ESC-TR-99-061

Technical Report
1062

1999 DARPA Intrusion Detection Evaluation: Design and Procedures

J.W. Haines
R.P. Lippmann
D.J. Fried
M.A. Zissman
E. Tran
S.B. Boswell

26 February 2001

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

Prepared for the Defense Advanced Research Projects Agency
under Air Force Contract F19628-00-C-0002.

This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Defense Advanced Research Projects Agency, ISO, under Air Force Contract F19628-00-C-0002.

The ESC Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Gary Tatungian
Administrative Contracting Officer
Contracted Support Management

Non-Lincoln Recipients

PLEASE DO NOT RETURN

Permission is given to destroy this document when it is no longer needed.

Massachusetts Institute of Technology
Lincoln Laboratory

1999 DARPA Intrusion Detection Evaluation: Design and Procedures

J.W. Haines
R.P. Lippmann
D.J. Fried
M.A. Zissman
E. Tran
S.B. Boswell
Group 62

TECHNICAL REPORT 1062

26 February 2001

Lexington Massachusetts

ABSTRACT

Recent DARPA Intrusion Detection (ID) and Strategic Intrusion Assessment (SIA) programs have funded development of new approaches to intrusion detection. The Information Systems Technology Group at MIT Lincoln Laboratory assisted this research with off-line evaluations of these new systems in 1998 and 1999. These evaluations measured detections and false alarm rates of the intrusion detection systems. Eight research sites participated in the second annual evaluation. A network testbed was developed for this evaluation. It included host computers that were attacked and recently-developed traffic generators that produced live traffic modeled after a small Air Force base. This traffic appears as if it were generated by hundreds of users and thousands of hosts. More than 200 instances of 58 attack types were launched against victim UNIX and Windows NT hosts in three weeks of training data and two weeks of test data. Objectives of this effort were to support algorithm development, perform a blind, off-line evaluation of intrusion detection approaches, and help DARPA guide research directions. This technical report describes the testbed design and operation, background traffic modeling and generation, attack modeling and automation, and the scoring procedure. Results of the 1999 evaluation are discussed in a separate technical report entitled "Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation."

ACKNOWLEDGMENTS

The 1998 and 1999 DARPA off-line intrusion detection evaluations were extensive efforts made possible only by the support and contributions of many individuals and organizations. DARPA program managers Theresa Lunt and Sami Saydjari provided guidance and funding. Additional funding was provided by MIT Lincoln Laboratory. Terry Champion and Steve Durst at Air Force Rome Laboratory provided valuable interactions and Linux kernel modifications that made it possible for one host to use many IP addresses. The Sandia National Laboratory Red-Team provided valuable feedback on evaluation design. At MIT Lincoln Laboratory, many full-time staff were involved in both the 1998 and 1999 evaluations over the past three years, including Rob Cunningham, Issac Graf, Dave McClung, Seth Webster, and Dan Wyschogrod. Simson Garfinkle provided attacks as a contractor. Several students dedicated their time to the effort as well, including Kumar Das, Kris Kendall, Jonathan Korba, and Dan Weber. Finally, we would like to thank those organizations and individuals that participated in the 1998 and 1999 evaluations. Many provided valuable feedback and suggestions. In particular, we would like to thank Dick Kemmerer and Giovanni Vigna at UCSB, Mabri Tyson and Phil Porras at SRI, Anup Ghosh from RST, R. C. Sekar from Telcordia, and NingNing Wu at George Mason University.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

1. INTRODUCTION

Lincoln Laboratory's Information and Systems Technology Group, sponsored by the DARPA Intrusion Detection (ID) and Strategic Intrusion Assessment (SIA) programs, conducted annual ID evaluations in 1998 and 1999. Underlying hypotheses, design, and procedures of the 1999 DARPA Intrusion Detection Evaluation are described in this technical report. Evaluation participants, results, analysis, and future research directions are described in a separate report entitled "Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation."

1.1 PURPOSE OF THIS REPORT

The 1998 and 1999 evaluations are the most comprehensive evaluations of intrusion detection systems performed to date and represent the first time DARPA-funded intrusion detection technology has been evaluated. Many choices were made in the design of evaluation procedures, the network testbed, and the metrics in order to satisfy the goals of the evaluation while meeting the constraints of time, funding, and manpower. The purpose of this report is to carefully document the evaluation and describe how and why decisions were made and what alternatives were considered. Where appropriate, discussion of how certain decisions may have affected evaluation results is included, along with suggestions for future evaluations.

The report is arranged as follows. The remainder of this introductory section provides background in intrusion detection and references to further research material. Section 2 discusses the objectives and hypotheses of the 1999 DARPA Off-Line Evaluations, and Section 3 shows how the evaluation design addressed these objectives and hypotheses. Section 4 presents a review of the 1998 evaluation and enhancements made to the evaluation design for 1999. Sections 5 and 6 describe the design of three main components used to generate the evaluation corpus: the network testbed, the attacks, and the background traffic. Section 7 describes procedures used to submit and score results. Section 8 discusses the evaluation timeline, presenting issues that arose at each stage and suggestions for future evaluations. Section 9 presents conclusions and a summary of suggestions for future evaluations.

1.2 BACKGROUND

A computer or network intrusion, sometimes called an attack, is a sequence of related actions by a malicious adversary whose goal is to violate some stated or implied policy regarding appropriate use of a computer or network. Examples include stealing protected data, denying service to a user or group of users, or performing probing actions in an attempt to gain information in preparation for an attack. Growing reliance on the Internet and worldwide connectivity have greatly increased the potential damage that can be inflicted by such attacks.

It is not possible to prevent all attacks with security policies, firewalls, or other mechanisms because system and application software often contains unknown weaknesses or bugs. In addition, complex, unforeseen interactions between software components and/or network protocols are continually exploited by attackers. Successful attacks inevitably occur despite the best security precautions.

Intrusion detection systems have become an essential component of computer security to detect and respond to attacks. They monitor a host or network and issue alerts, sometimes called putative detections, when abnormal behavior is detected or an attack is suspected to have occurred. Some approaches for intrusion detection find attacks in real time and can stop an attack in progress. Others provide after-the-fact information about attacks and can help repair damage, understand the attack mechanism, and reduce the possibility of future attacks of the same type. Advanced intrusion detection systems attempt to detect never-before-seen "new" attacks, as well as previously seen "old" attacks. Several sources provide more detailed information regarding modern techniques for intrusion detection. A review of current approaches is available in [1,2]. Detailed discussions of computer and network vulnerabilities and how they can be detected are provided in [3] and [4].

Evaluations of developing technologies, such as intrusion detection systems, are essential to focus research effort, document existing capabilities, and guide research. For example, annual DARPA sponsored evaluations in the speech recognition area have contributed substantially to rapid technical progress [5]. Periodic speech evaluations have focused research on difficult technical problems, motivated researchers to build advanced systems, facilitated information sharing, provided common corpora, and made it easier for new researchers to enter this field and explore alternate approaches [5].

Despite the importance of intrusion detection systems, we are aware of no evaluations prior to 1998 that accomplished the following: generated an intrusion detection corpus that could be shared by many researchers; evaluated many intrusion detection systems; included a wide variety of attacks; and measured both attack detection rates and false alarm rates. Most prior research in this area evaluated individual systems using a small number of attacks and little background traffic (e.g., [6,7,8]) or evaluated systems using confidential in-house red-teaming experiments where attacks are launched by teams of experts against a test or operational network.

The 1998 and 1999 DARPA evaluations were designed to overcome the limitations of past approaches. This technical report discusses the design and procedures of the 1999 evaluation, and an accompanying technical report discusses evaluation results. In addition, various papers, theses, and reports [9,11–13, 16–19] and many talks produced by MIT Lincoln Laboratory document these evaluations. In particular, the 1999 evaluation is documented in two papers [12,13] and two theses [17,18]. The most recent papers [12,13] provide a historical review of prior evaluations and analyses of results and the theses [17,18] provide attack descriptions. Resources and evaluation summaries can be found on the public domain website, http://www.ll.mit.edu/IST/ideval/index.html. Alternatively, contact Joshua Haines, jhaines@sst.ll.mit.edu, or Richard Lippmann, rpl@sst.ll.mit.edu, for information.

2. OBJECTIVES AND HYPOTHESES

This section discusses the objectives, hypotheses, and features of the 1999 evaluation. Major objectives are discussed in Section 2.1. Hypotheses that address these objectives are given in Section 2.2.

2.1 SUMMARY OF OBJECTIVES

The1999 DARPA Off-Line Intrusion Detection Evaluation had four main objectives, each of which shaped the evaluation and its hypotheses. The objectives were as follows:

- Support developers of intrusion detection systems, by providing rich data sets or corpora for testing and experimentation with various intrusion detection algorithms and technologies.
- Evaluate intrusion detection approaches by analyzing the strengths and weaknesses of each.
- Facilitate analysis of how and why alternate approaches differ. In particular, determine why attacks are not detected and why false alarms occur.
- Report to DARPA information necessary to guide and focus future research directions.

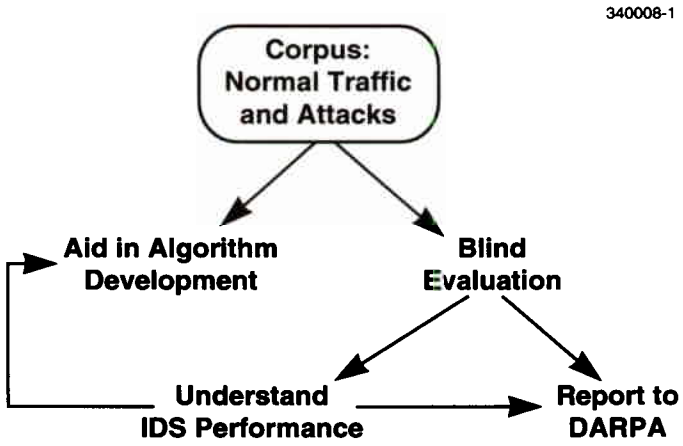


Figure 2-1. The intrusion detection corpus supports all objectives of the off-line intrusion detection evaluation.

Figure 2-1 illustrates evaluation objectives and their interaction. A corpus, or data set, produced as a part of the evaluation helped the evaluation meet its objectives. The corpus included examples of realistic background traffic and attacks. This data helped developers develop new algorithms and also made it possible to perform a statistically blind evaluation of new intrusion detection technology and systems. Results of this evaluation were reported to DARPA directly and were analyzed in great detail by both evaluators and participants to better understand how and why the various approaches performed

differently. This understanding is now being fed back into the development process to create new and better approaches for intrusion detection. DARPA is able to make use of both the raw performance results and the subsequent analysis to guide future research directions. Each component is described in greater detail below.

2.1.1 Corpus

Data collected from the running of attacks and background traffic on the network testbed makes up the intrusion detection corpus. Training data is produced to provide developers extensive examples of background and attack traffic. This data includes labels to indicate where attacks occur and is used to develop algorithms and train systems for the evaluation. Test data is produced in the same way, but is distributed without truth markings, or labels, to those sites participating in the evaluation. This test data is used to evaluate detection and false alarm performance of alternate approaches. Subsequent to the blind evaluation, the test data may then be used as training data for system and algorithm development and in preparation for future evaluations.

2.1.2 Objective 1: Support Developers

The 1999 evaluation corpus provided rich sets of input data for development of intrusion detection technologies. This data greatly expedites algorithm development by providing examples of normal and attack traffic and eliminating the redundant effort this traffic generation would require if it were performed by each system developer. Many researchers had no sources of input data on which to develop and test their systems, except for the Lincoln Laboratory data. This corpus has been downloaded by more than 80 sites and is currently used by many researchers.

2.1.3 Objective 2: Evaluate Technology and Systems

A blind evaluation measured the ability of intrusion detection systems to detect a wide range of attacks to uncover the strengths and weaknesses of different approaches. The testing component of the corpus is distributed without truth markings. Participants are thus "blind" to the truth until their systems are scored. Use of a corpus for off-line evaluations helps ensure that the greatest number of researchers can participate, simplifies entrance into this field, and makes it possible to compare alternate approaches. Efforts were made to keep the evaluation simple and to encourage the widest participation possible by working with developers to create test procedures. The evaluation was scored and results presented according to metrics that had been agreed upon ahead of time. Evaluation results, presented in terms of detection and false-alarm rates, showed how accurately each intrusion detection system performed.

2.1.4 Objective 3: Facilitate Analysis

An objective of the 1999 evaluation was to understand how and why intrusion detection approaches performed the way they did. Participants and evaluators analyzed missed attacks and high-scoring false alarms to understand why each occurred. Results of this process are helping researchers develop new and better approaches to intrusion detection, evaluators improve components of the evaluation, and DARPA guide research.

2.1.5 Objective 4: Report to DARPA

Summarized results of the blind evaluation and conclusions from the detailed analysis of each intrusion detection system in the evaluation were reported to DARPA by both evaluators and participants at a principal investigator's meeting in December, 1999. Viewgraphs from presentations at this meeting are available at a DARPA website [15]. In addition, this technical report, the corresponding report entitled "Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation," and a variety of publications by evaluation participants provide written documentation of the effort and its findings. The objective is to provide DARPA with the information necessary to guide future research efforts and coordinate efforts between the Information Assurance and Survivability (IA&S) programs. For example, evaluation results can help guide program directions by providing information on approaches that are promising, attack classes that are easy or difficult to detect, and situations where composite systems might yield better detection or identification of attacks.

2.2 DRIVING HYPOTHESES

Seven underlying hypotheses shaped the 1999 evaluation design, guided the analysis of results, and supported major evaluation objectives. DARPA, evaluation participants, and evaluators themselves identified these research questions as areas where information would best aid algorithm development, allow understanding of intrusion detection system performance, and help guide research. For each hypothesis, we discuss how it addressed the objectives and how testing was enabled by the evaluation. Results pertaining to each, however, are presented in "Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation."

2.2.1 Hypothesis 1: Attack Space Coverage

"*ID systems can reliably detect Probe, DoS, U2R, R2L, and data attacks against Solaris, SunOS, Linux, and Windows NT operating systems, with low false alarm rates.*"

The Attack Space Coverage hypothesis seeks to aid DARPA and the research community by highlighting areas of the attack space that are either effectively addressed by current approaches or where more research needs to be focused. To guide research directions and quantify performance of each intrusion detection system, results were presented to show how well each system addressed each area of the attack space. An attack classification scheme was designed to allow comparison between alternate approaches and illustrate coverage of each system with respect to the attacks in the evaluation and the attack space. Attacks were classified according to the goal of the attack (*Probe, DoS, U2R, R2L, or Data*) and the operating system against which that instance of the attack was run (*Solaris, SunOS, Linux, or Windows NT*). This classification scheme is discussed further in section 6.3.

To allow for comparison of system performance within each subset of the attack space, evaluators attempted to ensure that a sufficient number of attack instances within each category were run. Evaluation results were analyzed within each subset. The attack classification scheme allowed intrusion detection system designers to choose, prior to the blind evaluation, which types of attacks they felt that their system

should detect. As agreed upon by DARPA and evaluation participants, systems were scored against only those attacks that they claimed to be able to detect. Analysis of evaluation results shows that in some areas this classification was too coarse to meet the needs of every system. A more effective classification for the purpose of intrusion detection might incorporate the evidence each attack leaves on a host or network and how each attack can be detected.

2.2.2 Hypothesis 2: New and Stealthy Attacks

"*Systems can detect new and stealthy attacks as well as they detect old attacks used in prior evaluation data.*"

The 1998 evaluation demonstrated that current research intrusion detection systems were not able to detect new attacks as well as they could detect old attacks available in training data. To assist algorithm developers and further understand current approaches, the 1999 evaluation attempted to determine the following: whether systems had improved in their ability to detect new and stealthy attacks; why new attacks were not detected; and what could be done to improve the systems. Old attacks are those for which examples exist in training data or previous evaluation data, while new attacks are those for which examples had not been distributed to participants prior to the blind evaluation. "Clear" attacks are those that are run in the same form in which they were found on the Internet, while "stealthy" attacks are those that have been modified in some way to be harder to detect.

All attack instances in the evaluation were designed and classified as being either new or old and clear or stealthy, and evaluation designers ensured that there were sufficient numbers of each. Results were analyzed and presented within these categories to contrast performance of each intrusion detection approach in each category.

2.2.3 Hypothesis 3: Input Data

"*Different streams of input data are best for detecting different types of attacks.*"

The intrusion detection systems evaluated were either network-based systems using the network sniffing data as input, or host-based systems using either host auditing or other host data as input. Determining which systems best addressed each subset of the attack space gives DARPA a coarse idea of the input data and instrumentation required to detect attacks in the various subsets of the attack space. Future work along the same lines might entail a finer-grained analysis of which features of which input data stream are most useful for addressing each area of the attack space.

The evaluation was designed to address the requirements and capabilities of both network- and host-based systems. Evaluation results for each type of system are contrasted in the "Results" technical report.

2.2.4 Hypothesis 4: Anomaly Detection

"*Anomaly detection can detect new attacks with low false alarm rates.*"

Anomaly detection is a technique for detecting computer attacks, where an intrusion detection system "learns" the normal behavior of the computer or network as it's reflected in a particular data stream. Attacks are detected without prior knowledge of exactly what the attack looks like because they stand out sufficiently from the "normal" traffic. Signature-based detection, on the other hand, uses prior knowledge of what each attack looks like in order to detect it. Since systems of each type were run in the 1999 evaluation it was possible and useful to compare the performance of each and determine their strengths and weaknesses. In addition, the evaluation attempted to help anomaly system developers understand how performance of these systems can be improved.

The evaluation supported anomaly detection systems in a several ways. Two weeks of realistic background traffic, without attacks, were provided for anomaly system training. Detection false alarm plots were used for results analysis and provided detection rates at various false alarm thresholds. This made it possible to compare anomaly detection systems with signature-based detection systems at different false alarm rates. Subsequent to the evaluation, the analysis of false alarms helps anomaly system developers to improve those systems by minimizing false alarm rates. Results contrasting signature and anomaly detection were presented to DARPA and the research community and are summarized in the "Results" technical report.

2.2.5 Hypothesis 5: Improvement from 1998 to 1999

"*Systems used in the 1999 evaluation have improved since the 1998 evaluation.*"

DARPA, technology developers, and evaluators all need to know whether intrusion detection systems and technology have improved from the previous year and in what ways they have improved. This allows the research community to understand how well the research environment is fostering development of new and better approaches to intrusion detection. The overall goal of DARPA, and these evaluations, is to help intrusion detection technology improve, and therefore it is critical to be able to quantify how much or how little improvement has occurred.

The 1999 evaluation enabled comparison between performance in 1998 and 1999 in two ways. First, old and clear attack instances in the 1999 evaluation are roughly equivalent to the set of all attacks seen in the 1998 evaluation. Thus a comparison can be made that illustrates the raw improvement of intrusion detection systems between the two years. Second, new (unseen) attacks were included in both annual evaluations and performance can be compared for these attacks. Despite the fact that the two sets of new attacks will be distinct, they contain attacks not before seen by the systems being evaluated.

2.2.6 Hypothesis 6: Attack Identification

"*Systems accurately identify attack: name, category, start time, and components.*"

Participating systems could optionally provide more detailed information along with putative attack detections. This included the name, category, start time, and network components of each attack. Attack identification information can simplify analysis of alerts and guide responses to attacks. This type of forensic information is particularly important to understand and respond to new attacks.

Extensive analysis of traffic and audit records was performed to create "truth" markings for scoring system performance. Identification accuracy was scored and presented for each type of information provided. In the future, value functions for each piece of information might be integrated to be able to assign a "quality of detection" score. Results of the identification phase of the scoring procedure are provided in the "Results" technical report.

2.2.7 Hypothesis 7: Error Analysis

"*A detailed analysis of misses and false alarms can provide insight into system weaknesses and guide system development.*"

Determining how and why each intrusion detection system performed the way it did in the evaluation was an objective of the evaluation. This hypothesis proposes that an analysis of attacks missed and false alarms incurred is a good way to achieve this understanding.

To address this hypothesis, a procedure was developed whereby both participants and evaluators performed this analysis and attempted to quantify the results. This analysis and conclusions of this hypothesis are provided in the "Results" technical report.

3. DESIGN PRINCIPLES

This section introduces three major design principals of the evaluation. First, a stand-alone network testbed was used to generate an evaluation corpus. Second, intrusion detection system performance was measured using both attack detection rate and false alarm rate. Third, an off-line evaluation format allowed many systems to be evaluated and supported intrusion detection researchers with examples of realistic attacks and background traffic. Discussion of the evaluation testbed approach, the metrics, and the off-line format and how they support evaluation objectives and hypotheses is included in sections 3.1, 3.2, and 3.3.

3.1 THE TESTBED APPROACH

The off-line approach to evaluation requires creation of a corpus for measuring both detection and false alarm rates. Three approaches were initially explored to generate a corpus that could be widely distributed and included both background traffic and attacks. The first proposal was to capture operational traffic during normal operations and with controlled live attacks against selected components of the operational network. This was not possible because it would release private information, compromise security, and possibly damage systems or interrupt important network services. A second proposal was to sanitize sampled operational data and insert attacks into the sanitized data. Analysis showed that this again was not possible due to the difficulty of removing all security-related and private information from network traffic and the complexity of inserting attacks without introducing artifacts. Sanitization alone would require examining every email message, file transfer, and browser interaction to eliminate private or confidential information. It would also require changing all user names, IP addresses, file names, system names, and any other site-specific information. The final proposal, which was adopted, was to recreate normal and attack traffic on a private network using real hosts, live attacks, and live background traffic.

The three main components of this testbed approach are the network testbed, the background (or normal) traffic, and the attacks. A broad overview is given in Figure 3-1. The three major inputs in generating the 1999 corpora are synthetically modeled background traffic, attacks, and the testbed network on which the simulation is run and data collected.

Synthetic background traffic is generated by modeling both traffic collected at Hanscom Air Force Base and traffic statistics collected at many other bases. Attacks and scripts to carry out the attacks are collected from the Internet or developed independently and run on the testbed network in conjunction with background traffic. Data collected from this network form the intrusion detection corpora—the main components of these evaluations.

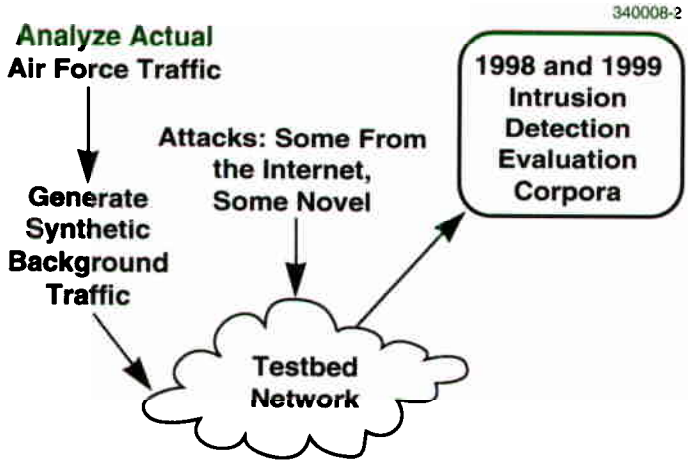


Figure 3-1. The three major components required to generate the 1999 corpora are synthetically modeled background traffic, attacks, and the testbed network on which the simulation is run and the data is collected.

These components are discussed in greater detail in later sections of this report. The testbed network, required to allow the simulation of an Air Force base and the subsequent data collection, is the subject of section 5. Attacks, used to measure detection rates, are discussed in section 6. Background or normal traffic, used to measure false alarm rates, is discussed in section 7.

3.2 ANALYSIS USING DETECTION AND FALSE ALARM RATES

The inclusion of realistic background traffic to measure false alarm rates is a novel feature of the 1998 and 1999 evaluations. Most prior evaluations have only measured detection rate with a small subset of attacks. Attack detection rate and false alarm rate, together, provide the best quantification of intrusion detection system performance. Measuring the detection rate alone only indicates the types of attacks that an intrusion detection system may detect. Such measurements do not indicate the human workload required to analyze false alarms generated by normal background traffic. False alarm rates above hundreds per day make a system almost unusable, even with high detection accuracy, because putative detections or alerts cannot be believed and thus security analysts must spend many hours each day dismissing false alarms. Low false alarm rates combined with high detection rates mean that the putative detection outputs can be trusted and that the human labor required to confirm detections is minimized. Evaluation results are always presented in terms of detection rate at a specified rate of false alarms.

3.3 OFF-LINE VS REAL-TIME EVALUATIONS

There were two components to the 1999 DARPA Intrusion Detection Evaluation: an off-line evaluation and a real-time evaluation. The real-time evaluation, performed at Air Force Research Laboratory

(AFRL), made use of background traffic, and included a smaller number of complete packaged systems run on a network test bed. The 1998 real-time evaluation is summarized in [10], and the 1999 real-time evaluation is summarized in a presentation found on a DARPA website [15]. Many of the traffic generation tools and attacks developed for the off-line evaluation were also used in the real-time evaluation. The off-line evaluation was carried out at MIT Lincoln Laboratory, applied to many intrusion detection systems, and resulted in an archived corpus that continues to be used for algorithm development and as a baseline for future evaluations. This report covers only the off-line evaluation, however, here we contrast some of the advantages of these complementary approaches.

