TERMS

SECURATE provides a “default” hierarchical structure of objects commonly found in computer installations.\(^{17}\) A small part of this object hierarchy and the corresponding threats and features are presented in the appendix.

There are two phases involved in using the system: (1) typing in a description of the installation and (2) using the security analysis functions.

Object value, threat likelihood, and feature resistance for each security point of interest are first specified by the user in terms of linguistic variables. The vocabulary and Backus-
Naur form syntax of the rating language, along with examples, are shown in Figure 4.

Once the installation to be analyzed is described in terms of these triples, the functions described in the fifth section are invoked by the user to evaluate and analyze its security.

THE USER INTERFACE

From the start of the SECUrate project, an important objective was to design and implement the system so that it would be as hospitable to the users as possible. Our goals concerning user-oriented features were primarily to keep the system simple, easy to use, and non-tedious. More specifically, we were concerned with the following points:

(a) User Understanding—for obvious reasons, achieving adequate user understanding is very important. Not only won’t the system be useful if the user doesn’t understand it, but it won’t be used.

(b) Simple, Non-tedious Interface—a similar, much simpler system had been developed earlier by a student as a term project. A unanimous criticism of that system was that it took too long to use and the data entry was too tedious. As our system was to require considerably more information, it seemed important to keep the interaction as short, concise, and painless as possible.

(c) Useful Evaluation Functions—while it may seem that this is the most important point, it may actually be the least. A system which a user understands and is comfortable using is more likely to be used and be helpful than a system that doesn’t possess these qualities, even if the evaluation functions provided by the first aren’t quite as useful as those provided by the second.

Since the object hierarchy was used as a guide for collecting user data, it provided a convenient basis for structuring the input. We drew up forms which corresponded in format exactly to what appeared on the screen. The user wrote down on the forms only the necessary information and then transferred it easily to the system. These forms, by integrating the system commands with the input data in a coherent way, familiarized the users with the system’s operation prior to their using it. Figures 5a and 5b show an example of the input form and the corresponding data entry. Further information concerning the user interface is given in Reference 18.

SECUrate assumes that the installation will be similar to that modeled by the hierarchy in Appendix A, although the user can modify the hierarchy appropriately as he or she supplies the triples information. SECUrate leads the user through the hierarchy (and any user modifications to it), providing the opportunity at each node to add offspring or specify triples. If a triple is specified for an object with offspring, it is assumed to refer to that object and each of its offspring. Objects, threats, and features appearing in more than one triple may have different values, likelihoods or resistances, respectively.

\[
\begin{align*}
\text{(sentence)} & ::= \text{(compound phrase)} | \text{(simple phrase)} \\
\text{(compound phrase)} & ::= \text{(conjunctive phrase)} | \text{(range phrase)} \\
\text{(simple phrase)} & ::= \text{(relational phrase)} | \text{(hedged primary)} \\
\text{(conjunctive phrase)} & ::= \text{(relational phrase)} \text{ AND (relational phrase)} \\
\text{(range phrase)} & ::= \text{(hedged primary)} \text{ TO (hedged primary)} \\
\text{(relational phrase)} & ::= \text{(composite relation) THAN (hedged primary)} \\
\text{(composite relation)} & ::= \text{(relation hedge) (relation) | (relation) (relation hedge) ::= NOT | MUCH | SLIGHTLY} \\
\text{(relation)} & ::= \text{LOWER | HIGHER} \\
\text{(hedged primary)} & ::= \text{(hedge) (primary) | (primary) | (fuzzy number)} \\
\text{(hedge)} & ::= \text{NOT | VERY | MOREORLESS | QUITE | PRETTY | SORTOF | REALLY | EXTREMELY} | \text{INDEED} \\
\text{(primary)} & ::= \text{LOW | HIGH | MEDIUM} \\
\text{(fuzzy number)} & ::= \text{(fuzzifier) (number)} \\
\text{(fuzzifier)} & ::= \text{ABOUT} \\
\text{(number)} & ::= 1 2 3 4 5 6 7 8 9 10
\end{align*}
\]

Figure 4a—BNF grammar of rating language

Figure 4b—Examples of terms available in rating language

From the collection of the Computer History Museum (www.computerhistory.org)
Once the triples are entered, they may be printed out using the DISPLAY function. For each triple this prints out: the triple number; the object name, number, and value; the threat name, number, and value; and the feature resistance. Figure 6 shows the DISPLAY output for a sample system. The information describing the installation is automatically saved and may be used later with repeated applications of the system.