For the real-time evaluation, systems are delivered to Air Force Research Laboratory and installed in the AFRL network testbed. While under evaluation, systems attempt to identify attack sessions in the midst of normal activities, in real time. Two main advantages of a real-time, on-line evaluation are that intrusion detection systems are able to perform active rather than passive monitoring during the evaluation and they can take action in response to a particular attack or a possible detection. Theoretically, these systems could query hosts to determine status and use data sources not provided in corpora used for the off-line evaluations. However, none of the DARPA research systems performed this sort of dynamic monitoring in 1999. Another advantage is that a real-time evaluation can measure practical characteristics of an intrusion detection system such as CPU and memory usage, and ease of installation and configuration. The main disadvantage of the real-time evaluation is that in many cases only one intrusion detection system can be evaluated at a time and attacks and background traffic must be regenerated for each system evaluated. Therefore it is a very time-consuming process and is not well suited to training (as opposed to testing) intrusion detection systems. In contrast, corpora for the off-line evaluation are produced once and can be used by any number of systems at any time for evaluation or training.

In the off-line evaluation, systems are tested using network traffic, audit logs, system logs, file system information, and other host information collected on a testbed network and distributed to evaluation participants. Systems process this data in batch mode and attempt to identify attack actions in the midst of normal activities. This off-line approach has several advantages. The evaluation allows more developers to participant since tuning and testing of intrusion detection systems is done in parallel at the developer site. This is unlike the real-time evaluation, which often must be performed individually for each system. The off-line technique focuses on technology development rather than complete system development by removing the need to deliver a mature system for installation in the testbed network, making it possible for a developer to evaluate algorithms and techniques in the absence of the complete system. Additionally, this approach leaves behind a corpus that can be downloaded to the research site and can be used any time to assist in testing and developing algorithms and techniques.

The off-line approach was well suited for those research systems that participated in the 1999 evaluation, as support of active monitoring was not required for those systems. Eventually it will be necessary to support future research systems and existing commercial systems that perform dynamic querying of the network or host. Although parts of these systems might be evaluated in the existing off-line style, real-time evaluation components will be required.

4. SUMMARY OF THE 1998 DARPA OFF-LINE EVALUATION

This section summarizes the first DARPA off-line evaluation, which took place in1998 and is discussed in more detail in [9,16]. Figure 4-1 shows the network testbed developed for the 1998 off-line evaluation. This testbed created live traffic similar to traffic that flows between a small Air Force base and the Internet. A rich variety of background traffic was generated that appeared to be initiated by hundreds of users on thousands of hosts. The left side of Figure 4-1 represents the "inside" network, the fictional Eyrie Air Force Base, and the right side represents the "outside" network, the Internet. The 1998 evaluation focused on UNIX attacks launched from remote machines. Automated attacks were launched against three inside UNIX victim hosts running SunOS, Solaris, and Linux, respectively. More than 300 instances of 38 different attacks were embedded in seven weeks of training data and two weeks of test data.

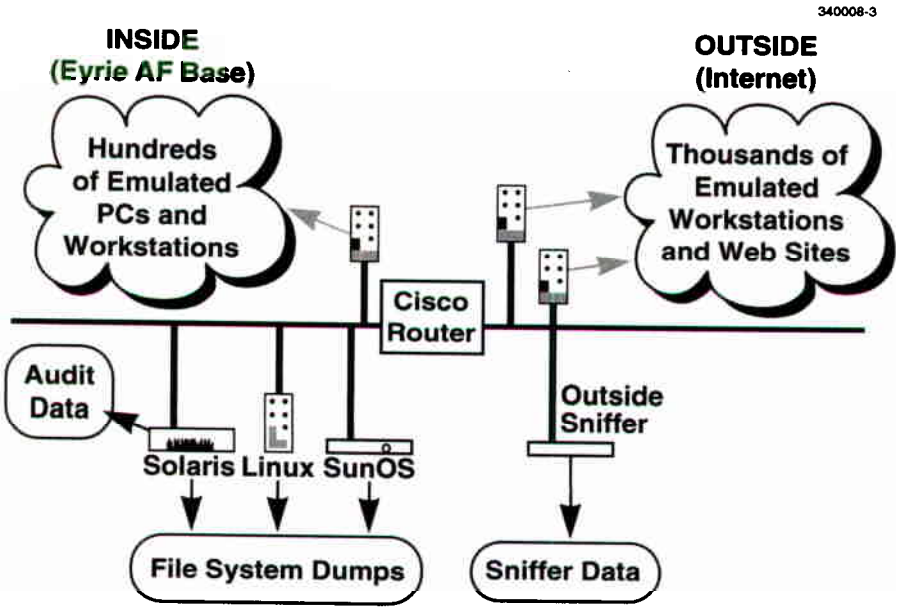


Figure 4-1. Testbed network used in the 1998 off-line evaluation.

Six research sites participated in the blind evaluation and results were analyzed to determine the attack detection rate as a function of the false alarm rate. Performance was evaluated for old attacks included in the training data and new attacks, which only occurred in the test data. The best of the systems evaluated detected approximately 60% of old and new user-to-root (U2R) attacks at a false alarm rate of 10 false alarms per day. Detection rates were lower for new denial-of-service (DoS) attacks and remote-to-local (R2L) attacks where a remote user illegally accesses a local host. Although detection accuracy for old attacks in these two categories was approximately 80%, detection accuracy for new and novel attacks was less than 25% even at high false alarm rates.

The 1998 evaluation demonstrated that it's possible to evaluate a diverse set of intrusion detection systems. In addition, this initial work demonstrated that there is a widespread interest in obtaining training and test corpora containing normal traffic and attacks to develop and evaluate intrusion detection systems. To date, more than 80 sites have downloaded all or part of the 1998 off-line intrusion detection archival corpus from a Lincoln Laboratory web site [5]. In addition, the learning experience provided by the first evaluation led to numerous suggestions for improvement. These suggestions include providing training data containing no attacks to train anomaly detectors, simplifying scoring procedures, more fully automating scoring and attack verification, modifying detection/false-alarm analyses to allow more flexible definition of analysis intervals, extending the attack taxonomy used to group results for analysis and specify system capabilities, exploring false alarm rates with a richer range of background traffic, providing a written security policy, and performing more detailed analyses of misses and false alarms. Almost all of these suggestions were incorporated in the 1999 evaluation and are discussed in the next section.

4.1 CHANGES FROM 1998 TO 1999

The many changes integrated into the simulation testbed and the design of the evaluation between 1998 and 1999 are summarized in Figure 4-2. For each component of the evaluation, the major features supported in the 1998 evaluation and the enhancements added for the 1999 evaluation are given. This section discusses each of these enhancements.

The testbed network in 1998 primarily emphasized security of UNIX hosts and UNIX-based intrusion detection systems. For 1999, the evaluation reacted to the deployment of Windows NT in many sectors of the government and included attacks against Windows NT and attacks launched from NT hosts. A Windows NT Microsoft Internet Information Server (IIS) and mail server, hume.eyrie.af.mil, was added to the simulation network as a victim host, as were two general-purpose Windows NT workstations for launching attacks that required Windows NT.

Many background traffic services were updated in 1999 to provide more realism and a wider distribution of background traffic. The telnet generation procedure was enhanced with user profiles to simulate different types of users with different work habits. Users were simulated at the console of the Windows NT machine in the form of an "auto-browser" that surfed the web using Netscape, visiting a wide range of web sites.

In response to developer requests, two weeks of the training data containing only background traffic, and no attacks, were distributed so anomaly detection systems could be trained on large samples of normal traffic.

No matter how diverse the simulated background traffic is, it can never match real-world traffic. Real-world traffic varies substantially both over time and across sites and it would be too costly to model this temporal and site-dependent variability in the evaluation testbed. Instead, we decided to measure real-world false alarm rates by running top performing intrusion detection systems from the 1999 evaluation on data collected at one site (Hanscom Air Force Base). Although results from multiple sites would be desirable, funding limits made this unfeasible. Since it's impossible to label this real data with what is or is not part of an attack, only the false alarm rate will be analyzed. Analysis will include a comparison of the alerts incurred by intrusion detection systems on real data to those false alarms incurred in the

340008-4

	1998 Baseline	Added in 1999
Network	UNIX	Windows/NT
Background Traffic	UNIX Background Traffic	NT Background Traffic Two Weeks No-Attack Train False Alarm Analysis for Top Systems on Actual AF Traffic
Attacks	Outside Attacks 38 Attack Types Stealthy Against Simple Keyword Systems Novel UNIX Attacks	Inside Attacks Data Attacks >50 Attacks Types Novel NT, UNIX Attacks Stealthy Against 1998 ID Systems
Metrics	Detection + False Alarm ROCs	Identification Error Analysis

Figure 4-2. Summary of changes incorporated into the 1999 evaluation.

in the simulated data and to statistically characterize the real-world data for comparison to that synthesized on the testbed network. The purpose is to quantify how realistic the simulation network is and determine in what ways the simulation network could be improved to yield more accurate results in future evaluations. This component of the 1999 evaluation is still in progress.

A much wider assortment of attacks was run in 1999, and many more were automated as opposed to being run by hand. One objective of the evaluation was to test detection of new and stealthy attacks. Thus many attacks not-before-seen in the evaluation were implemented and many old attacks modified to be stealthier. See sections 6.4 and 6.5 for discussion of new, old, clear, and stealthy attacks. See [17–18] for further discussion of attacks and techniques developed for the 1999 evaluation. "Insider" attacks, originating from within the simulated Air Force base, were also added. Attack goals, "flags" to be captured, in the form of secret files and directories, were also added. Attack instances that led to the illegal access to this information were classified as "data" attacks. Attack scripting and automation

technology was improved in 1999 and allowed for greater automation in documenting each attack instance. Hardware and software was configured to allow attacks to be launched from any of the virtual machines that are simulated on the testbed. This new configuration also allowed a local network sniffer to capture all attack actions without the interference of normal traffic or simulation artifacts.

Metrics used in the 1998 off-line intrusion detection evaluation were extended in the 1999 evaluation to include analysis of attack identification accuracy, i.e., how much information the intrusion detection system was able to report about each attack instance. Additional analysis was performed to understand why detection systems and algorithms missed certain attacks or incurred certain high-scoring false alarms.

5. NETWORK TESTBED

5.1 DESIGN OVERVIEW

Figure 5-1 shows conceptual view of the evaluation testbed used in the 1999 evaluation. The testbed generates live network and host traffic similar to that seen on one small Air Force base and between that base and the Internet. Attacks can be automated and run on the testbed against the Air Force Base servers. This simulated Air Force base is often referred to as Eyrie Air Force base, a fictional name created for the evaluation. In keeping with military standards, nodes on the Eyrie AFB network have fully qualified domain names, e.g., "node1.eyrie.af.mil."

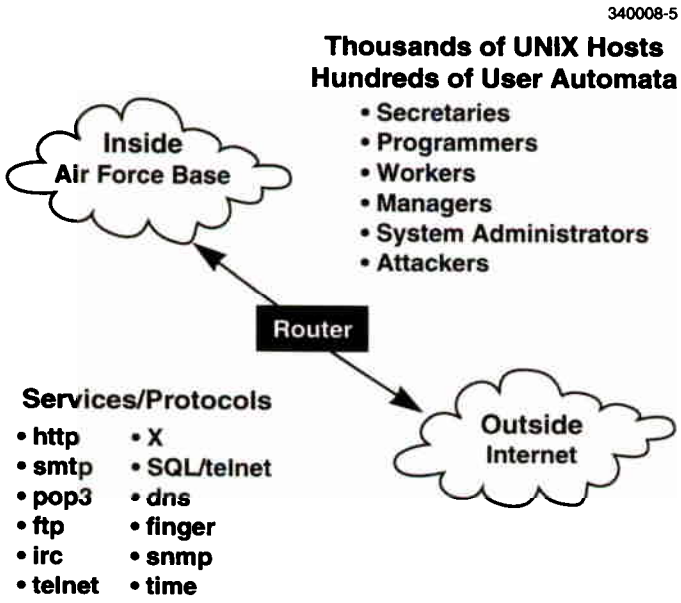


Figure 5-1. The evaluation testbed creates many types of live traffic using thousands of virtual hosts and hundreds of user automata to simulate a small Air Force base separated by a router from the Internet.

Software automata simulate hundreds of programmers, secretaries, managers, and other types of users running common UNIX application programs and some Windows NT programs. A custom Linux kernel modification developed at the Air Force Research Laboratory allows a small number of actual hosts to appear as if they were thousands of hosts with different IP addresses. An Expect-based scripting environment is used to automate background traffic sessions and many of the attacks run on the testbed. Protective devices such as firewalls were omitted to focus on attack detection not prevention.

Many types of background, or normal, traffic are generated using a variety of network based application programs. User automata send and receive mail, browse web sites, send and receive files using FTP, use telnet to log into remote computers and perform work, send and receive IRC messages, monitor the router remotely using SNMP, and perform other tasks. The proportion of traffic from different services and the variability of traffic with time of day are similar to that observed on one Air Force base. Background traffic statistics and the software used to generate the sessions are discussed in section 7.2.

This section discusses the testbed, testbed operation, and data collection. Sections 5.2 and 5.3 discuss the hardware and software used in the network testbed. Section 5.4 presents administration of the testbed, including simulation startup and data collection.

5.2 HARDWARE

Figure 5-2 shows a more detailed block diagram of the evaluation testbed. Machines on the left simulate Eyrie Air Force Base, or the "inside" network, while machines on the right simulate the Internet, or the "outside" network. A Cisco router connects the inside and outside networks.

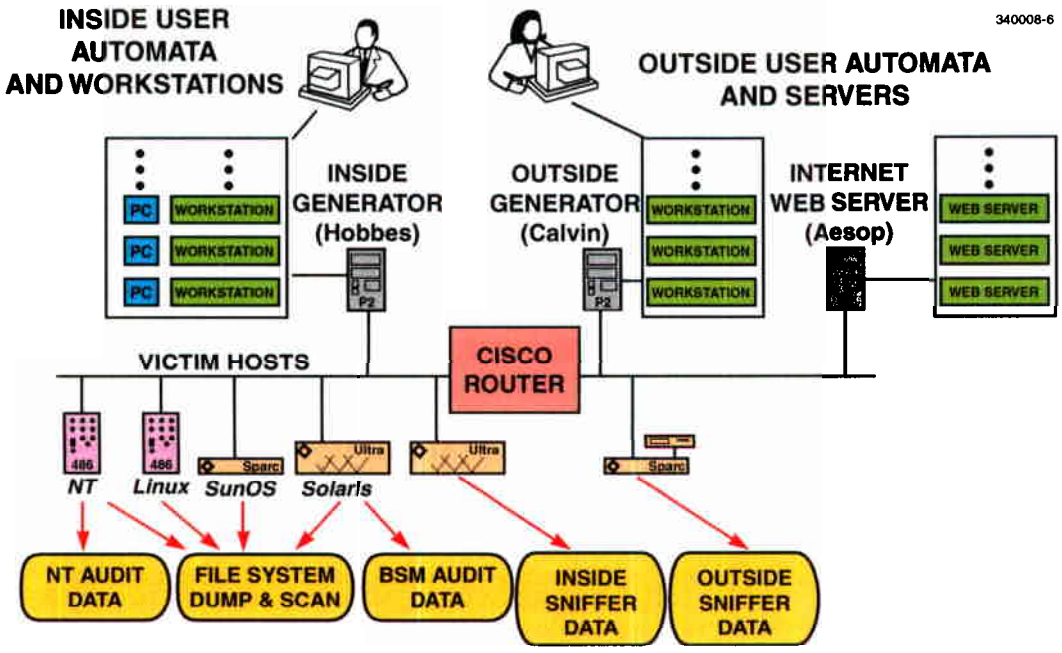


Figure 5-2. Diagram of the main operational features of the 1999 testbed network.

The following two lists summarize the hardware and operating system software of each host in the testbed network and the role that each plays. Victim hosts emulate servers on the Air Force base and are the primary targets for attacks. The inside network, which simulates Eyrie Air Force Base, contains the following:

- **Linux victim**: Marx.eyrie.af.mil, running RedHat 5.0 with Linux kernel version 2.0.30, on an Intel Pentium II, 266 Mhz, platform is the Eyrie AFB web server. It serves the Base's homepage for both the Internet and internal use. An Apache web server, version 1.1.3, was used along with sshd, version 1.2.25. Snmpd, from RedHat 5.0, was run to allow periodic monitoring of server statistics from a remote Air Force base. In addition, telnet, FTP, and finger, from Red Hat 5.0, were supported by Marx.
- **SunOS victim**: Zeno.eyrie.af.mil, running SunOS 4.1.4 on a Sparc 2, is an Eyrie Air Force Base file server. Zeno ran sendmail and allowed file sharing between users via an FTP server. Telnet and finger were also supported by Zeno. Versions of each were those that came with the operating system.
- **Solaris victim**: Pascal.eyrie.af.mil, running Solaris 2.5 on an UltraOne, is a shell or login server for Eyrie Air Force Base. Users often logged into Pascal to read mail or surf the web via the Lynx text-based web browser. Except for sshd, version 1.2.22, all other software came with Solaris 2.5. Commonly used services offered by Pascal were telnet, smtp, ssh, FTP, and finger.
- **Windows NT victim**: Hume.eyrie.af.mil is Windows NT Server, 4.0, build 1381, Service Pack 1 running on a Dell Poweredge 2300. Several services and third party applications were installed. Hume was equipped with IIS (Internet Information Server) version 2.0, which provides FTP, gopher, and web servers. The Resource Kit for Windows NT 4.0 was installed which includes a mail server, called MailSrv, and several other utilities. The telnet server provided by the Windows NT Resource Kit proved unreliable, and therefore a registered version of Ataman TCP Remote Logon Service (Version 2.4) was used to provide telnet login capabilities [www.ataman.com]. Netscape 4.0.8 and Microsoft Office 97 were installed. A compression utility, WinZip 7.0 [www.winzip.com], and a utility to gather web server statistics, Wusage 6.0 [www.boutell.com/wusage] were also installed. The most commonly used services offered by Hume were HTTP, smtp, FTP, and telnet. Reference [17] contains much more detailed descriptions of Windows NT in the network testbed and of Hume's configuration.
- **Inside traffic generator**: Hobbes.eyrie.af.mil, running RedHat 5.0 with kernel version 2.0.32 and the AFRL Linux kernel modification, was the inside traffic generator. Hobbes simulated 20 Eyrie Air Force Base PCs and workstations with the virtual host capability allowed by the kernel modification. Modifed versions of sendmail, telnet, and login, and the Expect-based traffic generation tools are used to automate services and simulate real users. These are discussed later in this section. Hobbes runs Bind version 4.9.6 (unpatched), serves as the domain name service (DNS) server for the inside network and is the primary DNS server for the eyrie.af.mil domain.

- **Inside sniffer**: Locke.eyrie.af.mil, running Solaris 2.6 on an Ultra One, was used to capture network traffic on the inside network. Tcpdump version 3.3 was used to collect traffic, and over 16 gigabytes of SCSI disk storage were allotted for storing large tcpdump files.
- **Inside attackers**: Two real workstations, one Linux and one Windows NT, could periodically act as attack victims and also serve as launch points for attacks from within the AFB. The Linux machine was a RedHat 5.0 machine outfitted with the Expect-based traffic generation tools for attack automation. The other host was Windows NT Workstation, 4.0, Service Pack 1, with PERL installed for attack automation.
- **Cisco router**: Loud.eyrie.af.mil, a Cisco 2500 router, running IOS v. 11.3 rev4, was the gateway router that connected the inside and outside network. No filtering was enabled there. Loud allowed itself to be monitored via SNMP by those who knew (or guessed!) the community password.

The "outside" of the testbed simulates the Internet and contains the following:

- **Internet web server**: Aesop.eyrie.af.mil, running RedHat 5.0 on an Intel platform, is the Internet web server and appears to be thousands of individual Internet web servers. Aesop uses the AFRL Linux Kernel modification and custom webserver software developed for the evaluation that responds to queries to each of the 2,500 virtual hosts and appears to be many different varieties of web servers.
- **Outside traffic generator**: Calvin.eyrie.af.mil, running RedHat 5.0 and kernel version 2.0.32 on an Intel platform, made use of the AFRL Linux kernel modification and the Expect-based traffic automation tools. Calvin simulated 65 virtual Internet hosts. Calvin appeared to be a gateway to 65 workstations around the Internet to and from which normal and attack traffic was generated. Custom tools similar to those on hobbes, the inside traffic generator, are employeed on calvin and are discussed throughout this section.
- **Outside attackers**: Three real workstations, two Linux and one Windows NT, act as launch points for attacks on the inside network. The Linux workstations are RedHat 5.2 machines equipped with Expect-based automation software; the Windows machine is NT, 4.0 Worktation, equipped with PERL for attack automation.
- **SNMP monitor**: Glassmac, running Mac O/S 8.5 on a Macintosh 6100/60, emulates an SNMP monitor at a remote military site monitoring various network components at Eyrie AFB. It runs the AG group's Netmeter 1.0 software for SNMP monitoring.
- **Outside sniffer**: Solomon.world.net, running Solaris 2.6 on a Sparc 2, was used to capture network traffic to and from the Air Force base. Tcpdump version 3.3 was used to collect the traffic.

5.3 SOFTWARE

Several major, custom software packages enable the testbed network and are presented in the following subsections. Sections 5.3.1 and 5.3.2 discuss the virtual host simulation required for the evaluation testbed. Section 5.3.3 discusses the AFRL Linux kernel module that allows one physical host to appear as many. Section 5.3.5 describes how virtual hosts were shared between background traffic

generators and attack generators. Section 5.3.6 describes how "Internet-like" web services were provided for this stand-alone network, and finally, section 5.3.7 describes the Expect-based software used to automate background traffic sessions as well as attacks.

5.3.1 Virtual Hosts

A challenge in creating the testbed environment was to create network and host traffic that appeared to be a large network without actually having to purchase, configure, and maintain hundreds or thousands of physical machines. To achieve this, some physical machines needed to support multiple virtual hosts. Each virtual host is simulated, in lieu of having actual hardware for each host. The barest level of simulation is that of providing an IP address for the virtual host that merely aliases the real host. Thus virtual hosts appears as replications of the original host but with different IP addresses and hostnames. Further levels of simulation involve having the operating system and applications of the real host respond in unique ways for each virtual host or having separate copies of applications for each host. Simulation of unique hardware and device configurations are another possibility. There are many levels at which this host simulation can be achieved. Several options for this virtual host simulation are discussed in the next section.

5.3.2 Options for Virtual Hosts

Several options were explored for emulating multiple hosts. The first option was to actually purchase, install, and maintain physical hosts for each desired host. Although this would provide the greatest level of simulation, the cost of purchasing the hardware and of administering such a large network was prohibitive.

Another option was to make use of commericial traffic generation software to generate network sessions. These tools allow the user to simulate network connections from any number of hosts to the real servers in the testbed network, thus providing simulation of the clients for these servers. However, these tools would not simulate a virtual host for incoming network connections. Also, these tools generally do not allow the detailed control necessary to create interactive sessions that look like real human users, emulating human-like typing time, misspelling and retyping commands, or automatic generation of whole sessions based on statistics. Nor do they use real implementations of the protocols and services required but simulate the protocols. Since many of the researchers supported by this testbed were analyzing the network and host traffic in considerable detail, the most realistic network protocol implementations possible were desired. These commerical tools did not seem to be appropriate for the level of simulation required.

A middle-ground was to assemble a collection of custom software tools to perform the simulation desired. To provide the IP address level simulation for both network clients and network servers, a Linux Kernel modification developed at AFRL was used. This software allowed for network traffic to and from virtual hosts by mapping process IDs of active network processes to the virtual IP address to or from which the network session was coming. This mapping is provided for all IP-based services including TCP, UDP, and ICMP. This kernel modification achieves the desired network simulation for all virtual hosts configured on a given host but does not provide the application-level simulation required for the content of virtual host traffic to appear different from one virtual host to the next.

The next step was to decide which software packages should be customized for each virtual host and also how they should be customized. One option was to use the chroot system-call for isolating incoming and outgoing network processes to a separate filesystem depending on the virtual host to or from which the session was desired. This would allow a completely separate file system and distinct set of applications for each virtual host. However, using chroot is not feasible for hundreds of virtual hosts. Chroot is designed for use with one or a few limited virtual file systems on a given physical host. Two sources of overhead stemming from the Linux operating system also posed problems for the chroot solution. First, file system management routines in the Linux kernel were not able to support hundreds of file systems and allow them to appear as if they were separate physical machines, load imposed on one virtual file system would greatly affect performance of another. Second, the chroot system call itself needs to be made for each and every network process and incurs overhead while isolating the calling process to part of the real file system.

Instead of using chroot, many often-used protocols and services were adapted for use on these traffic generation machines (where the virtual hosts would reside) so they would respond in a virtual-host specific fashion. Sendmail server daemons, webserver daemons, telnet server daemons, and the /bin/login program were all modified to provide emulation specific to each virtual host. Server programs were modified to check the destination virtual IP address of the network session and used that information to dictate the appropriate behavior.

The virtual host solution used in the 1999 evaluation provides complete simulation of virtual machines at the network layer, partial simulation of the file system and application layer, and allowed use of real protocol implementations. However, there are some limitations to note. The file system of each virtual host to a given physical host will be exactly the same as others. Users on the physical machine will have the same password and home directory on each virtual machine. Processing or other load, such as a denial-of-service attack, imposed on one virtual machine will affect the perceived performance of other virtual machines on the same physical host. The following subsection discusses the AFRL Linux kernel module that enables the IP level simulation.

5.3.3 Kernel Modifications

A modification to the Linux kernel allows a single physical Linux machine to look like many individual hosts and enables the emulation of hundreds of hosts with one physical machine. A custom user-side executable, along with a modification to the Linux kernel, allows real and automated users to generate network sessions from virtual IP addresses assigned to the machine. This custom software was developed at Air Force Research Laboratory by Steve Durst and Terry Champion and is now supported by their private company, Skaion. This Linux kernel natively allows the host to respond to network connections to these virtual hosts.

A special character device, /dev/ipst, allows communication with the Linux kernel which maintains a table of UNIX process ID numbers and corresponding IP addresses. TCP, UDP, and ICMP processing routines in the kernel are modified to consult this table and substitute the appropriate IP address when forming network packets and connections for these processes. To start an outgoing network session, a special executable "Ipsession" is passed the command to run and the IP from which to source the

connection. IPsession then executes the command and requests that the kernel map the processes PID to the desired virtual IP. Thus, all network traffic from that process is mapped to the virtual IP and source IP addresses are written appropriately. For example, the command

```
IPsession -i 172.16.118.59 -c telnet 172.16.112.50
```

would begin a telnet session to 172.16.112.50 from the virtual IP 172.16.118.59.

Incoming network connection requests to virtual IP addresses are handled by dummy virtual network interfaces, one for each IP. Network connections that result appear as though they are carried out between the source and the virtual host and not the underlying physical host. All outgoing packets are written using the correct IP address. Incoming network sessions are possible to any of the virtual hosts and outgoing sessions are allowed to emanate from any of the virtual hosts. All traffic to and from these virtual hosts looks as if it is going to or coming from a real host at the TCP, UDP, or ICMP level.

5.3.4 Partial Application-Layer Simulation

Several commonly used network services were modifed to support the virtual host simulation at the application layer. Web servers, sendmail daemons, telnet daemons, and login-specific programs have been modified to respond differently depending on the virtual host to or from which each network session is conducted. The modified /bin/login program checked the destination virtual IP address of each incoming login and used that address to find and display a unique message-of-the-day (motd) for the virtual host that the user was accessing. The sendmail program that sends mail for users did not work well with the virtual host configuration and was actually rewritten in PERL to ensure that mail messages sent by users on a virtual host appeared to be coming from the virtual host and not the physical host. The web server program was written in PERL to ensure that responses to queries appeared to come from the virtual host and to allow the server to look like different types and versions of webservers, depending on the virtual host being accessed.

5.3.5 Sharing IP Addresses with Attack Hosts

Attacks and background traffic need to be sourced from the same set of IP addresses so attack detection is not just a simple matter of learning the IP addresses of attacking hosts and issuing alerts when traffic is seen from these hosts. Network testbed operation, however, is made easier and more realistic if attacks are launched from separate real machines from background traffic generating machines. The advantages of launching attacks from machines other than the traffic generators include the following:

- Evaluation administrators can moniter all of the traffic to and from the attacking machine's network interface card to record evidence of each attack action.
- All components of the attack are guaranteed to have correct IP, TCP, and application-level actions when run from a physical machine with only one IP address, the address of the attacker.
- Attacks run from Windows NT platforms can be launched from a real Windows NT host without having to simulate Windows NT on the Linux background traffic machine.

- Possible interference between attack scripts and background traffic, such as excessive CPU, memory, or disk usage that could affect performance of background traffic, is avoided when attacks are launched from separate physical hosts.

For these reasons, attacks needed to be lanuched from separate physical hosts. To make this possible and ensure that attacks could not be detected by source IP address alone, addresses needed to be shared between background and attacking machines. To enable this sharing, custom scripts and a special network design were used to transfer virtual IP addresses between background and attack generation machines.

The Linux attackers were able to switch IP addresses with the Linux background traffic generators by running a short script. This script alerted the background traffic generator to disable the requested virtual interface and bring up one for the IP address that the attacker was relinquishing. The attacker was then reconfigured with its new address. In this process ARP entries and routes needed to be updated for the new location of both IP addresses. Attacking machines were situated on a private network behind traffic generation machines to ensure that this IP switching process was invisible to the sniffers collecting data for evaluation participants. Windows NT attacker hosts were not able to participate in this IP sharing process and maintained static IP addresses throughout each run and during the evaluation. If not done carefully, this IP switching could interrupt background traffic sessions in two ways: 1) If the addresses are swapped while a network session is in progress the session will not close properly and will appear to hang until an inactivity timeout is exceeded, and 2) If an address is needed by the background traffic generator, but is being used at the attacking machine then the desired background traffic session will be canceled as the IP address from which to source the address is not available.

To avoid this traffic interruption, an off-line check is performed during background traffic generation process to ensure that there are no conflicts between the attack schedule and the background traffic schedule. This off-line check avoided most problems, however there were some cases where problems remained when background traffic was destined for an attacking host while the virtual host IP address was in use there. The off-line check was not performed for mail messages being generated by mail-reading sessions when users replied to another message. When an IP address was in use by an attack host, mail destined to that machine was delivered to the attack host rather than the background traffic generator. These messages would bounce since the sendmail process running on the attacker was not configured to accept mail for the new IP address, only its original address. In future evaluations, the IP switching process will need to restart the sendmail process for each address change, or a check needs to be performed to ensure that such mail messages are not sent.

Some manually rescheduled attacks interfered with the background traffic and requested IP addresses that were "in use" by the background traffic generator. This caused occasional sessions to appear to hang. Most often, simulation administrator error contributed to this. In future evaluations, the IP switching software will need to automatically perform a check to see if an address is in use before doing the switch and, if so, alert the user rather than blindly cut the session short.

5.3.6 Internet Web Server

Aesop.eyrie.af.mil, a Linux server on the outside network, provided Internet web services that replicated a collection of thousands of real Internet web servers. Aesop used the same kernel modifica-

tion as the background traffic generators, Calvin and Hobbes, but added to it a custom web server that looks like a number of different varieties of web server. It serves unique sets of pages for each of 2,500 virutal hosts configured on the machine. The custom server operates in two ways, first in prepration for simulation and then during network testbed operation.

Prior to the simulation, the server is initialized by connecting it to the Internet and configuring it to cache web pages from each web server to be simulated, as well as recording logs of sessions to be used in the evaluation. First a list, shown in Table 5-1, was compiled of the top 20 web sites to be visited by personnel at the simulated Air Force base. Then, a custom software autonoma generates web surfing sessions by picking one of the 20 starting sites, according to its popularity, and beginning to surf on that site.

TABLE 5-1

20 "MOST POPULAR" WEB SITES SURFED

47	HTTP://www.geocities.com/
46	HTTP://www.usatoday.com/
46	HTTP://altavista.digital.com/
43	HTTP://www.zdnet.com/
43	HTTP://www.yahoo.com/
43	HTTP://www.boston.com/
42	HTTP://www.afpc.af.mil/
42	HTTP://www.afmc.wpafb.af.mil/
42	HTTP://cnnsi.com/
38	HTTP://www.msnbc.com/
38	HTTP://home.netscape.com/
37	HTTP://www.cnn.com/
34	HTTP://www.microsoft.com/
34	HTTP://home.microsoft.com/
33	HTTP://www.washingtonpost.com/
33	HTTP://209.1.112.251/
29	HTTP://www.lycos.com/
27	HTTP://www.aol.com/
27	HTTP://afpubs.hq.af.mil/
26	HTTP://www.af.mil/

These sites served as starting points for web sessions generated for the 1999 simulation. The numerals indicate the number of times each was used as a starting point.

Once the session starting point is determined, hypertext links to be followed are chosen similiarly to the way a real user might surf the web. There are equal probabilities of browsing links on the page in a depth first manner, in a breadth-first fashion, or jumping to a new site (chosen from the initial list of twenty site) and a 10% chance of stopping. This loop is iterated until the session stops and a new

session generation routine is begun. A log kept of each generated session lists the webpages surfed and all files downloaded. Html, image, and other files that are downloaded are cached and indexed in a database for playback during simulation. Naturally, during this processs of randomly surfing webpage links, it is likely that the autonoma will hit pages on webservers other than the starting server. Each new webserver is tallied and a virtual host is created to simulate that webserver with a virtual host. For the 1999 evaluation, 750 unique web surfing sessions were created and recorded using this process.

This initialization process yielded 2,500 webservers that needed to be simulated by the Internet web server. During the simulation run the server was connected only to the simulation network, and emulates these 2,500 Internet web servers for the network test bed. Background web traffic sessions are generated by choosing a session start time with an exponential interarrival time depending on the time of day, choosing one of the 750 recorded web sessions, and writing a script that re-ran the session with varied wait times between web page transitions. These scripted sessions generate real traffic like real users surfing the web. Virtual interfaces and custom HTTP server software responded to web queries by serving pages from the cache. More discussion on how these web session scripts are generated is found in section 7.4.

5.3.7 Expect and Perl for Automation

Expect is a common UNIX scripting language for automating interactive text-based sessions. For the 1999 evaluation, a script written in Expect allowed specification of every session with simple configuration files called EXS scripts. The EXS script specifies all details of the interactive session from the virtual IP address and username from which to source the session to a list of commands to be executed, with typing times and waiting times for each command. These files were given the file extension ".exs" thus the name "EXS scripts." In fact, they are a series of sequential inputs to a Expect script. For each command typed into the interactive session by Expect, waiting and typing times were modeled to look like a human typist and were specified in the EXS script. Documentation for the format of EXS scripts is provided in sections 4-5 of Appendix B and an example is provided in section 6 of Appendix B.

Thousands of statistically-based background traffic sessions were generated for each simulation day. A set of Perl scripts were used to generate the EXS scripts necessary for these sessions. The generation procedure and statistics modeled are discussed in section 7. EXS scripts were generated on a non-simulation machine and then uploaded to the background traffic generators, Calvin and Hobbes. Attacks and attack traffic were simulated using the same Expect script approach, however the sessions were generally created by hand, and the scripts were placed on the attacking hosts. Once all scripts were in place, the testbed was ready for a simulation run, discussed in the next section.

5.4 ADMINISTRATION

For the 1999 evaluation, the network testbed was used to generate five weeks' worth of data. These were five-day weeks and each 22-hour day was simulated in real time. Ideally this schedule allowed the testbed to run continuously for days at a time, allowing two hours each day (between 6:00 and 8:00 a.m.)

each day to retrieve the previous day's data and prepare for the next day. The goal was to produce the data as time-continuously as possible. However, this schedule was not always possible due to unexpected administrative tasks such as fixing a machine or service, troubleshooting an attack, or verifying the previous day's run. Despite the amount of automation used, the testbed required much manual administration throughout the simulation runs. A time line in Figure 5-3 shows the simulation run in real time, but with the time and date of the testbed network offset from the "wall clock" time and date. The list of activities shows the major administrative activities that were carried out prior to beginning a day's run and after the run was complete.

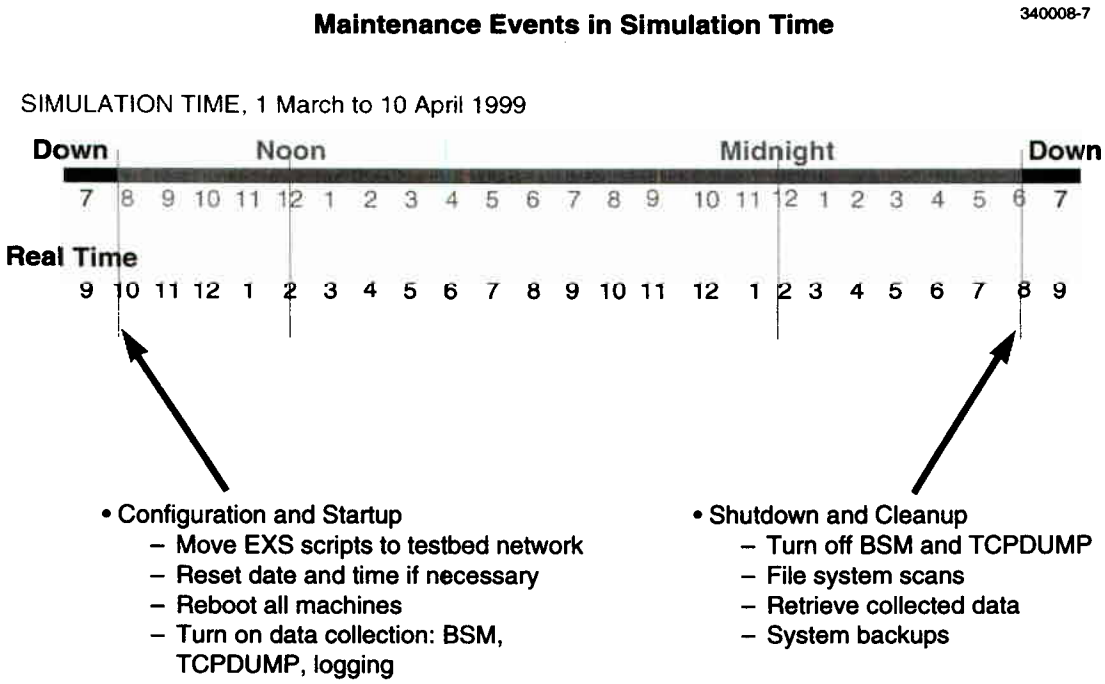


Figure 5-3. A time line showing how simulation time and wall clock time were offset during simulation runs and the maintenance events conducted during simulation.

The following subsections describe simulation date and time terminology and the procedures used to start the testbed at the beginning of a run and the data collection process following each day's run.

5.4.1 Simulation Time and Date

In the 1999 evaluation, each week consisted of five days, Monday through Friday, with 22 hours each day, 8:00 a.m. to 6:00 a.m. Testbed days were simulated in real time and as though they were

consecutive, starting on Monday, 1 March 1999, at 8:00 a.m. Individual days are often referred to with the notation "week-day." For example, 4-3 refers to the third day of the fourth week. Each simulation day begins by setting the date and time to the desired fictitious date and time. Table 5-2 gives the start and end dates and times for each week of data produced.

TABLE 5-2

DATES AND TIMES OF THE START AND END OF EACH WEEK OF DATA PRODUCED FOR THE 1999 EVALUATION

Week	Description	Start	End
Week 1	Training data, without attacks	3/1/99, 8:00 a.m.	3/6/99, 6:00 a.m.
Week 2	Training data, with attacks	3/8/99, 8:00 a.m.	3/13/99, 6:00 a.m.
Week 3	Training data, without attacks	3/15/99, 8:00 a.m.	3/20/99, 6:00 a.m.
Week 4	Testing data	3/29/99, 8:00 a.m.	4/3/99, 6:00 a.m.
Week 5	Testing data	4/5/99, 8:00 a.m.	4/10/99, 6:00 a.m.

Two things should be noted. First, the last week in March was skipped—no simulation activities were simulated during those times/days. Second, Daylight Savings Time began on Sunday, 4 April 1999. This means that dates and times stored in "UNIX format" (seconds from 1 January 1980), such as tcpdump times and dates, will convert an hour ahead for Week 5 (or an hour behind for weeks 1-4). All attack truth listings and other analysis performed on the 1999 data are based on simulation days beginning at 8:00 a.m. An appropriate conversion must be performed when working with the data due to differences in time zone and daylight savings time between Eastern Standard Time (Weeks 1-4) and Eastern Daylight Time (Week 5) and wherever the data is being used.

5.4.2 Startup and Shutdown

Starting the testbed network at the beginning of a day's run is not a trival process. An attack schedule and specific attack scripts for that day's run were generally created and tested days or weeks prior to a given simulation run. The scripts were then placed on the attacking machines and registered with a batch script written in Perl that would launch them at the appropriate times during simulation. EXS scripts for background traffic sessions were generated with the Perl based generation scripts. Each day of simulation required thousands of these EXS scripts, one for each network session. Once generated, they were pushed out to the traffic generation hosts and registered with a similar batch script that ran each session at the specifed time.

Once all of that day's scripts were in place around the testbed, the simulation was ready to begin. Simulation began by setting the date and time on each network host to the desired start time, 7:30 a.m. on the next simulation day. Machines were then rebooted so that each would start fresh for that run.

Aesop was reboot an hour or so prior to this as as it took much longer to boot due to the large number of virtual hosts. As soon as the machines were back up, the batch scripts that controled both background and attack traffic generation were started and the simulation was ready to go. The "day" officially began at 8:00 a.m. so no EXS scripts were launched prior to that time.

Despite the large number of sessions and services that were automated, many attack sessions, some background traffic sessions, and administrative tasks required manual human attention during these runs. A few of these manual activities included:

- Complex background traffic actions, like an administrator installing a new version of netscape on the Solaris server, or the administrator upgrading the powerpoint macro with which webserver statistics are put into a power point presentation.
- Complex attacks that involved X-window or Microsoft Window interaction – netcat, netbus, named.
- Administration of servers: Aesop, the Internet web-server sometimes required restarting when the web server program crashed. On hume, the mail server, Windows NT Resouce Kit MailSrv, sometimes needed the mail spool cleaned out and the server restarted, when it got bogged down.
- Monitoring of the batch programs that controlled background and attack activities to ensure proper opreation. Occasionally if an attack failed in an unexpected way the batch script could hang and require manual restarting.
- Rebooting hosts or restarting services following successful denial-of-service attacks.

Each day's run was complete twenty-two consecutive hours after the 8:00 a.m. start time, at 6:00 a.m. of the following simulation day. Traffic generation ceased at this time and data collection processes were shutdown. When shutdown is complete, the network testbed is ready for data retrieval. Data collected during the simulation run is copied from the network testbed and stored for use by evaluators and participants as part of the intrusion detection corpus.

5.4.3 Simulation Data Collection

A large amount of data (often many Gbytes) was collected from the network testbed for each day of simulation. At the end of each run, all data was gathered up and moved off the network testbed for analysis and verification to ensure that simulation had gone satisfactorily. Verification that the background traffic simulation had proceeded correctly involved ensuring that a reasonable number of complete sessions of each traffic type were found in the tcpdump network data. Verification of attacks was much more time-consuming and is discussed in section 6.7.

Most data collected and retrieved from the network testbed is a part of the intrusion detection corpus and is used by evaluation participants for system and algorithm development or by evaluators for blind evaluation of systems and algorithms. The only data retrieved from the network testbed after each simulation run but that is not part of the corpus are complete tape backups of each victim host in the network. These backups are made to ensure that each host could be rebuilt relatively easily if it were to experience failure. Each source of data collected and retrieved from the network testbed is described below.

5.4.3.1 Tcpdump Data

More than half of the participating intrusion detection systems relied on network trace data for input. This trace of network traffic was collected with a UNIX utility called tcpdump and is often referred to simply as tcpdump data.

Two network sniffers were setup on the network test bed, one just inside the Eyrie Air Force Base gateway to the Internet, and one just outside that gateway. These sniffers are referred to as "inside" and "outside" sniffers and their respective network traces are referred to as "inside" and "outside" tcpdump files. The inside sniffer was situated so that intrusion detection systems using the data could see traffic sourced from and destined to machines on the simulated Air Force base. This allowed detectors to detect attacks that originated from hosts on the simulated (Eyrie) Air Force Base. The outside sniffer data was situated so that intrusion detection systems could observe sessions and packets entering and leaving the Air Force Base.

The tcpdump files are binary traces of the network traffic, collected with the command:

```
tcpdump -i <interface> -s 60000 -w XXXside.tcpdump
```

The "-s" flag is included to ensure that even the longest packets are captured in their entirety. The tcpdump command can also be used to generate a textual representation of some of the data in each network packet that is collected. Table 5-3 shows an example of this textual output. Each line summarizes the header of each network packet, giving the time, source IP address, source TCP port, destination IP address and port, and information about what the packet contains. The first two lines, for example, show a Domain Name Service (DNS) query and reply, in which host 172.16.112.20 requests the IP Address Correspondence to "lumbala.orange.com". The second line in the reply from the DNS server, 192.168.1.10. The tcpdump binary file contains every network packet and its contents. The tcpdump data error output was monitored after every day of data collection and it indicated that packets were never dropped. Tcpdump data thus should contain all packets sent over the testbed.

TABLE 5-3

TEXT OUTPUT FROM THE tcpdump UTILITY

```
09:00:40.682681 172.16.112.20.53 > 192.168.1.10.53: 26839 A? lambda.orange.com. (35)
09:00:40.683705 192.168.1.10.53 > 172.16.112.20.53: 26839* 1/1/1 A 195.73.151.50 (91)
09:00:40.703176 172.16.114.207.1024 > 195.73.151.50.25: S 4180099195:4180099195(0) win 512
<mss 1460>
09:00:40.703496 195.73.151.50.25 > 172.16.114.207.1024: S 982117809:982117809(0) ack
4180099196 win   32736 <mss 1460>
09:00:40.706315 172.16.114.207.1024 > 195.73.151.50.25: . ack 1 win 32120 (DF)
09:00:40.874243 192.168.1.10.53 > 172.16.112.20.53: 57008 PTR? 207.114.16.172.in-addr.arpa. (45)
09:00:40.875567 172.16.112.20.53 > 192.168.1.10.53: 57008* 1/1/1 PTR pigeon.eyrie.af.mil. (134)
09:00:40.877054 192.168.1.10.53 > 172.16.112.20.53: 57009 A? pigeon.eyrie.af.mil. (37)
09:00:40.878077 172.16.112.20.53 > 192.168.1.10.53: 57009* 1/1/1 A 172.16.114.207 (102)
09:00:40.954857 195.73.151.50.25 > 172.16.114.207.1024: P 1:85(84) ack 1 win 32736 (DF)
09:00:40.968468 172.16.114.207.1024 > 195.73.151.50.25: . ack 85 win 32120 (DF)
09:00:40.991179 172.16.114.207.1024 > 195.73.151.50.25: P 1:27(26) ack 85 win 32120 (DF)
09:00:40.991443 195.73.151.50.25 > 172.16.114.207.1024: P 85:111(26) ack 27 win 32736 (DF)
09:00:40.992448 172.16.114.207.1024 > 195.73.151.50.25: P 27:53(26) ack 111 win 32120 (DF)
09:00:40.992690 195.73.151.50.25 > 172.16.114.207.1024: P 111:159(48) ack 53 win 32736 (DF)
09:00:40.993553 172.16.114.207.1024 > 195.73.151.50.25: P 53:94(41) ack 159 win 32120 (DF)
09:00:40.993800 195.73.151.50.25 > 172.16.114.207.1024: P 159:207(48) ack 94 win 32736 (DF)
09:00:40.994697 172.16.114.207.1024 > 195.73.151.50.25: P 94:129(35) ack 207 win 32120 (DF)
```

5.4.3.2 Solaris BSM and Windows NT Audit

Host-based auditing information was collected from both the Solaris 2.5 victim machine, pascal.eyrie.af.mil and the Windows NT 4.0 victim machine, hume.eyrie.af.mil. In the case of the Solaris machine this data is called BSM Audit data, referring to the Basic Security Module in the operating system that makes this data stream possible. For the Windows NT machine, it is referred to as Windows NT Audit data.

The BSM audit data was collected with full auditing enabled and provides a fairly complete picture of the activity that occurred on the Solaris machine during each simulation day. All events that could be audited, were audited. It records every system call, including "exec" calls, file opens, and file closes. Two utilities are provided under Solaris to parse this audit data. Praudit parses the binary log file and provides textual output. Table 5-4 hows an example of a single BSM auditing record that has been converted into text using praudit. This example shows an "execve" system call in which the kernel starts a /usr/sbin/quota process. Sun/Solaris documentation contains more information on the format of these records. In addition to praudit, a utility called auditreduce is provided. It can select records specified by type, by date entry, and other attributes.

TABLE 5-4

EXAMPLE OF A SINGLE BSM AUDIT RECORD AS CONVERTED TO TEXT BY THE PRAUDIT UTILITY

header,150,2,execve(2),,Mon Mar 01 07:32:19 1999, + 331077757 msec path, /usr/lib/fs/ufs/quota
attribute,104555,root,bin,8388614,187986,0
exec_args,1,
/usr/sbin/quota
subject,root,root,other,root,other,262,257,0 0 172.16.112.50
return,success,0
trailer,150
This is an execve system call in which a /usr/sbin/quota process is created by the kernel.

The Windows NT audit data was collected with a sub-set of possible auditing enabled, and provided a less-complete picture of the activity on the NT machine than the BSM data on the Solaris machine. Figure 5.7 shows the configuration settings used for Windows NT auditing in the 1999 Evaluation. All auditing was turned on, except for individual files and directories. Complete file system auditing was turned on after the evaluation and found to not load the NT server excessively. This type of auditing will be used in future evaluations. All events are logged to one of three audit log files, Security.Evt, System.Evt, and Application.Evt, which are stored in C:\WINNT\system32\config\. Log

settings were adjusted to allow very large log files, approximately 200 megabytes. This ensures that the log files do not fill up and begin to overwrite earlier log entries. In addition, it was specified that audit logs should not be automatically cleared at any time. Only the Administrator account has the ability to manually clear the audit logs. At the end of each simulation day all three eventlog files were retrieved from the Windows NT victim host, hume.eyrie.af.mil.

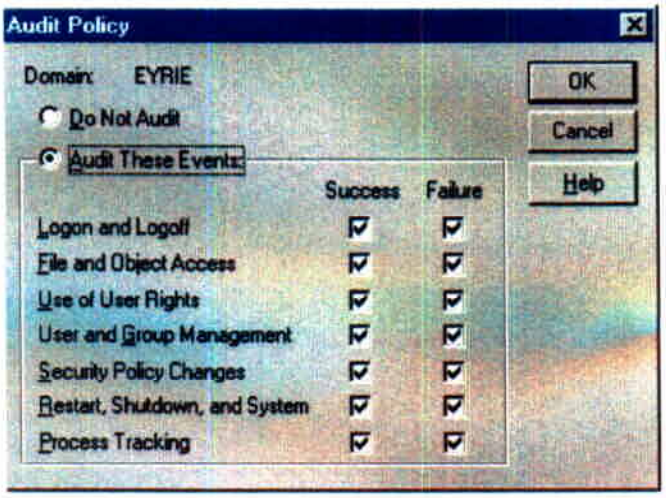


Figure 5-4. Windows NT auditing configuration used for the 1999 evaluation.