EVALUATION FUNCTIONS

The general system structure is shown in Figure 7. There are several security evaluation functions implemented. Each produces an evaluation based upon the section of the system being considered and upon a given outlook on security.

The possible sections of a system which can be evaluated are the following:

(a) Overall System—This function returns a security rating for the entire installation. That is, it rates the entire set of triples.

(b) All subsections of a Section—with either the top level of the installation hierarchy or one of the subsections specified, this function returns an individual rating for each subsection at the next lower level. For example, if in our sample system, the top level of the hierarchy was specified for a sectional analysis, security ratings would be printed out for each of the following subsections: hardware, software, and the computer center (Figure 8).

(c) Individual Subsection—a security rating is returned for a specified subsection of the installation. Only triples for that subsection (including offspring) are considered (see Figure 9).

In choosing an outlook, the user in effect describes how he or she views security. Once an outlook is chosen, it stays in effect for all of the evaluation functions until it is respecified.

The outlooks are the following:

(a) Weakest Link—this will look for the weakest feature resistance and return that as the security rating. The theory here is that the system is only as secure as its weakest link (Figure 10).

(b) Selected Weakest Link—this produces a weakest link rating based on those triples which satisfy the condi-
FOLLOWING IS A LIST OF OBJECTS ADDED, THEIR ASSIGNED OBJECT NUMBERS, AND THEIR PARENT IN THE HIERARCHY:

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>OBJECT NO</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>METERING EQUIPMENT</td>
<td>71</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TRIPLE NO</th>
<th>OBJECT</th>
<th>NUMBER</th>
<th>NAME</th>
<th>VALUE</th>
<th>NUMBER</th>
<th>NAME</th>
<th>LIKELIHOOD</th>
<th>RESISTANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>METERING EQUIPMENT</td>
<td>11</td>
<td>CENTRAL MACHINE</td>
<td>VERY HIGH</td>
<td>8</td>
<td>UNAUTHORIZED USE</td>
<td>MEDIUM</td>
<td>PRETTY HIGH</td>
</tr>
<tr>
<td>2</td>
<td>METERING EQUIPMENT</td>
<td>11</td>
<td>CENTRAL MACHINE</td>
<td>VERY HIGH</td>
<td>10</td>
<td>HUMAN ERROR</td>
<td>PRETTY LOW</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>STORAGE MEDIA</td>
<td>12</td>
<td>CENTRAL MACHINE</td>
<td>HIGH</td>
<td>13</td>
<td>UNAUTHORIZED READ</td>
<td>HIGH</td>
<td>PRETTY LOW</td>
</tr>
<tr>
<td>4</td>
<td>STORAGE MEDIA</td>
<td>12</td>
<td>CENTRAL MACHINE</td>
<td>HIGH</td>
<td>11</td>
<td>THEFT</td>
<td>MEDIUM</td>
<td>PRETTY LOW</td>
</tr>
<tr>
<td>5</td>
<td>METERING EQUIPMENT</td>
<td>71</td>
<td>HARDWARE TAMPERING-MODIFIED</td>
<td>LOW</td>
<td>4</td>
<td>SYSTEM</td>
<td>HIGH</td>
<td>PRETTY LOW</td>
</tr>
<tr>
<td>6</td>
<td>PROGRAMS</td>
<td>22</td>
<td>UNAUTHORIZED USE</td>
<td>MEDIUM</td>
<td>46</td>
<td>INADEQUATE DEBUGGING</td>
<td>FAIRLY HIGH</td>
<td>PRETTY LOW</td>
</tr>
<tr>
<td>7</td>
<td>DATA</td>
<td>23</td>
<td>UNAUTHORIZED USE</td>
<td>HIGH</td>
<td>20</td>
<td>UNSECURED STORAGE MEDIA</td>
<td>HIGH</td>
<td>PRETTY LOW</td>
</tr>
<tr>
<td>8</td>
<td>DATA</td>
<td>23</td>
<td>EXPOSED OUTPUT</td>
<td>HIGH</td>
<td>33</td>
<td>DATA PREPARATION ERRORS</td>
<td>HIGH</td>
<td>PRETTY LOW</td>
</tr>
<tr>
<td>9</td>
<td>DATA</td>
<td>23</td>
<td>DATA PREPARATION ERRORS</td>
<td>HIGH</td>
<td>43</td>
<td>DATA PREPARATION ERRORS</td>
<td>HIGH</td>
<td>PRETTY LOW</td>
</tr>
</tbody>
</table>