The Windows NT audit logs may be read and parsed by making use of the Windows NT Event Viewer. Work is being done on a program that will be able to perform functions similar to those of auditreduce on the Windows NT audit logs. Figure 5-5 shows an example of two records from the Windows NT Security log showing a process as it starts and as it exits.

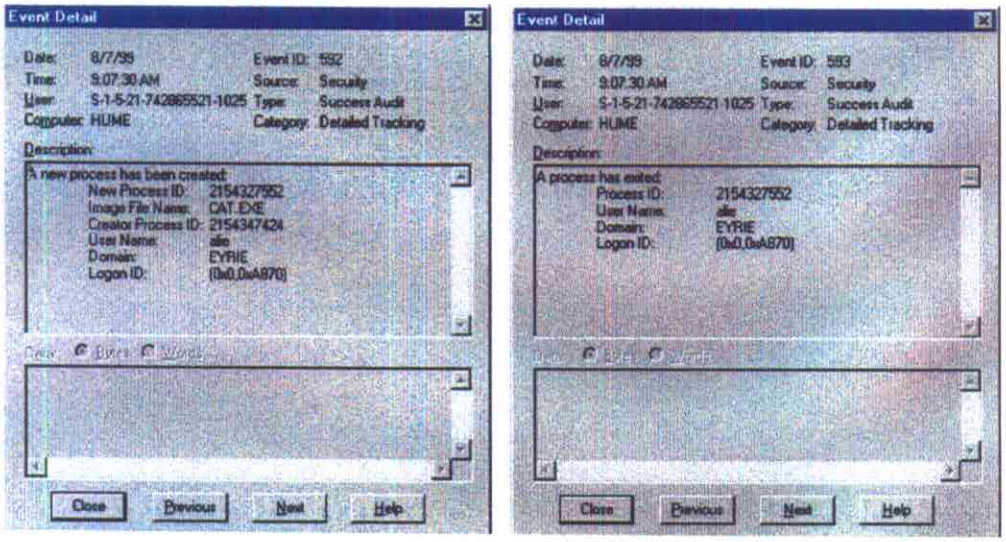


Figure 5-5. An example of a Windows NT Security Event log entries, as displayed by the Windows NT event viewer. The first shows a process creation record while the second shows that same process as it exits.

5.4.3.3 Selected files, including system log files

Certain files from each victim host were provided to evaluation participants. A complete backup of certain files and directories on each victim host was collected, archived, and distributed as a part of the corpus.

These files and directories were archived with the UNIX tar command. For example, the following command was used to collect all files and directories in the /etc directory:

```
tar -cvf etc.tar /etc/*
```

A slightly more complicate regular expression was used to collect every user's "dot-files." An example "dot-file" is the .cshrc file that resides in a user's home directory and contains user-specific commands to execute and configuration parameters for that user's shell. These files were collected from the UNIX servers with a command similar to the following:

```
tar cvf pascal_dot.tar /home/*/.???*
```

A full list of all files and directories collected from each UNIX victim host follows:

- Pascal (Solaris 2.5)
    - o /var/log/*
    - o /var/cron/*

- /var/adm/*
- /var/audit/*
- /etc/*
- User dot files

- Zeno (SunOS 4.1.4)
  - /var/log/*
  - /var/adm/*
  - /etc/*
  - User dot files
- Marx (RedHat 5.0)
  - /var/log/*
  - /etc/*
  - User dot files

From the Windows NT server, hume, the entire Windows NT Registry was collected and archived after each run. The Registery was collected with the following command, contained in the Windows NT Resource Kit:

```
c:\ntreskit\reg_back.exe
```

Only the Windows Registry was collected from the Windows NT Victim, as it is the only real equivilant to the "/etc" and the "dot-files" on the UNIX systems.

5.4.3.4 File System Info

Two types of file-system information were collected for the 1999 evaluation. The first is an ASCII text list of the information contained in each inode of the Solaris Victim's (Pascal.eyrie.af.mil) file system. The second is a "long listing" of all files on each victim system.

A custom inode reading utility, provided to evaluators by SRI-Derbi, was run daily to collect information on every inode in the Solaris victim's file system. The data provided by this scanner was required by the SRI-Derbi intrusion detection system to detect attackers [22]. The following command collected this information:

```
/.sim/home/bin/derbi-stat-ufs -af5
```

The inode information reader provides the latest read/write/modify times of each file/directory on the system without changing these times. A sampling of this output is shown in Table 5-5.

TABLE 5-5

EXAMPLE OF THE DERBI FILE SYSTEM SCANNER OUTPUT

6 pascal 0 1 / 1 0 0 0 8388608 2 16877 32 0 0 922792196 922792195 922792195 1024 0 32 6b68f30253d7f411af0fa882239eb53a 0
6 pascal 10 lost+found 1 / 1 24 44 24 8388608 3 16832 2 0 0 922974703 886172182 886172182 8192 0 32 a749b55d2bef57700e2f50c6cf68f7d0 0
6 pascal 3 usr 1 / 2 44 56 16 8388614 2 16877 31 0 0 922974703 891017559 891017559 1024 0 0 0
Shown are inodes for the /, /lost+found, and /usr, directories on the Solaris victim machine. The information on each line summarizes each inode including access and modification permissions and times.

Although the information collected was output and stored in an ASCII file format, it was used exclusively by the SRI Derbi intrusion detection system. This information was collected in lieu of making and distributing complete file system backups of each victim host as had been done in 1998. Unfortunately this information was collected after various parts of the file system had been tar'd up for inclusion in the corpus. This destroyed the latest read times on those inodes from which data was collected using tar.

The second type of file system listing provided was a long listing of each file on each of the four victim hosts: Solaris, Windows NT, SunOS, and Linux. The file system listing was performed with the UNIX find command:

```
find / -ls > file_system_listing.ls
```

An example of the output of this file system listing is shown in Table 5-6.

TABLE 5-6

SAMPLE OF THE FILE SYSTEM LISTING

2	1 drwxr-xr-x	32	root	root	1024	Mar 30 06:09	/
3	8 drwx------	2	root	root	8192	Jan 30 1998	/lost+found
2	1 drwxr-xr-x	31	root	root	1024	Mar 27 1998	/usr
A sample of the file system listing on pascal from Monday, week 1, showing the /, /lost+found, and /usr directories. All directories and files are listed in this fashion.							

On the Windows NT host (hume.eyries.af.mil) All files were listed using the following command, part of the Windows NT Resource Kit:

```
c:\ntreskit\posix\find / -ls
```

This Windows find command produced output very similar to that shown in Figure 0-2. These listings are distinct from the SRI Derbi inode reader in that they provide a textual long listing of every file on the system, while the inode reader provides a list containing information from every inode on the system, including all information about each.

5.4.3.5 File System Backups

Complete backups of each server on the testbed network were collected following each simulation day's run, however they were not distributed to participants as a part of the corpus. These backups contained giga-bytes of data and would have been very difficult to efficently distribute to participants. The SRI-Derbi filesystem scanner output was distributed in lieu of full filesystem dumps. Backups were collected to serve as a record of the state of each machine at the end of each simulation day. They can be used if more information is necessary concerning a particular simulation day.

6. ATTACKS

A computer or network intrusion, sometime called an attack, is a sequence of related actions by a malicious adversary whose goal is to violate some stated or implied policy regarding appropriate use of a computer or network. Attack components, sometimes referred to as exploits, are steps in carrying out a larger attack tree or scenario. The emphasis of this evaluation was on detection and identification of attack components, not attack trees or scenarios, since the detection systems being evaluated were geared mainly toward detection of attack components and not scenarios. This section discusses attack components in the 1999 Off-Line Evaluation, how they were classified, and details of attack instances themselves. In this section, the terms *attack*, *attack component*, and *exploit* all refer to sets of malicious actions with the goal of violating a stated or implied security policy. They could be thought of as part of a larger hypothetical attack tree or scenario, however, such a larger scenario was not specifically designed or carried out in the evaluation. An *attack instance* refers to a particular instance of an attack in the evaluation data. Over 200 instances of 58 attack types were run in the 1999 Off-Line Evaluation. Full descriptions of each of the 58 types of attacks are included in Appendix A.

6.1 ADVERSARY MODEL

In attempting to accurately model attacks seen in the real world, it is necessary to consider who the adversary is, what their goals are, and how they might carry out an attack. Two types of adversaries were considered in the 1999 evaluation; first, a relatively unsophisticated amateur hacker, and second, a more sophisticated, professional, or nation-state attacker. The amateur hacker is merely testing his or her skills and probably does not have a very specific goal in mind for their actions. The more sophisticated nation-state attacker however has a firm view of his or her goal and may very well be attempting to gain access to some particular piece of information or deny service during a precise period of time. The amateur is likely to carry out the attack using attack programs and scripts found on the Internet, while the sophisticated attacker is likely to go to great lengths to hide their actions by creating variations of well-known attacks or creating new attacks from scratch. An example of an adversary model is given in [14].

Attack components were included in the 1999 evaluation test data that matched the abilities of each adversary. Clear attacks are those that might have been carried out by the amateur hacker, where the exploit is carried out in the same form as found on the Internet. Stealthy attacks are modified from their original form, similar to the way a sophisticated hacker might modify an attack to attempt to evade detection. Many of these modifications are made with general knowledge of techniques used for intrusion detection and are attempts to evade these techniques.

6.2 SECURITY POLICY

Computer attacks are a violation of some stated or implied security policy. This evaluation employs both types of policies. The implied policy included the following assumptions:

- Denying or attempting to deny service to legitimate users constitutes an attack.

- Any sort of probing to gain information in preparation for an attack is considered an attack.

To complement the implied policy, the stated security policy gives some loose ideas about the types of user behavior that are allowed on these simulated Air Force base computers. (The full text of the security policy is in Appendix C.) The stated policy sets aside a "secret" directory and files on each Base server. These may only be accessed by specified users, and then only under certain circumstances. Access by other users or under other conditions specifically constitutes and attack. The secret files are "flags" or goals for some attacks and allow evaluation of how well intrusion detection systems can detect illegal accesses to these files.

6.3 ATTACK CATEGORIZATION

Attacks in the 1999 evaluation were placed in categories to compare how well various intrusion detection systems addressed each sub-set of attacks and thus determine coverage of the attack space. In addition, classifiying attacks allowed intursion detection systems to be evaluated only in the categories that the system developer has chosen. Evaluation participants were asked to specifiy ahead of time which attack categories their intrusion detection system would be scored in. This avoided irrelevant results that could occur if, for example, a host-based intrusion detectection system were scored against attacks aimed at a different host.

Attack categories along two main axes were used to divide the set of all attacks: the goal of the attack and the platform or operating system of the victim computer. The categories were well suited for specifying the capabilities of most intrusion detection systems in the evaluation, however they did not meet the needs of every system. In the future, more fine-grained categories should be used to further subdivide existing categories. The following two sections discuss each axis of this broad categorization.

6.3.1 Attack Goal

In the 1999 evaluation, all attack components had specific goals that included obtaining information about a computer/network system with intent to use that information as part of an exploit, achieving some level of privilege on a computer system not specifically granted by a system administrator, or violation of some explicitly stated security policy. Five major attack categories were chosen to group attack components with regard to the actions and goal of the attacker.

- **Denial of service** (DoS) attacks have the goal of limiting or denying service provided by to a user, a computer, or network. A common example is the SYN-Flood attack, where the attacker floods the victim host with more TCP connection requests than it can handle, causing the host to be unable to respond to valid requests for service.

- **Probe** attacks have the goal of gaining knowledge of the existence or configuration of a computer system or network. A common example is the IPsweep attack where the attacker sweeps one or more IP addresses in a given range to determine which addresses correspond to live hosts.

- **Remote-to-local** (R2L) attacks have the goal of gaining *local* access to a computer or network to which the attacker previously only had *remote* access. Examples are attempting to guess a password to illegally login to a computer system and the "imap" buffer-overflow, where a specifically crafted binary string sent during an "imap" mail connection yields access to the Linux system, for a particular version of imapd.

- **User-to-root** (U2R) attacks have the goal of gaining *root* or *super-user* access on a particular computer or system on which the attacker previously had *user*-level access. Common examples of U2R's are buffer-overflow attacks on various operating system programs, including eject, ffbconfig, and fdformat on Solaris.

- **Data** attacks have the goal of gaining access to some piece of information (file or directory) to which the attacker is not allowed access according the stated security policy for the 1999 Evaluation. Many U2R and R2L attacks had an end-goal of accessing a "secret file" given the new, elevated privilege. These attack instances were classified as either U2R or R2L and also considered Data attacks. Other data attack instances occur when valid users breaking the security policy by, for example, accessing a secret file from a remote, un-encrypted login or emailing a secret file off the host.

6.3.2 Victim O/S

In addition to being categorized by attack goal, attacks were also categorized by platform or operating system of the attack victim computer. These categories allow intrusion detection systems that detect attacks only against some of the victim platforms to specify those attacks that they are trying to detect. Reliance on system-level auditing information makes it likely that a host-based intrusion detection system will only be designed to detect attacks against that particular platform. Platforms that attacks were run against in the 1999 evaluation were:

- **Solaris**—Pascal.eyrie.af.mil, 172.16.112.50, was the Solaris 2.5 victim host.

- **Windows NT**—hume.eyrie.af.mil, 172.16.112.100, was the Windows NT 4.0 victim.

- **SunOS**—zeno.eyrie.af.mil, 172.16.113.50, was the SunOS 4.1.4 victim.

- **Linux**—calvin.eyrie.af.mil, 172.16.112.20, RedHat 5.0, and marx.eyrie.af.mil, 172.16.114.50, RedHat 4.2 were both victims of attacks in the evaluation.

- **Cisco**—loud.eyrie.af.mil, 172.16.0.1 and 192.168.0.1, was the Cisco IOS v.11.3 rev. 4 router which also served as a victim host.

- **All**—The All category contained attacks whose victims were a range of machines (an entire subnet for example), indiscriminate to the operating system of the hosts.

6.4 NEW VS OLD

An emphasis of the 1999 evaluation was to evaluate the performance of intrusion detection systems on "new" attacks, that is, attacks that the intrusion detection system might not have see before or been specifically designed to detect. For this reason, all attack instances were also placed into categories as being either new or old. The distinction has a specific meaning in the context of the 1999 evaluation.

- An attack is considered to be "new" if it had not been run in either the 1998 training data, the 1998 test data, or the 1999 training data.

- An attack is considered to be "old" if it had been run in either the 1998 training data, the 1998 test data, or the 1999 training data.

Appendix A lists all attacks used in the 1999 evaluation and indicates which were considered new. These new attacks were developed in the following ways:

- Exploit scripts and procedures were found on the Internet and implemented for use on the testbed. Examples are Dosnuke, Queso, AnyPW, and Ntfsdos.
- Vulnerabilities were discovered on the Internet and specific implementations were developed in-house to exploit them. Examples are Ppmacro, Selfping, and ResetScan
- Custom attacks were developed under contract specifically for the evaluation. Examples are the ARPpoison, Tcpreset, Sshtrojan, and Ssh-processtable attacks.

Results, presented in the Results Technical Report, compare performance of intrusion detection systems on old and new attacks.

6.5 STEALTHY VS CLEAR

Another goal of the evaluation was to determine whether systems could detect stealthy attacks designed to avoid detection by intrusion detection systems. Each attack instance in the evaluation was labeled either clear or stealthy. Clear attack instances conform to the amateur adversary model by appearing to be run by a novice hacker that knows little about how to effectively cover his/her tracks. Stealthy attack instances conform to the more experienced professional adversary model by making use of custom techniques to better evade detection. In the evaluation, attack terms are defined as follows:

- Clear attacks are those used in the simulation in a fashion very similar to the way the attack appeared on the Internet, or similar to the way the attack appeared in the 1998 evaluation or training data. An example would be running an attack script as found on the Internet and without modification.

- Stealthy attacks are those that have been modified, to evade detection, from the form in which they were found on the Internet. User-to-roots and probe attacks were made stealthy in the 1999 evaluation, via a number of different techniques.

The following summary lists the many possible ways used to make attacks stealthy in the 1999 evaluation:

- U2R attacks and some R2L attacks were made stealthy with a number of techniques, and combinations thereof. One or more techniques in the following list were employed for an attack instance to be considered stealthy.
  - Spread the attack across multiple network sessions.
  - Delay the execution of the attack with the *at* command, with *cron*, or by predicating the execution on another unrelated event – like a user logging in or a mail message being received.
  - Embed the attack in sessions that look like normal user traffic.
  - Modify buffer overflow machine code so that instead of spawning a root shell some other action is taken—most often a file gets *chmod*'ed, allowing the attacker access to it.
  - Change or replace any command-line strings that might tip-off a keyword spotting system to look like normal commands or filenames.
  - Encode source code for attack binaries or scripts to hide any possible key strings. Uuencode was used, as were various techniques to insert dummy characters into scripts with *sed* and *awk*.
  - Use shell variables to hide key-strings from the command line and use shell or PERL scripts for faster attack execution.
  - Redirect attack output, which might also contain identifiable key-strings, to a file or */dev/null* rather than standard output.
- Probe attacks, portsweeps, and ipsweeps were made stealthy by decreasing the number of machines or ports probed and increasing the time delay between consecutive packets or connections. In the 1999 Evaluation, a sweep was considered stealthy if there were 10 or fewer packets/connections or more than 59 seconds between consecutive packets/connections. In addition, some sweeps of port or IP ranges were conducted in random, rather than ascending, order or the source IP address of the probe changed with each consecutive packet or connection.

More detailed discussion on stealthy attack instances and techniques for making attack stealthy can be found in [18].

6.6 RUNNING ATTACKS

Most attack instances in the 1999 evaluation were automated using the same Expect-based scripting tools that were used to automate background traffic, described in secton 5.3.7. Attacks were generally run from special hosts dedicated to attacking. This helped avoid interference between attack generation and background traffic generation, and allowed attack instances to be documented fully with respect to network activity. Some attack instances could not be scripted because they required windowing system interactions and were run manually. Other attack instances were run from the console of the machine that was being attacked or from the console of one of the Eyrie Air Force Base servers to simulate an attacker who had gained physical access to the host. These console attacks also could not be scripted and were run manually.

6.7 ATTACK VERIFICATION

Upon completion of a day's simulation run all data was retrieved from the network testbed, and a detailed analysis was performed to ensure that the simulation run had proceeded correctly and to ensure that the attacks had run as planned. The processes of verifying correct background traffic is described in Section 5.4.2. For attacks, the verification involved:

- Verifying that the attack instance had run correctly
- Determining the start time and duration
- Classifying each attack instance (New/Old, Clear/Stealthy, etc.)

As an example, take an instance of the eject attack run on Friday of the second week of test data in the 1999 evaluation, ID# 55.141732. This is a user-to-root attack that was automated and for which an attack-specific network trace was collected. Using this network trace and the attack script, it is possible to verify that the attack consisted of three network sessions, as planned. These are an FTP connection, with FTP-data connection, to transfer the special eject exploit program to the victim host, and a telnet session in which the attacker compiles the program, runs it successfully, and illegally gains root access on the victim. The start time and duration of attack sessions are those of the network sessions that make up the attack and are learned through analysis of the network trace, as shown in Table 6-1. It was also necessary to verify that the same network sessions that appeared in the attack-specific dump file also occurred in the inside or outside dump files if appropriate.

TABLE 6-1

NETTRACKER OUTPUT, ATTACK 55.141732

Date	Start	Duration	Service	Source	SrcPort	Dest.	DestPort
04/09/1999	14:17:28	00:00:25	FTP	206.048.044.050	4711	172.016.112.050	21
04/09/1999	14:17:42	00:00:09	FTP-data	172.016.112.050	20	206.048.044.050	4712
04/09/1999	14:21:17	00:00:29	telnet	206.048.044.050	4713	172.016.112.050	23

Output summarizes the three network sessions that comprise attack #55.141732, an eject attack run on Friday of week 5, the second and final week of test data for the 1999 evaluation. This information is obtained from an attack-specific network trace.

Custom software is used to construct transcripts of the FTP and telnet connections, shown in Tables 6-2 and 6-3. These transcripts allow evaluation administrators to verify that the attack ran correctly and allow classification of the attack instance. In Table 6-2 the transcript of the FTP control connection is shown and allows verification that the session worked as planned and the attack file, called my_code.c, was transported to the victim host. Table 6-3 shows the transcript of the corresponding telnet session and allows verification of the attacker compiling the attack, running it, and accessing the /etc/shadow password file, to which the attacker could not have gained access without first gaining root privilege. Thus the attack instance is verified.

TABLE 6-2

TRANSCRIPT OF THE FTP CONTROL CONNECTION, ATTACK 55.141732

```
::::::::::::::

Apr-09-18:17:12.4711.21.2.dest

::::::::::::::

206.48.44.50:4711=>172.16.112.50:21
04/09/1999 18:17:12 S
220 pascal FTP server (UNIX(r) System V Release 4.0) ready.
331 Password required for marlync.
230 User marlync logged in.
500 'SYST': command not understood.
200 PORT command successful.
150 ASCII data connection for my_code.c (206.48.44.50,4712).
226 Transfer complete.
221 Goodbye.
```

This transcript was a component of attack ID# 55.141732. It shows the attacker successfully logging in via FTP and transporting the eject exploit program, here called my_code.c.

This particular attack is relatively easy to verify since it was run in a clear fashion from a remote network host and since a network trace of the attack was collected of the attack. Had the attack been stealthier, this verification would be more difficult because evidence of the attack might not be available in the session transcripts. Use of encryption to hide the contents of the telnet session or use of the "at" utility to postpone the actual instance of attack beyond the end of the login session would force verification be carried out manually on the victim host itself. This might involve a forensic examination of the file system after the attack to make sure that the desired attack action, reading, copying or deleting a file, had actually been carried out at the scheduled time. In many cases, finding definitive evidence of an attack was a time consuming, manual process, in addition to verifying and recording identification information, like the source, destination, username, network services, and important files used in the attack.

In the future, a possible alternative is to have an attack change the state of a file or some other resource on the target machine after it succeeds. One concern, however, with the use of such a "flag" is that it might generate an artifact which an intrusion detection system could use to detect the attack. The "flag" approach was not used in the 1999 evaluation for this reason.

TABLE 6-3

TRANSCRIPT OF THE TELNET SESSION, ATTACK 55.141732

```
::::::::::::::

Apr-09-18:21:02.4713.23.4.dest
::::::::::::::
206.48.44.50:4713=>172.16.112.50:23
04/09/1999 18:21:02 S
-UNIX(r) System V Release 4.0 (pascal)
login: marlync
Password:
Last login: Fri Apr  9 14:17:38 from 206.48.44.50
Sun Microsystems Inc.   SunOS 5.5        Generic November 1995
Official U.S. government system for authorized use only. Do not discuss,
enter, transfer, process or transmit classified/sensitive national security
information of greater sensitivity than that for which this system is
authorized.  Use of the system constitutes consent to security testing and
Unauthorized use and misuse of government equipment includes, but is not
limited to, playing computer games (hack,doom), sending chain letters,
gambling (sporting pools), personal business, pornography, or anything that
can offend or be construed as sexual harassment.
28-Jul-98
Project Screaming Otter will be using this server as a predeployment
test bed. This may cause a brief reduction in system response and/or
availability. If you need additional computing resources please use
the INMAZ or I-POL servers.
```

TABLE 6-3 (Continued)

TRANSCRIPT OF THE TELNET SESSION, ATTACK 55.141732

NOTE: ALL CLASSIFIED TRAFFIC WILL USE CODE BOOK BLUE-47 FOR THE DURATION.
If you have additional questions or other concerns, please e-mail us at
support@pascal.eyrie.af.mil
You have mail.
pascal> ls

1999	a.out	eject.c	mail	validate.sh
Attacks	bin	FTP	my_code.c	work
Downloads	doc	lib	src	working
NewProjects	docs	lz	temp	

pascal> pwd
/export/home/marlync
pascal> pwd
/export/home/marlync
pascal> gcc my_code.c
my_code.c: In function 'main':
my_code.c:20: warning: return type of 'main' is not 'int'
pascal> ls

1999	a.out	eject.c	mail	validate.sh
Attacks	bin	FTP	my_code.c	work
Downloads	doc	lib	src	working
NewProjects	docs	lz	temp	

pascal> ./a.out
Jumping to address 0xeffff7f0
Jumping to address 0xeffff7f0 B[364] E[400] SO[400]
cat /etc/shadow
root:kum1FrSa/kiFE:10227::::::
daemon:NP:6445::::::
bin:NP:6445::::::
sys:NP:6445::::::

This transcript of the telnet session of attack ID# 55.141732, created by NetTracker, shows the attacker compiling the exploit, running it, and gaining access to the /etc/shadow file to which user marlync would not normally have access.

Analysis was carried out for each attack instance, was very time consuming, but was vital to ensure that the truth against which the evaluation was to be scored was as accurate as possible. All of the information collected in this analysis was contained in two files that described all attacks in the two weeks of test data: the *detections truth list* and the *identifications truth list*, described in the following two sections.

6.7.1 Detection Truth List

The *detection truth file*, sometimes named the *master-listfile-condensed*, serves as the truth against which the evaluation was scored. Raw detection results, submitted by participants, are compared to the detection truth file to determine attacks detected, attacks missed, and false alarms incurred. Each line of the file represents one session of an attack component and gives the start time, duration, victim IP address, ID number, and categorization information specific to that attack instance. One attack instance may be represented by multiple lines in the detection truth file since that instance may occur over several distinct periods of time. Categorization information about the attack instance is indicated with keywords that indicate various features of the attack or categories into which the attack instance is placed. These keywords are chosen for each attack-instance, so all truth file entries that correspond to the same attack instance will be labeled with the same attributes and the same ID number.

Many detection truth list entries are based upon a real network session that has been recorded during the attack. For each such entry it is relevant to provide the source and destination IP address and ports numbers for that session in the list entry. However, some attacks do not instantiate themselves in the form of network sessions. For these attacks, a line is inserted into the truth list that indicates the date, time, duration, and victim of the attack or attack session. Wild cards are inserted for all network information. The detection truth list takes two forms, one is a summarized or condensed version of the other, a more verbose version:

- **Complete or extensive list** gives every network and non-network session that is a component of that attack. For example, a list of every session of a Neptune attack, listing all thousand or so network connection on its own line would be considered a complete or extensive list.

- **Condensed or summarized list** gives only summary entries for each attack instance. Adjacent or time-overlapping entries, with the same destination IP address, are condensed into a single list entry, with the start time and duration modified to indicate the time spanned by the summarized entry. In these summarized list entries the source IP, source port and destination port are replaced with wildcard placeholders to maintain file format.

The scoring procedure uses the condensed or summarized version of the detection truth file as it is much shorter but still contains the information necessary for detection scoring.

TABLE 6-4

A CONDENSED DETECTION TRUTH LISTFILE

41.084031 03/29/1999 08:18:35 00:04:07 srv spt dpt 209.154.098.104 172.016.112.050 1 ps
out,,rem,succ,aDmp,oDmp,iDmp,BSM,SysLg,FSLst,Stlth,Old,llU2R,llSOLARIS
41.084031 03/29/1999 08:19:37 00:01:56 srv spt dpt 172.016.112.050 209.154.098.104 1 ps
out,,rem,succ,aDmp,oDmp,iDmp,BSM,SysLg,FSLst,Stlth,Old,llU2R,llSOLARIS
41.084031 03/29/1999 08:29:27 00:00:43 srv spt dpt 209.154.098.104 172.016.112.050 1 ps
out,,rem,succ,aDmp,oDmp,iDmp,BSM,SysLg,FSLst,Stlth,Old,llU2R,llSOLARIS
41.084031 03/29/1999 08:40:14 00:24:26 srv spt dpt 209.154.098.104 172.016.112.050 1 ps
out,,rem,succ,aDmp,oDmp,iDmp,BSM,SysLg,FSLst,Stlth,Old,llU2R,llSOLARIS
The first 4 lines of the condensed detection truth listfile for Monday of week 4, the first week of evaluation test data. These entries summarize activity related to attack #41.084031, a "ps" User-to-root attack against the Solaris victim. Entries are very long and are shown here line-wrapped on to two lines.

The fields and keywords used in both the extensive and condensed versions of the detection truth files are the same. Table 6-4 shows the four condensed detection truth list entries that represent the four non-overlapping session of a "ps" attack carried out against pascal.eyrie.af.mil, 172.16.112.50. "Ps" is a U2R type attack on Solaris that in this instance was run in a stealthy fashion. The keywords "llU2R," "llSOLARIS," and "Stlth" indicate this information. Each line is very long and is shown here wrapped around to a second line. The format of entries is fixed and each field is a single string, white-space delimited before the hash-mark and comma delimited after the hash-mark. The meanings of the fields, in order, are as follows:

- Fields most important in the scoring procedure (given in the listfile entry before the #-sign):
  - **IDnum** is the ID number of this attack instance.
  - **Date** gives the date of this summarized session of this attack.
  - **StartTime** gives the start time of this summarized session.
  - **Duration** gives the duration of this summarized session.
  - **Srv, spt, dpt** are the service, source port, and destination port of the network session. Since the example given is of a condensed detection truth list, the wild-card replacements srv, spt, and dpt are filled in lieu of actual values.
  - **Source** gives the source IP address of this summarized session – usually this is the attacker, but might also be the victim if the session/packet originated at the victim.

- **Destination** gives the destination IP address of this summarized session - usually this is the victim of the attack, but could be the attacker if the session/packet originated at the victim.

- **Score** indicates the likelihood that the intrusion detection system thinks that the detection matches a true attack, in a list of putative detections. In a detection truth listfile score is always set to one.

- **Attackname** is the common name of this attack.

- These following fields are attribute keywords that indicate categories into which the attack instance is placed. They are presented in a comma separated list after the hash-mark. These fields can take on only certain discrete values depending on the field, which allows for easy regular-expression-based searching of a long list:

  - **Insider** is either "in" if the attack source is an Eyrie AFB host or "out" if the attack source is external to the base.

  - **Manual** is either "man" if the attack, or a major component thereof, was carried out manually or "auto" if the attack was scripted.

  - **Console** is either "cons" if the attack was carried out from the console of a machine or "rem" if the attack was carried out remotely, with respect to the victim.

  - **Success** is "succ" if the attack instance was successful or "not" if unsuccessful.

  - **aDump** is "aDmp" if a TCPdump file, or network trace, is available for that attack instance, otherwise "not."

  - **oDump** is "oDmp" if there is evidence of the attack in the outside sniffer dump file, otherwise "not."

  - **iDump** is "iDmp" if there is evidence of the attack in the inside sniffer dump file, otherwise "not."

  - **BSM** is "BSM" if there is evidence of the attack in the Solaris BSM log, otherwise "not."

  - **SysLogs** is "SysLg" if there is evidence of the attack in the system logs distributed to evaluation participants, otherwise "not." In this case, Windows NT audit logs were grouped as "system logs." In all other discussions, NT logs are referred to as WinNT audit logs.

- **FSListing** is "FSLst" if there is evidence of the attack in the file system information distributed to evaluation participants, otherwise "not."

- **Stealthy** is "Stlth" if the attack is considered Stealthy, "Clr" if considered clear.

- **New** is "New" if the attack was considered to be New in the 1999 eval, "Old" if the attack is considered old.

- **Category** gives the category, with respect to the goal of the attack, pre-pended with "ll".

- **OS** gives the platform/Operating System of the victim host of this attack instance, with ll pre-pended to the name.

6.7.2 Identification Truth List

Some evaluation participants optionally provided identification information for each detection entry submitted. This was used to evaluate the ability of the intrusion detection system to provide the system administrator with different types of information for each attack instance. The truth against which identification entries are scored is the Identification Truth file, also know as "master-identifications." In this file, each attack instance is represented by a multiple line entry. Entries in the truth file take the same format as that required for submission to the evaluation. Three example entries are given in Table 6-6. The identification truth entry provides much more information about each attack than is provided in the detection truth listfile entries. Creating this truth file, and verifying that entries were correct was a rather time-consuming procedure. Fields of the identification list entry are:

1. **ID** is the ID number of the attack instance and allows correlation between the identification entry and the corresponding detections list entry.
2. **Date**: This is the date the attack begins, given in the format MM/DD/YYYY.
3. **Name**: The common name of the attack. This information was only scored for attacks that are considered "Old" in the 1999 Evaluation. A list of attack names was provided for these attacks.
4. **Category**: The five categories with respect to the attack goal (DoS, PROBE, U2R, R2L, Data) will be used to categorize attacks, allowing some attacks to be in more than one category when necessary.
5. **Start**: The start time, in the format "HH:MM:SS" is the time at which activity for that attack instance begins. This start time need not be identical to the start time provided in the corresponding detection list entry.
6. **Duration**, in the format "HH:MM:SS," is the amount of time between the attack start and end times. If the attack has several stages (such as a setup, break-in, and actions), then the start time should be when the beginning of the setup occurs and the duration would encompass the full time period until all actions of the attacker are complete. The hour field can be larger than 24 if the attack extends over multiple days.

7. **Attacker** machine(s) are the sources of the attack or the attacking host(s). Either IP addresses or fully qualified machine names were accepted. Multiple sources are comma separated, and the notation x.y.z.(1-100) refers to all 100 machines, " x.y.z.1, x.y.z.2, ..., x.y.z.100".

8. **Victim** machine(s) are the victim(s) of the attack instance. Format is similar to that for source machines.

9. **Username** lists any username(s) that are made use of by the attacker during the attack instance to login to a particular host computer.

10. Destination **Ports** and Number of Connections to each: This is a list of all well-known ports that are made use of during an attack and any ports that are used as the destination of a TCP connection, UDP packet, or ICMP packet. The exception is for FTP-data connections where the well-known port number (20), on the machine where the port is used is given. The list of ports is classified according to weather they are ports on the attacking machine or the victim machine. For example, a telnet from host "Attacker", port 12345, to host "Victim", port 23 (Attacker:12345 -> Victim:23), results in port 23 on the "Victim". A range of ports is given with a dash, for example, 1-100 refers to ports 1,2,3,...,100. UDP connections are labeled as port#/u. ICMP connections will be labeled as "i". The number of times a port or range of ports is used during the attack is given with a numeral placed in curly braces after the port or port-range. For example: (1-100){5} indicates that ports 1, 2, 3, ... , 100 were each used 5 times, for a total of 500 connections. The list is comma separated.

11. The **Comments** field is optional and will not be made use of to score the identification part of the evaluation, but was used to provide addition information like the original alert as produced by the intrusion detection system.

To take an identification example from Table 6-6, the first entry in that table describes a U2R attack in which the attacker has obtained a user login to the victim. The attacker logs in, x-displays netscape to her machine and downloads the attack scripts to the victim. Then, over two more telnet sessions, the attacker unpacks the exploit, runs it, and cleans up.

6.8 ATTACKS

Fifty-eight types of attacks were run in the 1999 evaluation. Table 6-5 illustrates the categories in which these attack are placed for the 1999 evaluation and shows how the attack instances run in the evaluation fall into those categories. Attacks in italics are those that are considered "new" in the 1999 Off-Line Evaluation, while those in regular font are "old." Detailed descriptions of each of these attacks are given in Appendix A.

TABLE 6-5

ATTACKS RUN IN THE 1999 EVALUATION AND THE ATTACK CATEGORIZATION ACCORDING TO GOAL OF ATTACK AND VICTIM PLATFORM

	DoS (65)	Probe (37)	R2L (56)	U2R (37)	Data (13)
Solaris	Neptune pod Processtable *Selfping* Smurf Syslogd *Tcpreset* warezclient	Portsweep *Queso*	Dict Ftpwrite Guest Httptunnel Xlock Xsnoop	Eject Fdformat Ftbconfig Ps	*secret*
WinNT	*Arppoison* *Crashiis* *Dosnuke* *Tcpreset*	*Ntinfoscan* portsweep	*Dict* *Framespoofer* *Netbus* *Anypw* *ppmacro*	*Casesen* *Netcat* *Sechole* *Yaga*	*ntfsdos*
SunOS	*Arppoison* Land Mailbomb Neptune Pod processtable	Portsweep *queso*	Dict Xsnoop	Loadmodule	
Linux	Apache2 Arppoison Back Mailbomb Neptune Pod Processtable Smurf *Tcpreset* Teardrop udpstorm	*Ls_domain* Mscan *Queso* satan	Dict Imap Named *Ncftp* Phf Sendmail *Sshtrojan* Xlock Xsnoop	Perl Sqlattack Xterm	*secret*
Cisco	Neptune	Portsweep *queso*	Snmpget		
All O/S		*Illegalsniffer* Ipsweep portsweep			

TABLE 6-6

THE FIRST OF THREE ENTRIES IN THE 1999 EVALUATION IDENTIFICATION TRUTH FILE

```
ID: 41.084031
Date: 03/29/1999
Name: ps
Category: u2r
Start_Time: 08:18:35
Duration: 00:46:05
Attacker: 209.154.098.104
Victim: 172.016.112.050
Username: haraldl
Ports:
        At_Attacker: 80{1}, 6000{2}
        At_Victim: 23{3}

ID: 41.084818
Date: 03/29/1999
Name: sendmail
Category: r2l
Start_Time: 08:48:12
Duration: 00:00:02
Attacker: 202.049.244.010
Victim: 172.016.114.050
Username: n/a
Ports:
        At_Attacker: 113{1}
        At_Victim: 25{1}

ID: 41.090000
Date: 03/29/1999
Name: New_Attack_1999 (ntfsdos)
Category: r2l
Start_Time: 09:08:00
Duration: 00:08:00
Attacker: 172.16.112.100
Victim: 172.16.112.100
Username: n/a
Ports:
        At_Attacker:
        At_Victim:
```

6.9 SCHEDULE OF ATTACKS

This section lists all attack instances scheduled and run in two weeks of testing data produced for the 1999 Off-Line Evaluation. Tables 6-7 through 6-11 list all attack instances in the 1999 test data. Each row represents one attack instance. More detailed information about each attack instance is available at http://ideval.ll.mit.edu as noted in section 1.2. For each, the following information is given:

- **IDnum** – The instance ID number. The format is wd.hhmmss, so 41.084031 would indicate Week 4, Day 1(Monday), 08:40:31. This is roughly when the attack was scheduled to begin, however in the course of simulation, the actual time (not date) will vary slightly.
- **Name** – The common name of this attack.
- **Stealthy/Clear** – Was the attack instance considred Stealthy or Clear?
- **New/Old** – Was this attack considered new or old with regard to the 1999 evaluation?
- **Category** – The major category that classifies this attack (preprended with 11).
- **VictimOS** – The operating system or platform that fell victim in this instance.

TABLE 6-7

ATTACKS RUN IN SIMULATION WEEK 4, DAYS 1, 2, AND 3

ID Number	Name	Stlth/Clr	New/Old	Category	Victim O
41.084031	ps	Stlth	Old	IIU2R	IISOLARIS
41.084818	sendmail	Clr	Old	IIR2L	IILINUX
41.090000	ntfsdos	Clr	New	IIDATA-IIU2R-	IINT
41.091531	portsweep	Stlth	Old	IIPROBE	IILINUX
41.093708	sshtrojan	Clr	New	IIR2L	IILINUX
41.111531	portsweep	Stlth	Old	IIPROBE	IICISCO
41.112127	xsnoop	Clr	Old	IIR2L	IILINUX
41.114554	snmpget	Clr	Old	IIR2L	IICISCO
41.114703	dict	Clr	Old	IIR2L	IISUNOS
41.122222	portsweep	Stlth	Old	IIPROBE	IINT
41.133333	dict	Clr	Old	IIR2L	IISOLARIS
41.135830	ftpwrite	Clr	Old	IIR2L	IISOLARIS
41.155048	yaga	Clr	New	IIU2R	IINT
41.161308	crashiis	Clr	Old	IIDOS	IINT
41.162715	portsweep	Stlth	Old	IIPROBE	IILINUX
41.182453	secret	Clr	New	IIDATA	IISOLARIS
41.213446	smurf	Clr	Old	IIDOS	IISOLARIS
42.090909	httptunnel	Clr	Old	IIR2L	IISOLARIS
42.094131	phf	Clr	Old	IIR2L	IILINUX
42.104107	loadmodule	Stlth	Old	IIDATA-IIU2R-	IISUNOS
42.112913	ps	Clr	Old	IIDATA-IIU2R-	IISOLARIS
42.120000	ntfsdos	Clr	New	IIDATA-IIU2R-	IINT
42.122248	secret	Clr	New	IIDATA	IILINUX
42.135452	sqlattack	Clr	Old	IIDATA-IIU2R-	IILINUX
42.143228	sechole	Clr	New	IIU2R	IINT
42.145441	land	Clr	Old	IIDOS	IISUNOS
42.155148	mailbomb	Clr	Old	IIDOS	IILINUX
42.174944	processtable	Clr	Old	IIDOS	IILINUX
42.210410	crashiis	Clr	Old	IIDOS	IINT
43.080401	satan	Clr	Old	IIPROBE	IILINUX
43.084000	netcat	Clr	New	IIR2L	IINT
43.093814	imap	Clr	Old	IIR2L	IILINUX
43.100000	ppmacro	Clr	New	IIDATA-IIR2L-	IINT
43.101313	processtable	Clr	Old	IIDOS	IISUNOS
43.103931	fdformat	Stlth	Old	IIU2R	IISOLARIS
43.110000	netcat	Clr	New	IIR2L	IINT
43.111111	warezmaster	Clr	Old	IIDOS	IISOLARIS
43.113032	arppoison	Clr	New	IIDOS	IISUNOS
43.114500	ncftp	Stlth	New	IIR2L	IILINUX
43.122854	secret	Clr	New	IIDATA	IISOLARIS
43.125900	named	Clr	Old	IIR2L	IILINUX
43.134223	dict	Clr	Old	IIR2L	IISUNOS
43.144547	smurf	Clr	Old	IIDOS	IINT
43.155357	guest	Clr	Old	IIR2L	IISOLARIS
43.164334	portsweep	Stlth	Old	IIPROBE	IINT
43.165422	mailbomb	Clr	Old	IIDOS	IISOLARIS
43.175811	dict	Clr	Old	IIR2L	IISOLARIS
43.191217	snmpget	Clr	Old	IIR2L	IICISCO

TABLE 6-8

ATTACKS RUN IN SIMULATION WEEK 4, DAYS 4 AND 5

ID Number	Name	Stlth/Clr	New/Old	Category	Victim O/S
44.110000	dosnuke	Clr	New	IIDOS	IINT
44.114500	ncftp	Stlth	New	IIR2L	IILINUX
44.120500	ppmacro	Clr	New	IIDATA-IIR2L-	IINT
44.124700	guest	Clr	Old	IIR2L	IISOLARIS
44.130700	xlock	Clr	Old	IIR2L	IILINUX
44.131529	dict	Clr	Old	IIR2L	IILINUX
44.161242	phf	Clr	Old	IIR2L	IILINUX
44.164944	processtable	Clr	New	IIDOS	IISOLARIS
44.183234	mailbomb	Clr	Old	IIDOS	IILINUX
44.201454	sqlattack	Clr	Old	IIDATA-IIU2R-	IILINUX
45.084547	smurf	Clr	Old	IIDOS	IISOLARIS
45.090000	arppoison	Clr	New	IIDOS	IILINUX
45.095541	sshtrojan	Clr	New	IIR2L	IILINUX
45.100334	ipsweep	Stlth	Old	IIPROBE	IIALL
45.103937	xlock	Clr	Old	IIR2L	IILINUX
45.105138	named	Clr	Old	IIR2L	IILINUX
45.111010	portsweep	Stlth	Old	IIPROBE	IISUNOS
45.114500	ncftp	Stlth	New	IIR2L	IILINUX
45.114900	netbus	Clr	New	IIR2L	IINT
45.123234	mailbomb	Clr	Old	IIDOS	IISUNOS
45.130542	named	Clr	Old	IIR2L	IILINUX
45.140000	ipsweep	Stlth	Old	IIPROBE	IIALL
45.162148	loadmodule	Clr	Old	IIU2R	IISUNOS
45.165009	sechole	Clr	New	IIU2R	IINT
45.181011	portsweep	Clr	Old	IIPROBE	IISUNOS
45.192523	ipsweep	Clr	Old	IIPROBE	IIALL
45.203400	secret	Clr	New	IIDATA	IISOLARIS

TABLE 6-9

ATTACKS RUN IN SIMULATION WEEK 5, DAYS 1 AND 2

ID Number	Name	Stlth/Clr	New/Old	Category	Victim O/S
51.083800	pod	Clr	Old	IIDOS	IISOLARIS
51.084334	portsweep	Stlth	Old	IIPROBE	IICISCO
51.085000	pod	Clr	Old	IIDOS	IILINUX
51.085947	warezclient	Clr	Old	IIDOS	IISOLARIS
51.093123	smurf	Clr	Old	IIDOS	IISOLARIS
51.094334	portsweep	Clr	Old	IIPROBE	IISOLARIS
51.102700	apache2	Clr	Old	IIDOS	IILINUX
51.105811	dict	Clr	Old	IIR2L	IILINUX
51.111333	arppoison	Clr	New	IIDOS	IISUNOS
51.114500	dosnuke	Clr	New	IIDOS	IINT
51.120309	loadmodule	Stlth	Old	IIU2R	IISUNOS
51.121101	ffbconfig	Clr	Old	IIU2R	IISOLARIS
51.131803	smurf	Clr	Old	IIDOS	IILINUX
51.133019	arppoison	Clr	New	IIDOS	IINT
51.140100	apache2	Clr	Old	IIDOS	IILINUX
51.142100	pod	Clr	Old	IIDOS	IILINUX
51.144601	imap	Clr	Old	IIR2L	IILINUX
51.150000	ipsweep	Stlth	Old	IIPROBE	IIALL
51.163200	dict	Clr	Old	IIR2L	IILINUX
51.171917	syslogd	Clr	Old	IIDOS	IISOLARIS
51.180445	neptune	Clr	Old	IIDOS	IISOLARIS
51.183623	crashiis	Clr	Old	IIDOS	IINT
51.185613	ls	Clr	New	IIPROBE	IILINUX
51.194715	dosnuke	Clr	New	IIDOS	IINT
51.200037	udpstorm	Old	IIDOS	IIALL	1
51.201715	selfping	Clr	New	IIDOS	IISOLARIS
51.204631	ncftp	Stlth	New	IIR2L	IILINUX
52.081109	tcpreset	Clr	New	IIDOS	IILINUX
52.083236	teardrop	Clr	Old	IIDOS	IILINUX
52.085357	casesen	Clr	New	IIU2R	IINT
52.092200	xsnoop	Clr	Old	IIR2L	IISOLARIS
52.094514	selfping	Clr	New	IIDOS	IISOLARIS
52.100738	xterm	Clr	Old	IIU2R	IILINUX
52.101901	ftpwrite	Clr	Old	IIR2L	IISOLARIS
52.103409	back	Clr	Old	IIDOS	IILINUX
52.112045	ps	Stlth	Old	IIU2R	IISOLARIS
52.113855	neptune	Clr	Old	IIDOS	IILINUX
52.120600	httptunnel	Clr	Old	IIR2L	IISOLARIS
52.125501	eject	Stlth	Old	IIU2R	IISOLARIS
52.130655	pod	Clr	Old	IIDOS	IISUNOS
52.132827	yaga	Clr	New	IIU2R	IINT
52.135003	crashiis	Clr	Old	IIDOS	IINT
52.140207	ppmacro	Clr	New	IIR2L	IINT
52.141200	syslogd	Clr	Old	IIDOS	IISOLARIS
52.142452	perl	Clr	Old	IIU2R	IILINUX
52.162435	fdformat	Clr	Old	IIDATA-IIU2R-	IISOLARIS
52.165435	queso	Clr	New	IIPROBE	IISUNOS
52.181637	neptune	Clr	Old	IIDOS	IICISCO
52.205605	dosnuke	Clr	New	IIDOS	IINT
52.211313	ipsweep	Clr	Old	IIPROBE	IIALL
52.214522	ncftp	Stlth	New	IIR2L	IILINUX
52.050813	udpstorm	Clr	Old	IIDOS	IIALL

TABLE 6-10

ATTACKS RUN IN SIMULATION WEEK 5, DAYS 3 AND 4

ID Number	Name	Stlth/Clr	New/Old	Category	Victim O/S
53.045454	selfping	Clr	New	IIDOS	IISOLARIS0
53.084346	xlock	Clr	Old	IIR2L	IISOLARIS
53.085717	phf	Clr	Old	IIR2L	IILINUX
53.092039	tcpreset	Clr	New	IIDOS	IISOLARIS0
53.094800	netbus	Clr	New	IIR2L	IINT
53.102617	back	Clr	Old	IIDOS	IILINUX
53.110500	netcat	Clr	New	IIR2L	IINT
53.110516	queso	Stlth	New	IIPROBE	IILINUX
53.123735	portsweep	Stlth	Old	IIPROBE	IILINUX
53.133203	perl	Stlth	Old	IIU2R	IILINUX
53.134015	queso	Stlth	New	IIPROBE	IISOLARIS0
53.144800	snmpget	Clr	Old	IIR2L	IICISCO
53.150110	processtable	Clr	Old	IIDOS	IISUNOS
53.152648	back	Clr	Old	IIDOS	IILINUX
53.155432	ffbconfig	Stlth	Old	IIU2R	IISOLARIS
53.171350	apache2	Clr	Old	IIDOS	IILINUX
53.195130	portsweep	Clr	Old	IIPROBE	IISOLARIS
54.082003	ps	Stlth	Old	IIU2R	IISOLARIS
54.090101	phf	Clr	Old	IIR2L	IILINUX
54.091200	casesen	Clr	New	IIU2R	IINT
54.102102	ntfsdos	Clr	New	IIDATA	IINT
54.103459	portsweep	Stlth	Old	IIPROBE	IISOLARIS
54.110416	ntinfoscan	Clr	Old	IIPROBE	IINT
54.115000	yaga	Clr	New	IIU2R	IINT
54.115701	crashiis	Clr	Old	IIDOS	IINT
54.120600	httptunnel	Clr	Old	IIR2L	IISOLARIS0
54.125758	fdformat	Clr	Old	IIU2R	IISOLARIS
54.145832	satan	Clr	Old	IIPROBE	IILINUX
54.155338	teardrop	Clr	Old	IIDOS	IILINUX
54.160341	sechole	Clr	New	IIU2R	IINT
54.170132	resetscan	Stlth	New	IIPROBE	IIALL
54.171643	ipsweep	Clr	Old	IIPROBE	IIALL
54.175007	snmpget	Clr	Old	IIR2L	IICISCO
54.183002	ntinfoscan	Clr	Old	IIPROBE	IINT
54.190707	ls	Clr	New	IIPROBE	IILINUX
54.194108	warezclient	Clr	Old	IIDOS	IISOLARIS0
54.195951	mscan	Clr	Old	IIPROBE	IIALL
54.225131	arppoison	Clr	New	IIDOS	IINT

TABLE 6-11

ATTACKS RUN IN SIMULATION WEEK 5, DAY 5

ID Number	Name	Stlth/Clr	New/Old	Category	Victim O/S
55.080105	portsweep	Clr	Old	IIPROBE	IISUNOS
55.080500	xsnoop	Clr	Old	IIR2L	IISUNOS
55.081418	crashiis	Clr	Old	IIDOS	IINT
55.082500	illegalsniffer	Stlth	New	IIPROBE	IIALL
55.084452	back	Clr	Old	IIDOS	IILINUX
55.085500	illegalsniffer	Clr	New	IIPROBE	IIALL
55.085514	netcat	Clr	New	IIR2L	IINT
55.091529	xterm	Clr	Old	IIU2R	IILINUX
55.093137	portsweep	Stlth	Old	IIPROBE	IISOLARIS
55.100600	anypw	Clr	New	IIU2R	IINT
55.100830	guest	Clr	Old	IIR2L	IISOLARIS
55.102000	tcpreset	Clr	New	IIDOS	IINT
55.103001	perl	Clr	Old	IIU2R	IILINUX
55.110800	framespoofer	Clr	New	IIR2L	IINT
55.115202	portsweep	Stlth	Old	IIPROBE	IISUNOS
55.123412	sqlattack	Stlth	Old	IIU2R	IILINUX
55.124400	yaga	Clr	New	IIU2R	IINT
55.125112	crashiis	Clr	Old	IIDOS	IINT
55.125811	dict	Clr	Old	IIR2L	IINT
55.125830	crashiis	Clr	Old	IIDOS	IINT
55.140643	syslogd	Clr	Old	IIDOS	IISOLARIS
55.141732	eject	Clr	Old	IIU2R	IISOLARIS
55.163447	land	Clr	Old	IIDOS	IISUNOS
55.171917	syslogd	Clr	Old	IIDOS	IISOLARIS
55.172757	sendmail	Clr	Old	IIR2L	IILINUX
55.174733	xterm	Clr	Old	IIU2R	IILINUX
55.183012	neptune	Clr	Old	IIDOS	IISUNOS
55.184715	perl	Stlth	Old	IIU2R	IILINUX
55.185233	warezclient	Clr	Old	IIDOS	IISOLARIS
55.202002	queso	Stlth	New	IIPROBE	IICISCO
55.204925	casesen	Clr	New	IIU2R	IINT
55.200530	secret	Clr	New	IIDATA	IISOLARIS

6.10 MULTIPLE-SESSION ATTACK SCENARIOS

Although the focus of the evaluation was on detection of simple attacks components or exploits, some of these components are comprised of several steps carried out over multiple sessions. For the most part, these are remote-to-local attacks against the Windows NT victim and user-to-root attack instances. Often these exploits can be thought of as having a setup phase and a break-in phase. Multiple-session attack components such as these might be of use to developers of fusion and correlation intrusion detection technology. Several lower-level intrusion detection systems could be run on these data sets and the outputs might show that different systems, perhaps using different data sources, are able to detect different facets of these multiple phase attacks. The following two subsections describe how these multiple phase exploits were carried out in the 1999 evaluation.