Figure 6—Display of Triples of the Sample System

From the collection of the Computer History Museum (www.computerhistory.org)
ENTER THE NUMBER OF THE RATING FUNCTION YOU WISH TO USE: 2
SECTIONAL RATING
ENTER THE PARENT OBJECT NUMBER (0 FOR THE TOP LEVEL IN THE HIERARCHY): 0
SPECIFY MINIMUM FOR HARDWARE: MEDIUM
4 ELEMENT(S) USED
SPECIFY MINIMUM FOR SOFTWARE: HIGH
1 ELEMENT(S) USED
SPECIFY MINIMUM FOR THE COMPUTER CENTER: PRETTY HIGH
4 ELEMENT(S) USED

<table>
<thead>
<tr>
<th>NAME</th>
<th>RATING (USING SELECTED WEAKEST LINK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HARDWARE</td>
<td>PRETTY LOW</td>
</tr>
<tr>
<td>SOFTWARE</td>
<td>PRETTY HIGH</td>
</tr>
<tr>
<td>THE COMPUTER CENTER</td>
<td>PRETTY HIGH</td>
</tr>
</tbody>
</table>

Figure 8—Sectional Ratings for Sample System

SETRATE 3
INDIVIDUAL RATING
ENTER THE NUMBER OF THE OBJECT TO BE RATED: 2

<table>
<thead>
<tr>
<th>NAME</th>
<th>RATING (USING FUZZY MEAN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOFTWARE</td>
<td>SORTOF MEDIUM</td>
</tr>
</tbody>
</table>

Figure 9—Individual Subsection Rating (Software Subsection) of Sample System

ENTER THE NUMBER OF THE RATING FUNCTION YOU WISH TO USE: 1
OVERALL RATING

<table>
<thead>
<tr>
<th>NAME</th>
<th>RATING (USING WEAKEST LINK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>THE INSTALLATION</td>
<td>LOW</td>
</tr>
</tbody>
</table>

Figure 10—Rating Using Weakest Link Outlook and Overall Section of Sample System

SECTIONAL RATING
ENTER THE PARENT OBJECT NUMBER (0 FOR THE TOP LEVEL IN THE HIERARCHY): 0

<table>
<thead>
<tr>
<th>NAME</th>
<th>RATING (USING FUZZY MEAN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HARDWARE</td>
<td>(SLIGHTLY LOWER) THAN FAIRLY HIGH) AND (SLIGHTLY HIGHER) THAN SORTOF HIGH</td>
</tr>
<tr>
<td>SOFTWARE</td>
<td>SORTOF MEDIUM</td>
</tr>
</tbody>
</table>

Figure 11—Partial Rating of Overall Section of Sample System Using Fuzzy Mean Outlook
Figure 12—Sample System Software Sectional Rating Using Fuzzy Mean Weighted by Value Outlook

by Maximum Object Value—for each major subsection of the object specified, this finds the fuzzy mean of the resistances. It then weights these fuzzy means by the maximum object value found in the triples for each major subsection and averages these weighted means. The theory is similar to (d), but with the assumption that the major subsections should be weighted by their relative values, irrespective of the number of triples they each have.

THE USE OF APL

APL is extremely well suited to applications involving linguistic variables and fuzzy set operations. Using appropriately named functions and variables, the linguistic variables can be easily converted into the corresponding base variables using the APL "execute" function. For example, HIGH might be a vector of (0 0 0 0 .0 1 .5 .9 1), representing the linguistic variable high. VERY might be a function which sharpens the curve given to it as its argument, perhaps squaring the argument. Then as shown in Figure 13, if VALUE were a variable containing the character string "VERY HIGH", executing it would return the vector (0 0 0 0 .0 1 .25 .8 1 1), representing the base variable for the linguistic variable very high (Figure 3 gives the curves representing high and very high). The important point here is that APL eliminates the need to do any parsing of the input values; the linguistic variables input are stored in character string format and used as input to the scoring functions at the appropriate time. Additionally, the built-in APL matrix operations are well suited to the fuzzy set operators, which use vectors and matrices extensively. These operators are described in detail in Reference 15.

On the negative side, APL is interpretive; this makes it significantly slower than compiled programs for repeated runs. In addition, it is poorly suited to applications not involving vectors or arrays. The latter point is important for the security evaluation system since most of the code deals with the user interface and the analysis functions. Not only were these awkward to program, but they run rather slowly (these two points not being unrelated). The rating functions, however, which make heavy use of the matrix capabilities while performing fuzzy set operations, are well suited to APL.