6.10.1 U2R Attacks

Many of the U2R attacks, especially those considered to be stealthy, consisted of a setup phase in which the attacker downloads the attack script or program to the victim machine, and a break-in phase in which the attacker logs in, compiles the program runs the attack and cleans up. Figure 6-1 summarizes the five major actions carried out by the attacker as a part of the exploit and lists some specific implementation options for each.

Transporting the scripts and programs for the attack via some encoded means comprises the setup phase. Compiling attack programs, executing the break-in, effecting the desired malicious actions, and cleaning up comprise the break-in phase.

The arrows in Figure 6-1 show how attack ID#53.155432, an ffbconfig attack against the Solaris victim, was carried out. Initial state for this attack instance supposed that the attacker had already gained access (username and password) to the victim host. Setup consisted of mailing uuencoded tar-file to the compromised account on the victim host. The tar-file contained two pre-compiled ffbconfig exploits, one that sets "world-read and write" permissions on another user's home directory and one that resets these permissions, and a shell script which runs these two programs and performs the malicious action of deleting files from that user's directory. Break-in consisted of a login session during which the attacker uudecode's the tar-file, un-tars it, runs the shell script and then cleans up by deleting the tar-file, the script, and the ffbconfig exploit programs.

The individual steps of this exploit might be detected differently by different intrusion detection systems, a feature which might be useful for a fusion or correlation intrusion detection system. More detailed discussion of these multiple session U2R attacks against the Solaris host can be found in [18].

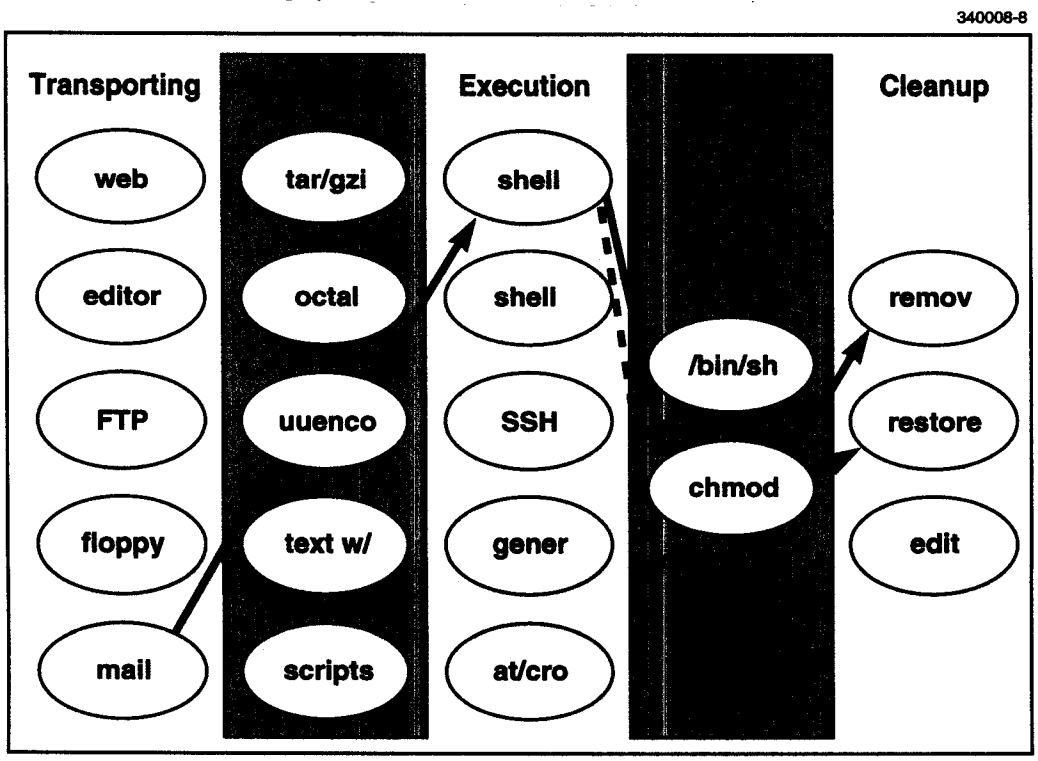


Figure 6-1. Many U2R attack components in the evaluation were carried out in several network or console sessions, with steps distributed across the sessions.

6.10.2 R2L Attacks vs. Windows NT

Many of the remote-to-local attack instances run against the Windows NT victim host were also multiple session attacks. Netbus, Netcat, and Ppmacro attacks, described in detail in Appendix A, are examples.

Often these instances of these attacks were composed of three phases, each containing multiple steps:

- In the first phase, the attacker sends a spoofed mail message, containing a Trojan program or script, to a legitimate user or administrator on the victim host.
- In the second phase, a user or administrator in the victim host reads the mail message and is tricked into executing the attached program or script. The script then performs the actions promised to the unsuspecting user and also carries some Trojan actions, with that user's privi-

leges, on behalf of the attacker. Often these actions involved setting up a backdoor or otherwise allowing the attacker access to information that was previously only available to legitimate users on that host.

- In the third phase the attacker either uses the backdoor or accesses the now accessible information, perhaps available via HTTP.

Similarly to the stealthy U2R attacks against Solaris, there are many individual steps to these exploits. Facets of these steps might be detected differently by different intrusion detection systems, a feature that could be useful for a fusion or correlation intruction detection system. More detailed discussion of these multiple session R2L attacks against the Windows NT host can be found in [17].

7. BACKGROUND TRAFFIC AND GENERATION

7.1 OVERVIEW

Background traffic was generated in the testbed for a variety of reasons. This traffic made it possible to measure baseline false alarm rates of evaluated intrusion detection systems and to deter the development of limited non-robust intrusion detection systems that simply trigger when a particular traffic type occurs. It also led to reasonably high data rates and a fairly rich set of traffic types that exercise traffic handling and analysis capabilities of network analysis and intrusion detection tools tested with evaluation corpora. Finally, the synthesized nature of the traffic allows widespread and relatively unrestricted access to the evaluation corpora. False alarm rates measured with the evaluation corpus may not represent operational false alarm rates at any location. Network traffic varies widely with location and time. This implies that it may be difficult to predict the false alarm rates at operational sites from false alarm rates measured during any evaluation because traffic characteristics, including details that affect false alarm rates, are likely to differ widely from those used in the evaluation. The approach taken in the testbed was to generate realistic traffic that is roughly similar to traffic measured on one Air Force base in early 1998. In addition, details of this traffic (e.g., the frequency of occurrence of words in mail, telnet sessions, and FTP file transfers) were designed to produce false alarm rates similar to operational rates obtained in 1998 using the Air Force ASIM intrusion detection system. False alarm rates measured with this traffic can be used to benchmark or compare intrusion detection systems on reference evaluation background traffic corpora. They may not, however, be representative of false alarm rates on operational data. Supplementary measurements using restricted-access data are necessary to determine operational false alarm rates.

A large amount of web, telnet, and mail traffic was generated between the inside PCs and workstations and the outside workstations and web sites. In addition, there are many user automata of various types (e.g., secretaries, programmers, managers) on outside workstations who perform work using telnet and other services on the three inside victim machines and the other inside workstations. The three traffic generation machines, described in section 5, contain operating system kernel modifications similar to those used in [10] in conjunction with custom software web, mail, telnet, and other servers to allow a small number of actual hosts to appear as if they were thousands of hosts with different IP addresses. The contents of network traffic such as SMTP, HTTP, and FTP file transfers are either statistically similar to live traffic, or sampled from public-domain sources. For example, some email message contents are created using statistical bi-grams frequencies to preserve word and two-word sequence statistics from a sampling of roughly 10,000 actual email messages. These messages were sent to and from computer professionals and are filtered using a 40,000 word dictionary to remove names and other private information. Other email messages are actual messages from a variety of public-domain mail list servers. Similar approaches were used to produce content for FTP file transfers.

Section 7.2 discusses how real-world traffic to and from Air Force bases was analyzed to provide a model to which synthesized traffic would be generated. Section 7.4 describes the algorithms used to synthesize background traffic session. Section 7.5 presents a brief analysis of the 1999 background traffic as generated during the evaluation.

7.2 TRAFFIC STATISTICS DERIVED FROM BACKGROUND TRAFFIC

Composition of background traffic for the 1999 and 1998 evaluations was based on statistics that have been collected from a number of sources. The following subsections describe the types of information we used to generate traffic for the evaluation.

7.2.1 AFIWC data

The Air Force Information Warfare Center has installed and maintained ASIM intrusion detection systems at most Air Force bases around the world. These detectors analyze and log many of the network services from the network traffic seen at the base's uplink to the Internet. In 1997, AFWIC provided four months of ASIM connection information and transcripts from more than 50 Air Force Bases. This ASIM data contained details of more than 375,000 session transcripts and more limited information on 34.6 million connections. A wide range of services were sampled (e.g. SMTP, FTP, telnet, rlogin, echo, finger, shell, rlogin, domain, TFTP, ...). Figure 7-1 summarizes the relative number of connections per day for various services seen going to and from these 50 Air Force bases. Web (HTTP) traffic was not monitored by ASIM at the time and is not shown in this chart. Transcript data was a biased sample, because it included primarily sessions judged suspicious by ASIM. From the connection data we created statistics of (1) traffic per base, (2) traffic per TCP services, and (3) TCP/IP domain source/destinations connections. From the transcript data we extracted histograms of command usage in remote sessions. We then developed custom PERL scripts to determine what happened in the interactive telnet, rlogin, shell sessions, to extract the host operating system types, to extract banners, to extract passwords and check for dictionary attacks, and to find probes and sweeps and detect when users illegally become root. See [19] for information on software developed to extract passwords and banners and an early neural network to detect attacks.

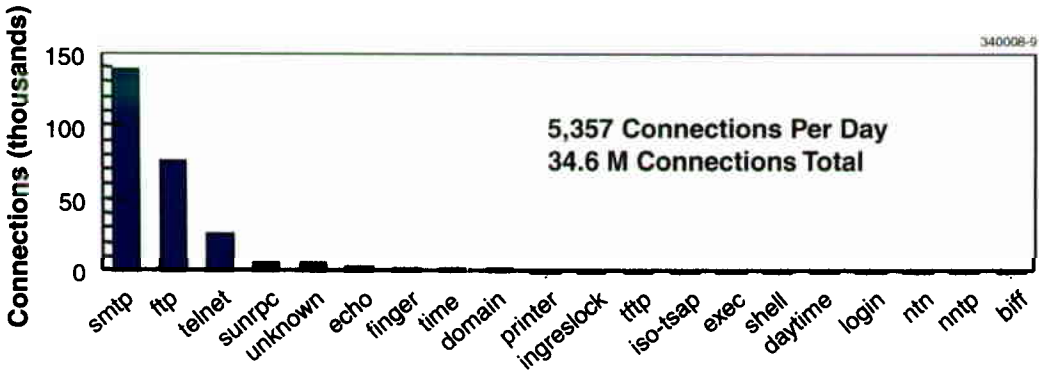


Figure 7-1. Average number of connections per day for various services summed across 50 Air Force bases. Figures represent only those services that were monitored by the ASIM devices on these bases.

AFWIC data also provided histograms of command usage during telnet sessions, from which we were able to more accurately model command usage in simulated telnets.

These data motivated the types of services we included in the evaluations, the content of FTP, telnet, and mail sessions, and the style of many clear user-to-root and probe attacks.

7.2.2 Hanscom Sniffing Data

A few weeks of continuous sniffing data was collected from a few locations at Hanscom Air Force Base. This data was treated as classified and not distributed. Initial histograms were made of source/ destinations, services, and TCP/IP domains. Telnet sessions and unusual non-standard services were analyzed by hand to determine their purpose. Detailed analyses of email messages were also performed to determine the number of email messages with attachments, the sizes and types of attachments, and the sizes of FTP transfers. Web access statistics were also provided for analysis. We also analyzed the traffic in five-minute intervals over a few days to determine how traffic varies during a day.

Information from this data used to drive the simulation included the following:

1. The relative percentage of the traffic per service type.
2. The overall traffic level.
3. The list of web sites visited.
4. The purpose of telnet sessions and their content.
5. The variation of traffic during a day in five-minute intervals.
6. Special use of services such as the time service to synchronize clocks using a remote site.
7. Examples of remote administration of on-site hosts.
8. Frequency and sizes of email message attachments.

All of the raw data used to create these statistics was classified.

7.2.3 Public Domain Files

Many mail messages and files transferred using FTP or attached to email messages came from public domain mailing lists. In addition, many programs created by program automata were public domain or internal C and PERL programs. In addition, two-gram word sequence generators were used to generate files and mail messages. Two-gram word sequence statistics were trained using 10,000 actual mail messages from researchers on intrusion detection and speech recognition and the content of all UNIX man pages. These data were first sanitized to remove proper names and locations.

7.2.4 Other sniffer analyses

Sniffing data from Lincoln Laboratory and other sites were also examined to determine the traffic types, variation of traffic with time, and overall load. Some of these analyses motivated the addition of SNMP traffic to the simulation. We also obtained a list of web sites visited by folks from Lincoln Laboratory and combined it with the Hanscom web site list to generate a list that covered more sites and could not be used to reverse-engineer the web-site statistics to determine which sites were visited by Hanscom personnel.

7.3 BACKGROUND TRAFFIC SESSION PLAYBACK

Background traffic was actually automated, or played out, on the testbed with special *Expect* scripts. This script-based technique for automating background traffic sessions in the testbed environment is described further in section 5.3.7 and Appendix B. Background traffic session synthesis is described in detail, for each traffic type, in the next section.

7.4 SYNTHESIS OF BACKGROUND TRAFFIC SESSIONS

This section presents the algorithms used to synthesize automated background traffic sessions for each day of 1999 evaluation data. For each session, an EXS script specifies all facets of the session including the source IP address of any network traffic, the user who will be simulated, and the commands or network actions that will be carried out. PERL scripts, run prior to the simulation day on a separate machine, synthesize all automated background traffic sessions and write the EXS scripts to automate these sessions. EXS scripts and their format is discussed in Appendix B.

Traffic is generated on the network testbed using a variety of protocols or network services and is created in terms of user sessions. These sessions may result in the use of more than one protocol during the session. Different types of sessions simulate users performing different types of activities, as shown in Table 7-1. The first and second columns refer to the network protocol and the type of session that uses the protocol. The term *internal* refers to users or machines that reside on the simulated Eyrie Air Force Base, and *external* refers to users or machines that reside on the simulated Internet. Sections that describe the algorithms used to synthesize each type of session are listed in the last column of the table.

TABLE 7-1

TYPES OF BACKGROUND TRAFFIC SYNTHESIZED FOR THE 1999 EVALUATION

Protocol	Session Type	Summary	Details
FTP	FTP	Internal and external users use ftp to get and put files on one of three Air Force base ftp servers	Section 7.4.1
HTTP	Lynx	Internal/External users surf Air Force base servers with Lynx	Section 7.4.2
	Browser	Internal/External users surf Air Force base servers with PERL-based browser	Section 7.4.3
	Internet Browser	Internal users surf the Internet	Section 7.4.4
	AutoBrowser	User on Windows NT server surfs the Internet using Netscape	[13]
Telnet	Telnet	Telnets to/from Air Force base represent users performing daily, work-related tasks	Section 7.4.9
	Mailread	Users telnet to internal and external hosts to check their email	Section 7.4.7
POP	POP	Internal users use POP to get their email from External mail servers	Section 7.4.6
SMTP	Sendmail	Lots of email generated to and from Air Force base users	Section 7.4.8
	Mailread	Users telnet to internal and external hosts to check their email using the UNIX command-line "mail" program	Section 7.4.7
SQL	SQL	Internal and external users telnet to an SQL server and query the database	Section 7.4.4
ICMP	Sendmail, Telnet	Users have a chance to attempt to ping a host to ensure that it's up before sending email there. In addition, users performing telnet sessions have a chance to use ping.	Sections 7.4.8 and 7.4.9
Finger	Sendmail	Users have a chance to attempt to finger a user to ensure that the username exists before sending email	Section 7.4.8
SSH	Telnet	Certain users have a high probability of using the SSH protocol instead of telnet when generating these login sessions.	Section 7.4.9
IRC	IRC	Throughout the day, users participate in a IRC chat room, external to the Air Force base.	Section 7.4.10

The general algorithm of this synthesis process is very similar for the most traffic types. Session arrivals are a Poisson distribution with time-varying rate. Each session is a single-user session where the user could be on one of many machines on the inside or outside networks. For example, an http session models a single user surfing the web for a period of time. Collectively, all sessions of all services model the traffic seen on this Air Force base. For all services except Telnet, the rate is given for each 15-minute interval throughout the day and specified in a file for each service. Session arrival rates for six types of traffic are plotted in Figure 7-2. The X-axis is the hour of the day, while the Y-axis is the arrival rate for each interval, or mean number of session arrivals per 15-minute interval. Most sessions occur between 8:00 a.m. and 6:00 p.m. as was observed in the real world traffic being modeled. These are session arrival rates, not connection arrival rates. A session consists of a user using the service, thus a single HTTP session will result in multiple HTTP connections as the user web-surfs multiple web pages.

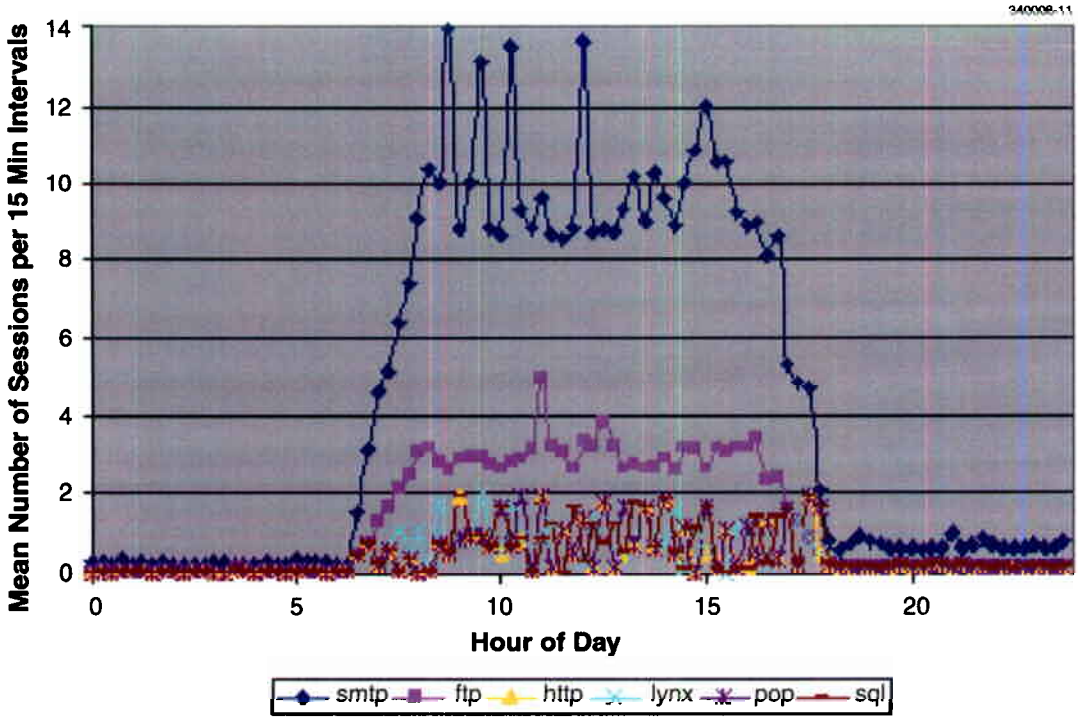


Figure 7-2. A plot of the number of sessions of various traffic types that are to arrive, on average, for each 15-minute interval throughout the simulation day.

The algorithm that drives the session arrival times for the synthesis of most traffic types is shown in Figure 7-3. This algorithm specifies the starting times of HTTP, Lynx, POP, FTP and SQL sessions throughout each simulation day. Specifics of session synthesis are given in the following sections, which describe each individual service. Sessions are distributed throughout the simulation day based on traffic arrival rates that are specified for each 15-minute interval throughout the day. These arrival rates vary depending on the time of day and are specified in flat text files read as part of the initialization process for traffic synthesis.

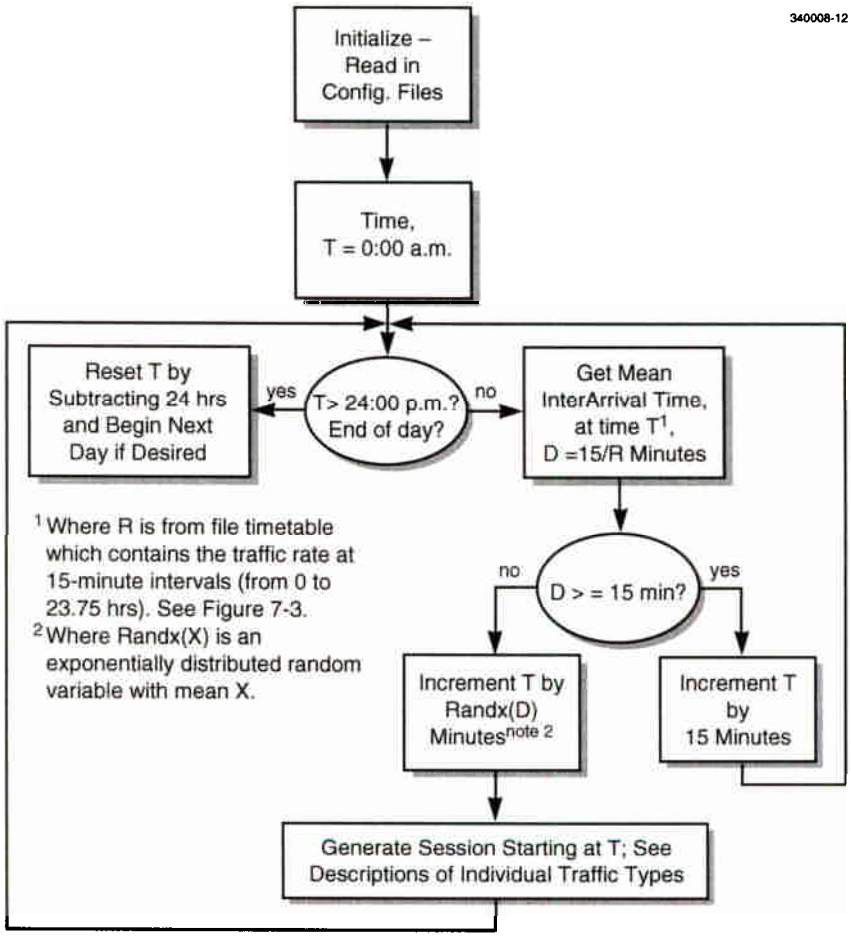


Figure 7-3. General algorithm for synthesizing several traffic types, including HTTP, Lynx, SQL, POP, and FTP.

The general algorithm for synthesizing background traffic sessions begins with initialization of the generator by reading configuration files into memory. Each type of traffic requires different data files, but in general these will include the arrival rates file, lists of possible source and destination hosts for the session, lists of users who might carry out the session, and lists of commands that might be run or files that might be transferred. To begin the synthesis loop, time, T, is set to 00:00 a.m., midnight, the beginning of the simulation day.

Then the process enters a loop, shown in Figure 7-3, and steps through the simulation day generating session start times and synthesizing sessions by calling the traffic-specific synthesis routine. Current time, T, is incremented with each pass through the loop. In intervals where the traffic rate is greater than 1 (one or more sessions per 15 min.), the current time, T, is incremented by an exponentially distributed amount based on the session arrival rate in the current 15-minute interval, and a session is generated. The increment is determined by randx(Mean_Interval), where randx(M) is an exponentially distributed random variable with mean M, and Mean_Interval is 15 minutes divided by the number of expected arrivals in that interval. In this way, session interarrival times are distributed, within each interval, based on the expected number of sessions in that interval. This technique works best when the expected number of sessions per interval is greater than one. When the number of arrivals drops below one, the expected interarrival time begins to have a reasonable probability of exceeding the interval duration, at which point it would be possible to increment time and skip one or more intervals. In intervals where the traffic rate is less than 1, the current time is incremented by 15 minutes, into the next interval, to avoid the probability of skipping the next interval. Analyses of the resulting background traffic are presented in Section 7.5. In the future, a better way to distribute sessions based on these 15-minute intervals would be to consider each interval separately; use an exponential random variable with mean being the number arrivals expected in that interval, and then place those arrivals in the interval according to a uniform random variable. However, with the traffic rates desired in the evaluation, the technique that was used worked well.

Each session generated is checked to ensure that it does not conflict with the attack schedule. If the source IP address of the background session matches the source of an attack and the start time of the session falls within the duration of or within 15 minutes before or 1 hour after the attack session, then they are considered to conflict. If there is a conflict, the session is cancelled. In the future, particularly if greater rates of traffic are generated, it might be required to more carefully intermingle attacks and background traffic sessions. For the rates considered in the 1999 evaluation, however, the technique of canceling conflicting sessions seems to have been acceptable, as each session source IP address was not often requested for both attack and background traffic, and few sessions had to be canceled.

7.4.1 FTP

FTP session model internal and external users using FTP to transfer files to and from one of the three Air Force base FTP servers. The start times of FTP traffic sessions is according to a Poisson distribution with time varying rate. Rates are based on time of day and are given by a timetable file. The algorithm shown in Figure 7-3 determines FTP session arrivals. As seen from Figure 7-2, most of the traffic occurs between 8:00 a.m. and 5:00 p.m. This is not surprising since these are the normal working hours, and there would be low traffic at an Air Force base outside of these hours. The general traffic generation algorithm, shown in Figure 7-3, generates start times for FTP sessions in each simulation day.

The algorithm used to synthesize each individual FTP session is shown in Figure 7-4. These are anonymous ftp sessions which originate from the inside network or the outside network and are destined to one of the three anonymous FTP servers on the inside network. There is a chance of retrieving either all files in a given directory or just retrieving a single file. For each FTP session, the user is chosen from a list of 689 possible users, the FTP directory index is chosen from a list of 14, and FTP page index is chosen from a list of 1241. All are chosen with a uniform distribution. FTP sessions can be generated from either the inside or from the outside. FTP sessions can either be sourced directly from the local machine or by telnetting into another host. The telnet option is only available when the session originates from the inside network. Telnet requires users to log in with a valid username and password. There is a 0.1 probability that one bad log in will occur and a 0.01 probability that two bad logins will occur (with equal probability of miss typing the username or password). There is a 0.001 probability that a failed login will occur. There is a 2% chance that the anonymous FTP login will fail (all FTP logins are anonymous). If the directory name contains "/src/c", there is a 50% chance that the user will FTP all files in the directory, or the user will only FTP one file. If the user gets one file, there is a 0.1% chance that he will mistype the filename.

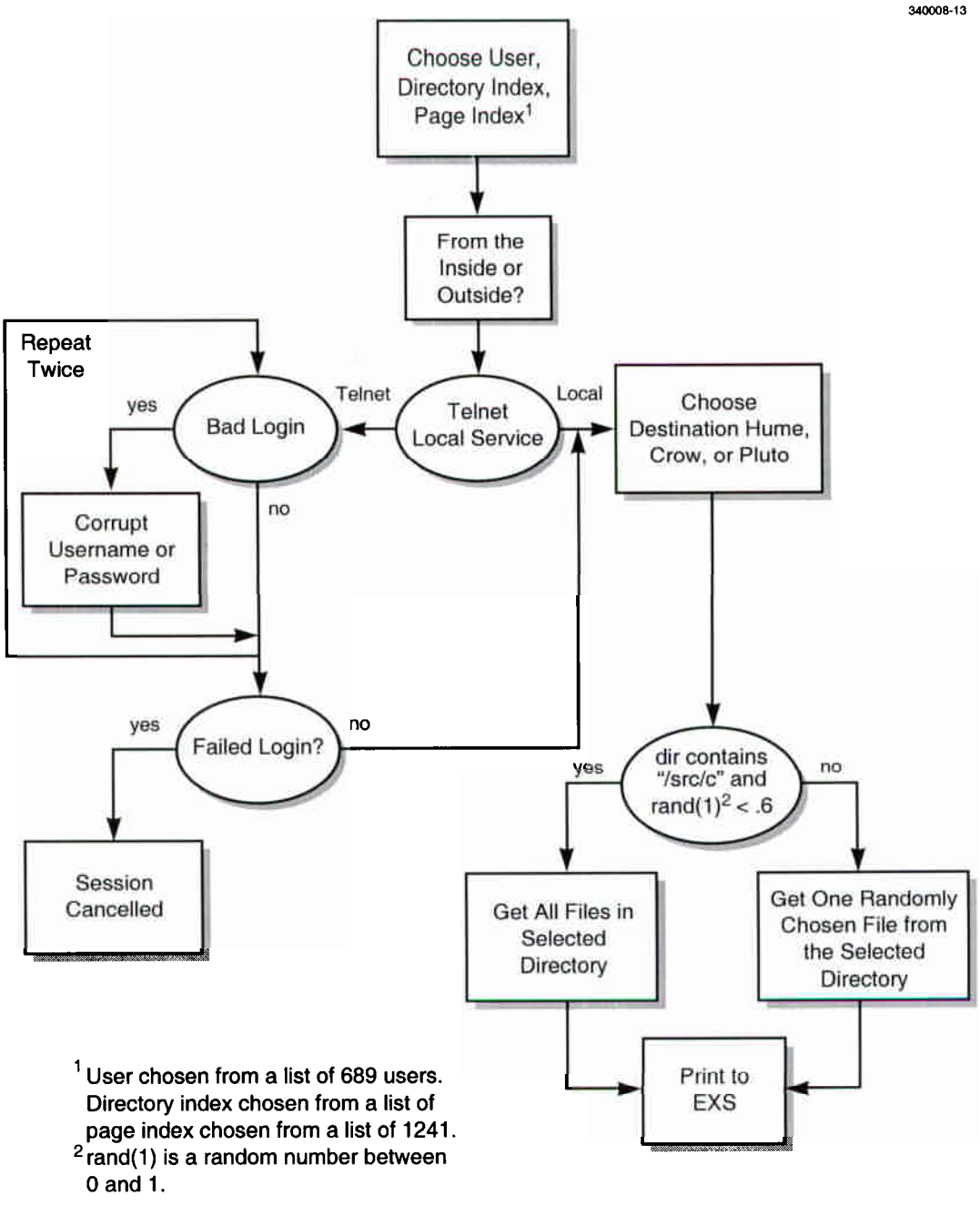
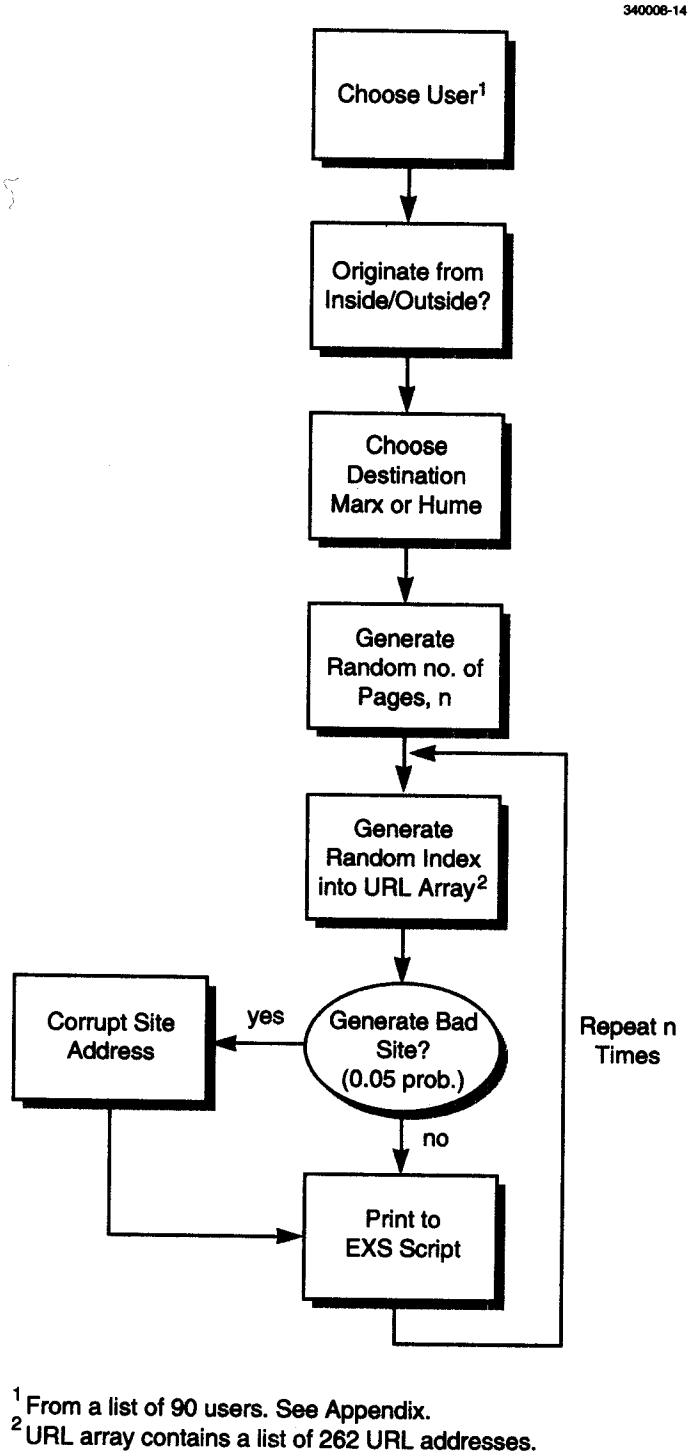


Figure 7-4. FTP—details of session generation.

7.4.2 Lynx

Lynx sessions simulate users in the Air Force base and on the Internet using Lynx to surf one of the two websites on the Air Force base. Similar to the FTP traffic description in the previous section, the start time of Lynx traffic sessions are chosen according to a Poisson distribution with rate given by a timetable file. The session start times are generated according to algorithm shown in Figure 7-3 and the start time is checked against the attack schedule in case of conflicts.

Most Lynx traffic occurs between 8:00 a.m. and 5:00 p.m., as illustrated in Figure 7-2. Of the total 90 simulated users of Lynx, 45 of them reside on the inside Air Force base network while 45 reside on the outside Internet network. The user for each session is randomly chosen, as are the web pages accessed and the number of pages accessed per session (between 1 and 15). There is a one in 20 chance of accessing a bad web site. The average wait time between accessing web pages is between 10 and 70 seconds. The average typing time of a web page address is between 1 and 10 seconds. The general block diagram for Lynx traffic sessions is the same as that for FTP. Figure 7-5 shows a block diagram illustrating the details of the synthesis of Lynx sessions.



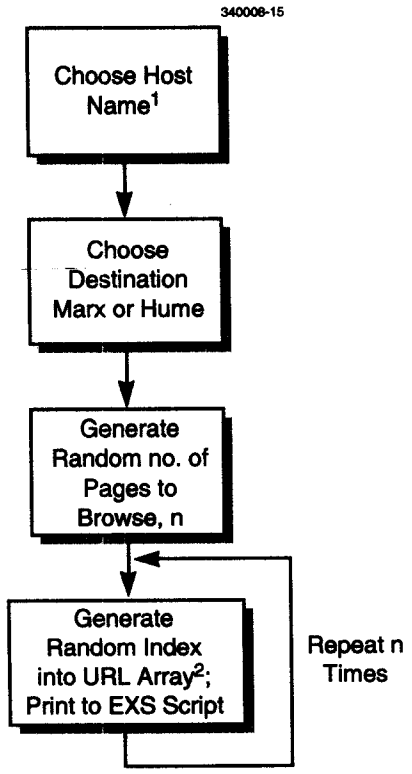
[1] From a list of 90 users. See Appendix.
[2] URL array contains a list of 262 URL addresses.

Figure 7-5. Lynx—details of session generation.

7.4.3 Browser

Browser session simulate users on the Air Force base and on the Internet who are surfing one or both of the web servers on the base using a generic web browser. Similar to the FTP traffic description above, the start times of browser traffic sessions are a Poisson distribution with time-varying rate, given by a file timetable. The algorithm shown in Figure 7-3 generates these session start times. Sessions are generated according to the same criteria as above and the start time is checked against the attack schedule in case of conflicts.

Most of the traffic occurs between 8:00 a.m. and 5:00 p.m., as illustrated in Figure 7-2. Sessions originate from a list of 10 Internet hosts and a list of web pages available for surfing are read into arrays to initialize the synthesis process. Browser traffic simulates users surfing Air Force base web servers, Hume and Marx, and thus only originate from the outside Internet network. Both the web page index and host are randomly chosen from the generated session. The number of pages per browser session is randomly chosen between 1 and 25. The range of wait times between accessing web pages is between 0 and 300 seconds, and the range of typing times of a web page address is between 0 and 10 seconds. The algorithm for synthesizing Browser traffic sessions is shown in Figure 7-6.



[1] From a list of 10 host names. See Appendix.
[2] URL array contains a list of 262 URL addresses.

Figure 7-6. Browser—details of session generation.

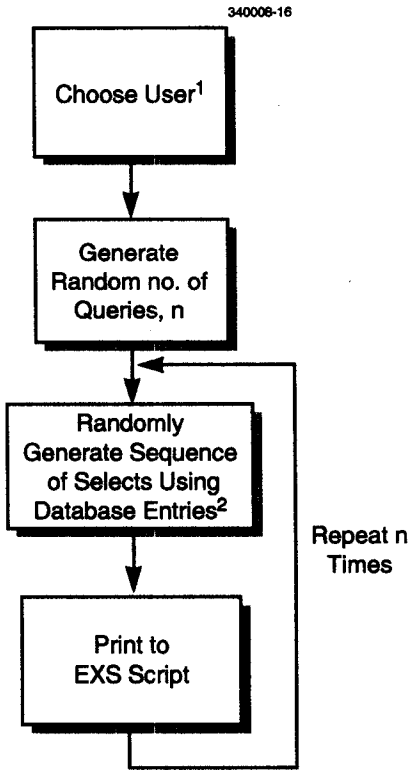
7.4.4 Internet Browsers

Internet browser sessions simulate Air Force base users surfing the Internet. These sessions are synthesized differently than the Lynx and Browser sessions described earlier. Prior to the evalutation, HTML and other content of these web-browsing sessions were captured using a custom web automaton run on the real Internet. This automaton was programmed to surf the web like a real user. It visited thousands of web sites popular with Air Force personnel with a frequency that depended on the site's popularity. Links at each site are traversed certain probabilities. The algorithm used to generate these web sessions and cache all content is described in section 5.3.6. As sites were hit and pages downloaded, each page was cached and the server name and IP address was tallied. This large cache or database of public-domain site content was transferred to the evaluation testbed. A special web server was configured to play back this content just as though it was coming from the original servers. Since the evaluation testbed was disconnected from the Internet for security reasons, live web sites could not be accessed. During simulation, these web-browsing sessions that were previously cached were converted into EXS scripts and replayed from the inside traffic generator. Session start times were again selected with Poisson distribution depending on time of day, and sessions were replayed with exponentially varied wait times between the retrieval of subsequent pages.

7.4.5 SQL

In SQL sessions, users telnet to an SQL server on the Air Force base and perform queries of the database. In this simulation network, the database contains information about makes, models, colors, and other attributes of cars and trucks. Similar to the ftp traffic description above, the start time of SQL traffic sessions is generated according to a Poisson distribution with time-varying rate given by a timetable in the SQL directory. Start times are generated with the algorithm shown in Figure 7-3, and are checked against the attack schedule in case of conflicts.

Most of the traffic occurs between 8:00 a.m. and 5:00 p.m., as illustrated in Figure 7-2. Users are chosen randomly from 10 possible SQL users. Telnetting to the server yields access to the SQL server. The number of queries made in each session is randomly generated, between 1 and 10. Queries are composed of a number of values of various database fields. They are formed randomly by choosing a color (between 0 and 12), a continent (between 0 and 6), an mtype (between 0 and 6), and a condition (between 0 and 5). The average wait time between SQL queries is 2 seconds, and the average typing time of a SQL command is 0.1 seconds. In the future this typing time will be made more realistic; 0.1 seconds is a bit fast. The block diagram for SQL traffic session synthesis is shown in Figure 7-7.



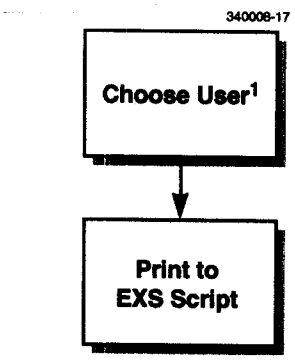
[1] From a list of 10 users. See Appendix.
[2] Database entries have four keys: color, continent, mtype, and condition. See Appendix.

Figure 7-7. Details of SQL session synthesis.

7.4.6 POP

Similar to the FTP traffic, POP traffic session start times are Poisson distributed with a time-varying rate given in a timetable and illustrated in Figure 7-2. Most of the traffic occurs between 8:00 a.m. and 5:00 p.m. POP session start times are generated according to the same algorithm as FTP, as shown in Figure 7-3. The start time is also checked against the attack schedule in case of conflicts.

POP sessions connect from PCs on the simulated Air Force base to the mail servers out on the Internet to download mail messages from those servers. The user that is performing the POP session is randomly chosen from 10 possible machines. Choosing the user dictates which remote mail server will be connected. Figure 7-8 illustrates a block diagram of how each POP session is synthesized. POP is a very simple protocol for retrieving email from a mail server. The only parameter chosen is the username, which in turn dictates the server.



[1] From a list of 10 users.

Figure 7-8. POP—details of session generation.

7.4.7 Mailread

Mailreading sessions are instances of Air Force base users reading and responding to their email using the UNIX "mail" program. Mailread traffic sessions can be generated from either the inside or outside traffic generators, Calvin or Hobbes. Interarrival times of mailread sessions are according to an exponential distribution with a mean that depends on the time of day and is derived from a simple mathematical expression rather than a timetable, as is the case with other traffic types.

The mean interarrival time is a function of the current time and a baseline mean interarrival time. If it is a weekend or a weekday, and before 8:00 a.m. (and after midnight), the mean interarrival time is twice the baseline. If it is a weekday, and after 6:00 p.m. (and before midnight), the mean is 10 times the baseline. Otherwise, between 8:00 a.m. and 6:00 p.m. on weekdays, the mean is set to the baseline.

There are three types of mail reading sessions: "out_to_in," "in_to_out," and "in_to_out_multi." Each type of session has an equal probability of being generated. Session out_to_in means that users will telnet from outside the network to inside the network to run the mail program. Session in_to_out means that users will telnet from the inside to the outside to run the mail program. Session in_to_out_multi means that users will telnet from inside the network to two outside machines to run the mail program.

Users have three retries for a bad login before a failed login occurs. When reading mail, there is a 0.8 probability of users reading the next mail message, a 0.8 probability of users answering a mail message if he reads it, a 0.5 probability of users deleting the next mail message, and a 0.2 probability of users quitting the current mail session.

The mailread session generation and synthesis loop begins at 0:00 a.m. Mail sessions are generated all day, but sessions that occur between 12:00 a.m. and 8:00 a.m. are automatically dropped. Mail sessions are also only generated if the status of the user (in or out) matches the source of the particular session type, for example, if a user is supposed to be "outside," then it is not possible for them to appear as though they are telnetting from the Air Force base. The current time does not increase until a mail session has been generated. Figure 7-9 shows a general block diagram of generating mailread traffic sessions. Figure 7-10 shows details of the generate session box and Figure 7-11 shows details of the run mail program box.

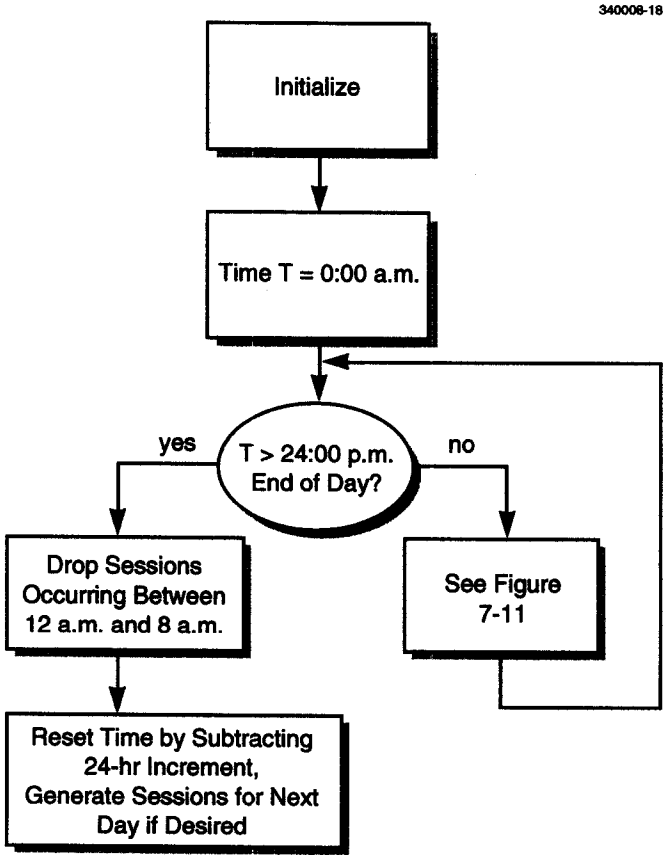
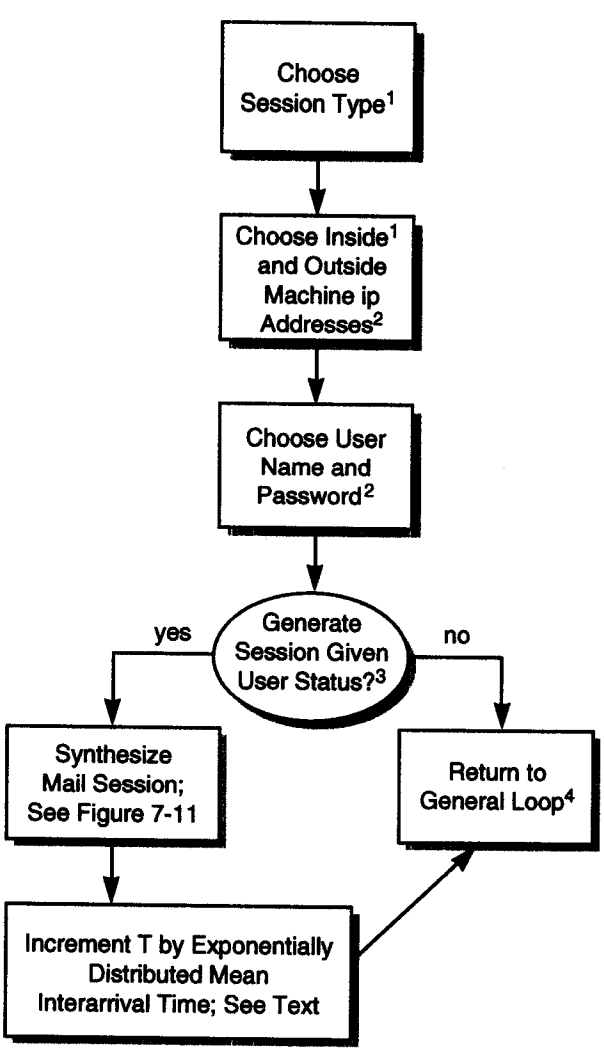
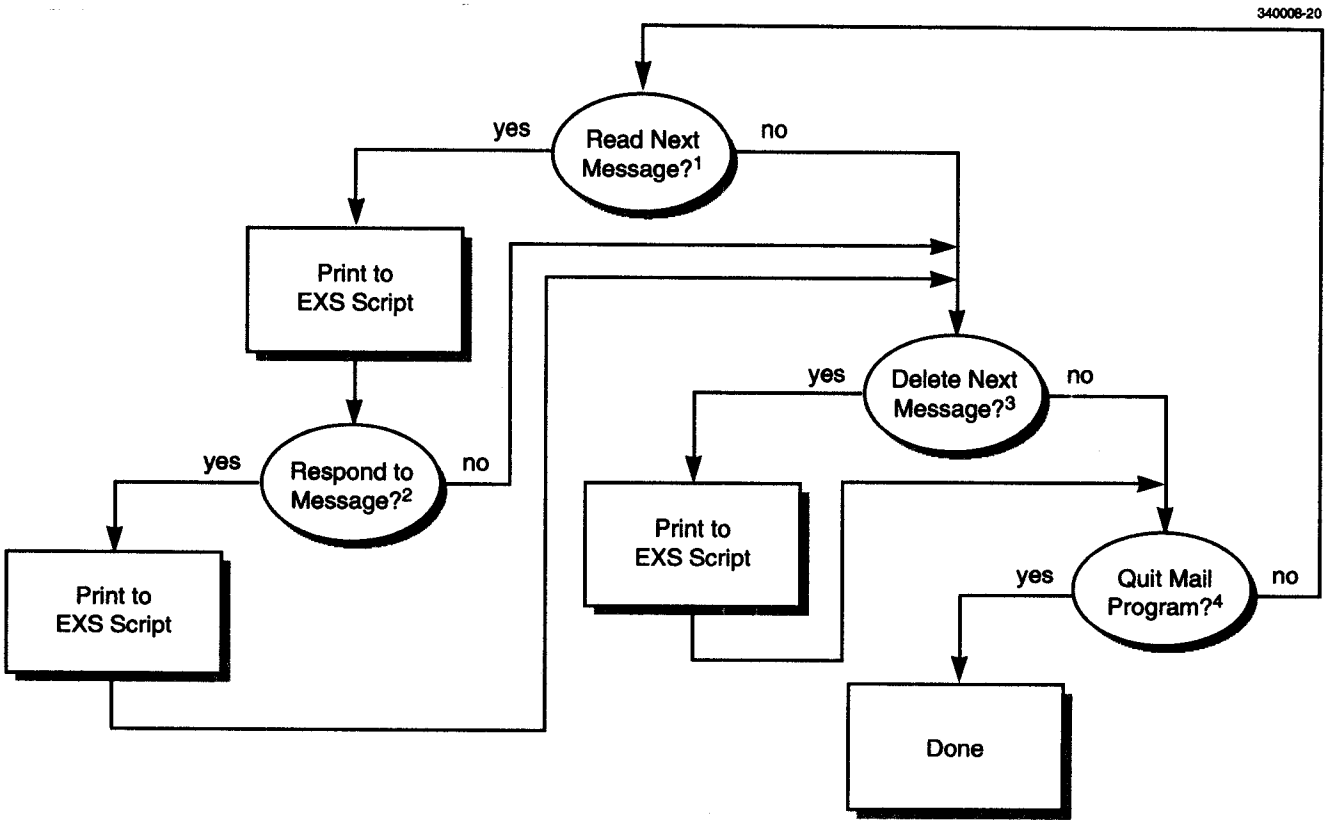


Figure 7-9. Mail read—general block diagram.