USER REACTIONS

Shortly after development started, we arranged to have the system (running on an IBM 370/145 system under the CP/CMS operating system) tested by students who were doing risk analyses of computer installations as term projects. SECURATE was used to analyze seven installations, including one at a large bank and one at a large utility company.

From written evaluations solicited from each of the users, as well as from conversations with them, it seems clear that the system achieved its goal of increasing understanding of installation security. In fact, a couple of users remarked that just filling out the forms made the strengths and weaknesses of an installation’s security a lot clearer. Apparently focusing their thoughts into a logical, well-defined framework enabled them to view the situation more clearly and—even before using the system—to gain some of the insights we had hoped the system would provide.

The most interesting observations were those concerning the use of fuzzy variables. There appears to be definite tradeoff between user acceptance and ease of use. The concept of fuzzy variables was new to all of the users and was greeted with a certain amount of skepticism. While their acceptance of the idea grew as they continued to be exposed to it and had experience in using it, some of them remained skeptical. On the other hand, some of them commented, and we strongly feel this to be true, that the use of these words instead of numbers was a definite help in minimizing the...
tedium involved in collecting the input data. The largest installation turned out to be represented by 136 triples, which came to over 300 different measurements the user had to make. Pinpointing each one on a scale of 1 to 10 appears to us to be a lot more taxing than rating each one as a linguistic variable. Although we didn’t do any comparative studies (which in retrospect would have been a good idea), many users seemed to agree with this in informal discussions.

SUMMARY

We have described an interactive security evaluation system which uses fuzzy metrics. The system models the installation to be analyzed as a set of object-threat-feature triples. The associated measures—object values, threat likelihoods, and feature resistances—are then used as input to security evaluation functions. The user specifies these features in terms of “fuzzy” linguistic variables. The system, implemented in APL, is currently operational on an Amdahl 470 computer.

REFERENCES

5. Goldstein, R. C., The Cost of Privacy, Honeywell Information Systems, 40 Guest Street, Brighton, MA., 02135.

APPENDIX—THE OBJECT HIERARCHY, THREATS, AND FEATURES

The structure of the threats is based on the object hierarchy, which is used as an outline. Threats are listed after the objects they refer to, the objects being specified by name and number from the object hierarchy. A threat listed after a non-terminal node of the object hierarchy refers to all objects descending from the node. Features which counter threats follow the listing of the threats.

Partial object hierarchy

2. Software

2.1 Operating system

2.2 Programs

2.2.1 Applications

2.2.1.1 Source

2.2.1.2 Non-source

2.2.2 Contract programs and packages

2.2.3 System utilities

2.2.4 Test programs

2.3 Data

2.3.1 Personal data

2.3.1.1 Payroll

2.3.1.2 Personnel

2.3.1.3 Other personal data (Privacy Act of 1974, §3(a)(4))

2.3.2 Institution data

2.3.2.1 Marketing

2.3.2.2 Financial

2.3.2.3 Operations

2.3.2.4 Planning

2.3.2.5 Other

Partial Threats Listing

2. Unauthorized access: R/W/E

16 Modification of operating system and system routines

17 Inadequate controls on I/O facilities

18 Password compromise

20 Unsecured storage medium

21 Access outside of allocated memory

22 Modification of stored state vector
23 Unauthorized CE activity
24 Line tapping and spoofing
25 Erroneous or inadequate usage of protection facilities
26 B. Unauthorized access: read
27 Extra copies of output printed
duplicates printed
printing restarted before end
30 Use of erroneous distribution labels
31 Use of erroneous distribution lists
32 Theft of mail
33 Exposed output
34 in user possession
35 within distribution system
36 at operator’s console
37 work in progress
38 Unauthorized reading of terminal buffers
39 Indirect exposure of output
40 C. Unauthorized access: write
41 Modification or spoof of mail transactions
42 Unauthorized modification of data during preparation
43 Data preparation errors
44 Modification of original written data input
45 Defective implementation
46 Inadequate debugging
47 Incomplete operation specifications
48 Inadequate or erroneous error handling
49 Exposure following abnormal end
50 Improper operation
51 Dishonest programs
52 Unexpected alteration of real data

<table>
<thead>
<tr>
<th>Feature No.</th>
<th>Threat Numbers</th>
<th>Feature Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>46</td>
<td>Effective authorization and access control mechanism</td>
</tr>
<tr>
<td>17</td>
<td>47</td>
<td>Minimum authorization policy</td>
</tr>
<tr>
<td>18</td>
<td>48</td>
<td>Effective authorization and access control mechanism</td>
</tr>
<tr>
<td>19</td>
<td>49</td>
<td>Minimum authorization policy</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>Dual authorization required for changes</td>
</tr>
<tr>
<td>21</td>
<td>51</td>
<td>Super user authorization required for changes</td>
</tr>
<tr>
<td>22</td>
<td>52</td>
<td>Log of attempted violations</td>
</tr>
<tr>
<td>23</td>
<td>53</td>
<td>Self-modifying I/O routines not allowed</td>
</tr>
<tr>
<td>24</td>
<td>54</td>
<td>Direction in password choice</td>
</tr>
<tr>
<td>25</td>
<td>55</td>
<td>Store in encrypted form</td>
</tr>
<tr>
<td>26</td>
<td>56</td>
<td>Automatic delay after invalid login attempt</td>
</tr>
<tr>
<td>27</td>
<td>57</td>
<td>Encrypted transmissions to terminals</td>
</tr>
<tr>
<td>28</td>
<td>58</td>
<td>Use of interactive authentication procedure</td>
</tr>
<tr>
<td>29</td>
<td>59</td>
<td>Adequate access controls</td>
</tr>
<tr>
<td>30</td>
<td>60</td>
<td>Adequate and enforced library facility</td>
</tr>
<tr>
<td>31</td>
<td>61</td>
<td>Usage log</td>
</tr>
<tr>
<td>32</td>
<td>62</td>
<td>Proper labelling</td>
</tr>
<tr>
<td>33</td>
<td>63</td>
<td>Proper system design</td>
</tr>
<tr>
<td>34</td>
<td>64</td>
<td>Effective authorization and access control mechanism</td>
</tr>
<tr>
<td>35</td>
<td>65</td>
<td>Adequate I/O controls</td>
</tr>
<tr>
<td>36</td>
<td>66</td>
<td>Protection of state vector</td>
</tr>
<tr>
<td>37</td>
<td>67</td>
<td>Storage in protected storage</td>
</tr>
<tr>
<td>38</td>
<td>68</td>
<td>Administrative controls</td>
</tr>
<tr>
<td>39</td>
<td>69</td>
<td>Human verification</td>
</tr>
<tr>
<td>40</td>
<td>70</td>
<td>Supervision</td>
</tr>
<tr>
<td>41</td>
<td>71</td>
<td>Limited CE access</td>
</tr>
<tr>
<td>42</td>
<td>72</td>
<td>Encryption</td>
</tr>
<tr>
<td>43</td>
<td>73</td>
<td>Effective human engineering</td>
</tr>
<tr>
<td>44</td>
<td>74</td>
<td>Clear, easy to use protection facilities</td>
</tr>
<tr>
<td>45</td>
<td>75</td>
<td>Adequate documentation</td>
</tr>
<tr>
<td>46</td>
<td>76</td>
<td>User education</td>
</tr>
<tr>
<td>47</td>
<td>77</td>
<td>Dishonest programs</td>
</tr>
<tr>
<td>48</td>
<td>78</td>
<td>Unexpected alteration of real data</td>
</tr>
<tr>
<td>49</td>
<td>79</td>
<td>Test programs</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>51</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>52</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>53</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>54</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>55</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>56</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>57</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>58</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>59</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>60</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>61</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>62</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>63</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>65</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>66</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>67</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>68</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>69</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>70</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>71</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>72</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>73</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>74</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>75</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>76</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>77</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>78</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
<tr>
<td>79</td>
<td></td>
<td>Secure but not very effective security features</td>
</tr>
</tbody>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
Print log
Print log
Security conscious I/O routines
Careful administrative procedures
Careful administrative procedures
Important mail sent registered or by courier
Delivery confirmation
Trace log of sensitive output
Library facility for sensitive output
(See also features for threats 34-37)
Clean desk policy
User education
Guarding work in transit
(Refer to features for threats 1-18)
Guarding work in progress
Buffer erase mechanism
Paper shredder
Use of old ribbons for sensitive jobs
Destruction of carbon paper and ribbons
(Refer to features for threats 41-44)
Important mail sent registered or by courier
Delivery confirmation
Second person verification
Checksums
Software checks
Verification checks
Checksums
Software checks
Originator verification
Testing
Audit programs
Testing and verification
Penetration attempts
Program testing and validation
Adequate documentation and design specs
Program testing and validation
Adequate documentation and design specs
Program testing and validation
Adequate documentation and design specs
Program testing and validation
Adequate documentation and design specs
Code inspection, recompilation
Choosing writer who could not benefit
Testing on setup data
Containment of test programs