[1] Possible session Types are "out_to_in" (outside to inside machine), "in_to_out" (inside to outside machine), and "in_to_out_multi" (inside to two outside machines).
[2] Inside machines from a possible 23, outside machines from a possible 10, inside virtual machines from a possible 20, and users from a possible 90 (the same users as telnet).
[3] User status either "in" (telnetting from inside machine to outside machine) or "out" (telnetting from outside machine to inside machine).
[4] The current time is not incremented until a legal mail session has been generated.

Figure 7-10. Mail read—details of session generation.



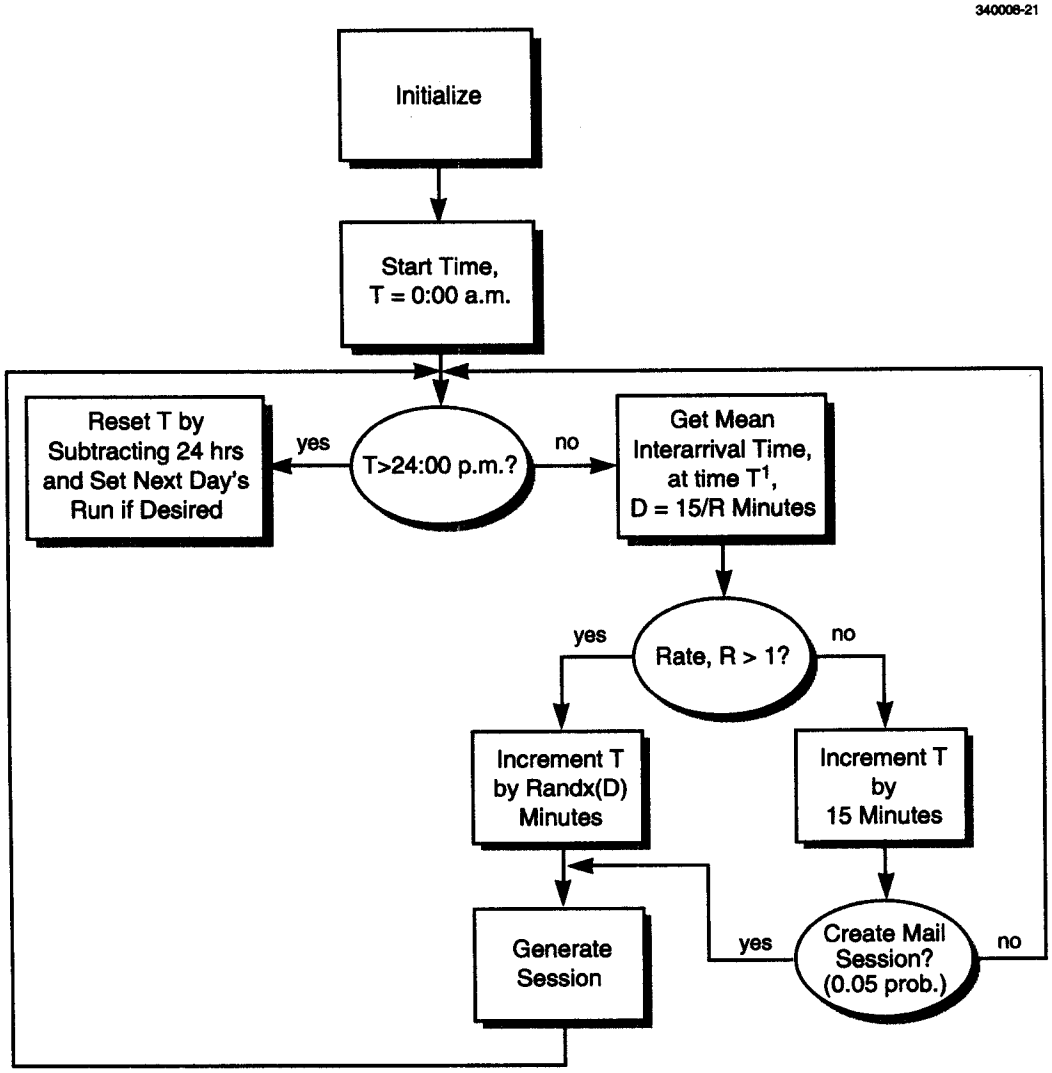
[1] Probability of reading next mail message is 0.8.
[2] Probability of answering mail message if message is read is 0.8.
[3] Probability of deleting next mail message is 0.5.
[4] Probability of quitting the current mail session is 0.2.

Figure 7-11. Details of navigating the mail program.

7.4.8 SMTP (Sendmail)

SMTP sessions simulate email being sent to and from users on the Air Force base and on the Internet. Similar to the FTP traffic description above, mail traffic sessions are distributed according to a Poisson distribution with time-varying rate given by a timetable. The algorithm shown in Figure 7-3 generates these start times, but with one difference. When the traffic rate is less than one session per 15 minutes, there is a 5% chance of creating a mail session anyway. Besides this difference, the session times are generated according to the same algorithm and the start time is also checked against the attack schedule in case of conflicts.

Mail traffic begins around 7:00 a.m. and continues until 6:00 p.m. There are a total of 689 users for Sendmail traffic. They are divided into 22 groups with 30 users and one group with 29 users. Figure 7-12 shows a general block diagram for generating mail message start times.



[1] From file timetable which contains the traffic rate at 15-minute intervals (from 0 to 23.75 hrs).
[2] Randx(D) is an exponentially distributed random variable with mean D.

Figure 7-12. SMTP—general session generation algorithm.

When mail sessions are synthesized, one of seven different types of mail messages are chosen: "within-broadcast," "within-discussion," "within-random," "global-random," "global-discussion," "global-broadcast,"and "list." Eight-five percent of messages are of type "within." Ten percent are "global" and 5% are "list." Of the traffic types that are not "list," 10% of them are "broadcast," 80% are "discussion," and 10% are "random."

For traffic type "within-broadcast," a group is chosen randomly from the 23 possible groups, and mail is sent to all 30 members of the group. Between 5 and 15 recipients will respond to the sender. There is, on average, 60 seconds between responses. See figure 7-13.

For traffic type "within-discussion," the group number is chosen randomly from the 23 possible groups, and mail is sent to between 1 and 10 of the group's 30 members. All recipients have a probability of responding to the sender. There is a 60-second mean time between responses. See Figure 7-13.

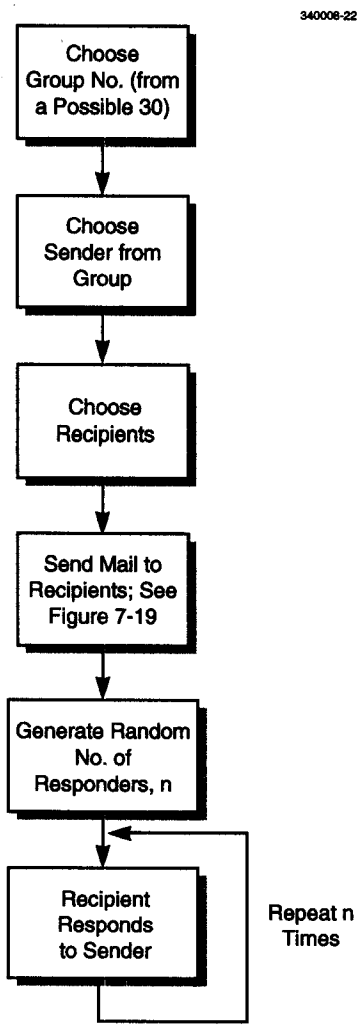


Figure 7-13. Broadcast and within-discussion traffic types.

For traffic type "within-random," the group number is chosen randomly from the 23 possible groups, and mail is sent to one person in the group (30 members). If this mail session is called from when the traffic rate is less than one, then the recipient will not respond to the sender because these are not normal working hours. For all other cases, the recipient will respond 50% of the time within an average of 60 seconds. See Figure 7-14.

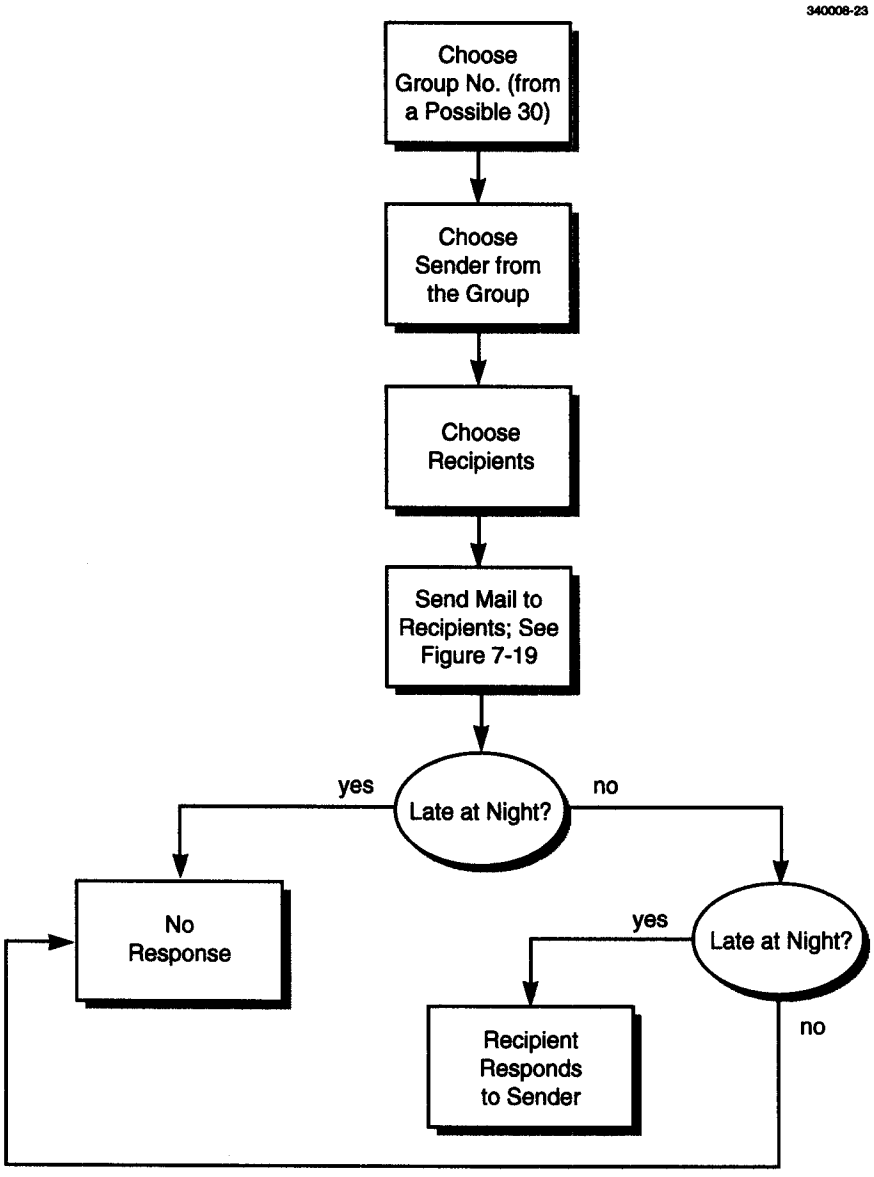


Figure 7-14. Generation of "within-random" email traffic.

For traffic type "global-random," mail is sent to one person out of the 689 users. The recipient will respond 50% of the time within a mean of 60 seconds. See Figure 7-15.

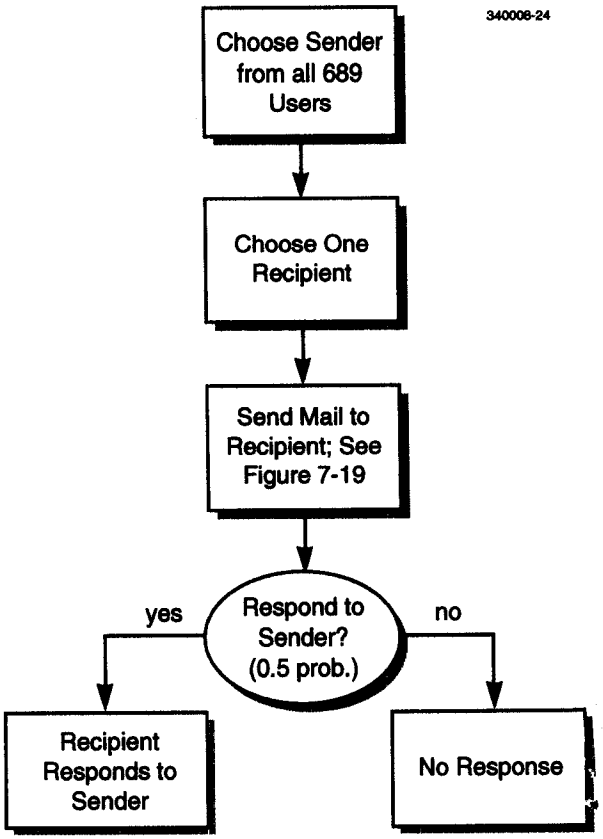


Figure 7-15. Generation of "global-random" email traffic.

For traffic type "global-discussion," mail is sent to 1–10 people out of 689 users. Between one and the total number of recipients will respond to the sender. There is a 60-second mean delay between responses. See Figure 7-16.

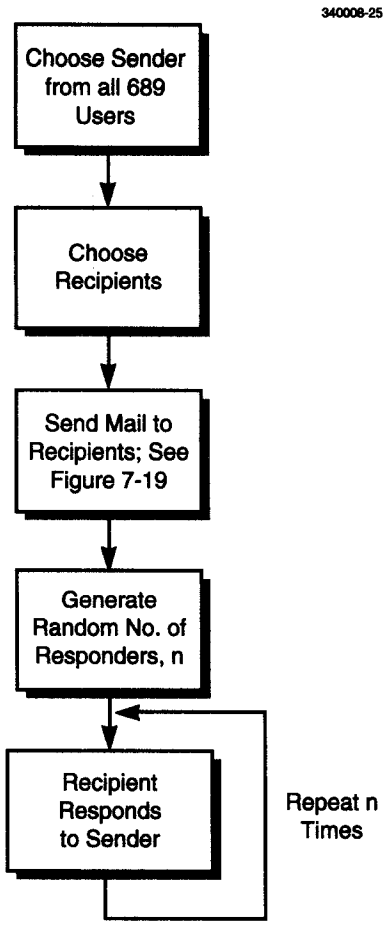


Figure 7-16. Generation of "global discussion" email traffic.

For traffic type "global-broadcast," mail is sent to all 689 users. Between 5 and 50 of the recipients will respond to the sender. There is a 60-second mean delay between responses. See Figure 7-17.

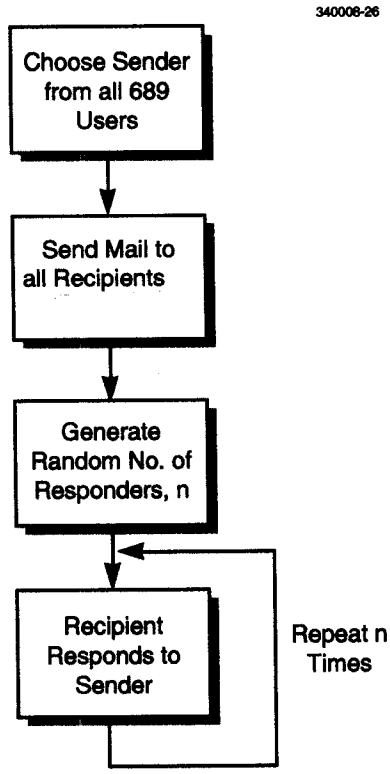


Figure 7-17. Generation of "global-broadcast" email traffic.

For traffic type "list," the list number is randomly generated (out of 13). Each listserv group contains between one and 10 members. See Figure 7-18.

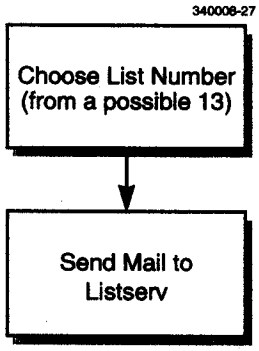


Figure 7-18. Generation of "list" email traffic.

Mail sessions can be generated from either the inside or outside traffic generators, thus from either Calvin or Hobbes. Mail sessions can either be run directly from the local machine (Calvin or Hobbes) or by telnetting into another machine. This telnet service is only available on the inside traffic generator, hobbes. The telnet service also requires users to log in with a valid username and password. There is a 0.1 probability that one bad login will occur and a 0.01 probability that two bad logins will occur (with equal probability of mistyping the username or password). There is a 0.001 probability that a failed login will occur. A finger will occur one in 10 times and a ping will occur one in 100 times. Finger and Ping represent users probing the network prior to sending their message. Figure 7-19 shows a block diagram of the subroutine for sending messages.

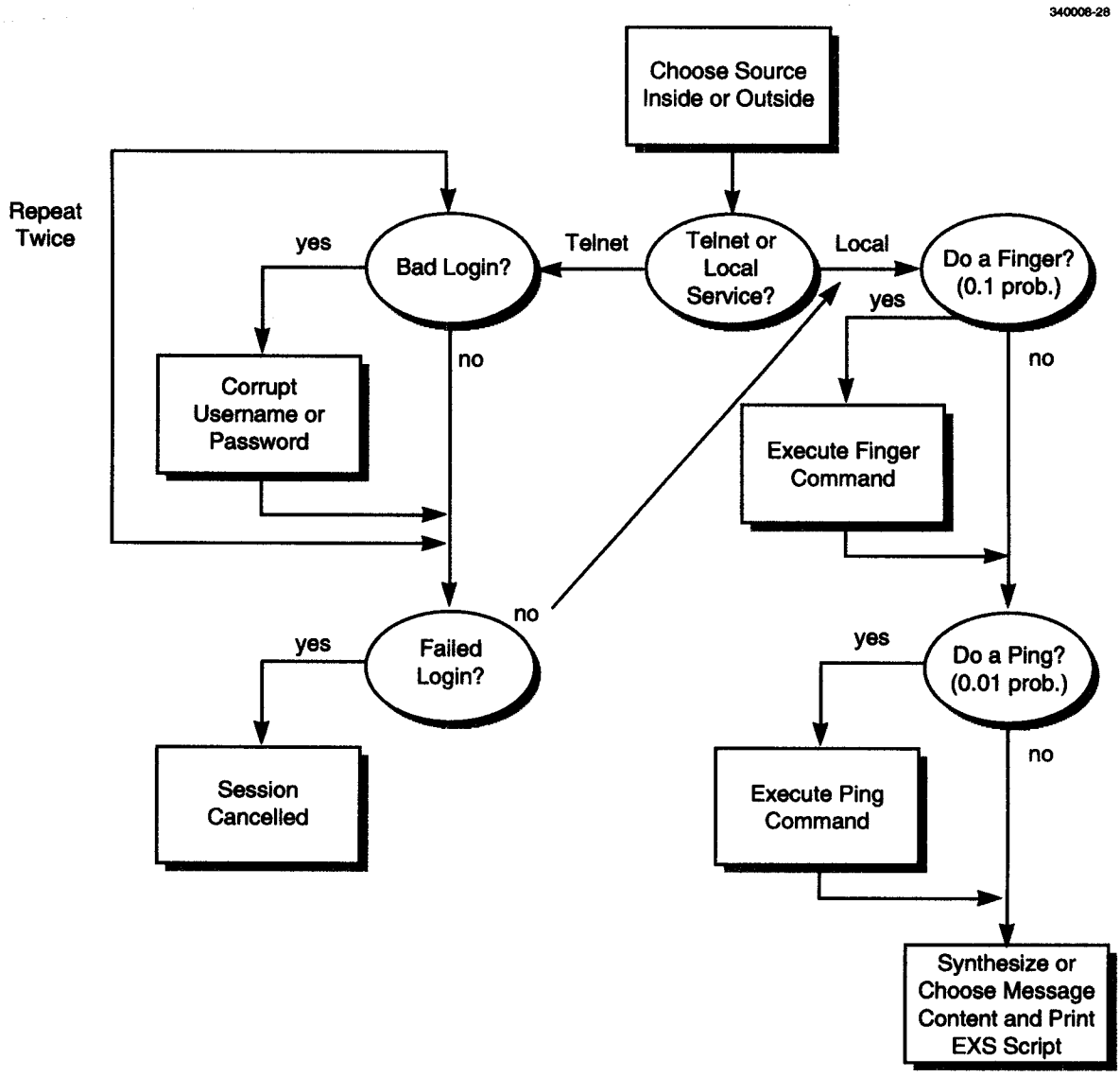


Figure 7-19. SMTP—details of subroutine for sending mail messages.

7.4.9 Telnet

Telnet sessions are synthsized rather differently than other traffic types. They are synthsized from statistical profiles of individual users to generate interactive sessions. These statistical profiles indicated the frequency of occurrence of different UNIX commands (e.g., mail, Lynx, ls, cd, vi, cc, and man), typical login times, session durations, source and destination machines, and other profiling information for each user. Table 7-2 shows the 54 commands the can be run from these telnet sessions and Figure 7-20 shows the relative probabilities with which these commands are chosen for two "programmer" users. User types included programmers, secretaries, administrators. For example, programmers primarily edited C programs, compiled these programs, sent mail, read UNIX manual pages, and ran programs. Secretaries edited documents and sent mail. Many other users primarily sent and received mail and browsed web sites. Public domain software source code files were created and compiled by these simulated programmers. Documents created and edited by secretaries were also those found in the public domain and placed in the test bed environment. The same custom *expect* script used for other background traffic automation was also used for telnet to create user automata which behaved as if users were typing at keyboards.

For the telnet traffic sessions, there are 90 users with complete user profiles given. Users were programmers, secretaries, system administrators, secret users, and regular users. User profiles were generated using a PERL script and specified the expected behavior of these different user types. The following describes each entry in the user profile:

USER_NAME: Username of user.

PASSWORD: Password of user.

FULL_NAME: Full name of user.

TITLE: Either programmer, secretary, sysadm, secret_user, or regular. Only users rexn, alie, virginil, and bramy are system administrators. Only users abramh, elmoc, quintond, and orionc are secret users. About 8% of the users are programmers and about 6% of the users are secretaries. The rest are regular users.

OUTSIDE_HOST: Name of host on the Internet (outside network) to or from which this user logs in.

INSIDE_HOST: Name of host on the simulated Air Force base (inside network) to or from which this user logs in. Secret users must go to either Pascal or Marx.

OUTSIDE_IP: IP address of the outside host.

INSIDE_IP: IP address of the inside host.

LOCATION: Where the user is currently located—on the Air Force base or not. Location takes the values either inside or outside.

PROB_MORNING: Probability of telnet session occurring in the morning. All programmers, secretaries, and system administrators log in every morning and afternoon, except for user alie and clintonl, who only log in at night.

PROB_AFTERNOON: Probability of telnet session occurring in the afternoon.

PROB_EVENING: Probability of telnet session occurring in the evening.

AVG_MORNING_START: The average morning start time of a telnet session occurs randomly between 9:00 a.m. and 1:00 p.m.

AVG_AFTERNOON_START: The average afternoon start time of a telnet session occurs randomly between 4 and 5 hours after AVG_MORNING_START.

AVG_EVENING_START: The average evening start time of a telnet session occurs randomly between 4 and 5 hours after AVG_AFTERNOON_START.

START_VARIANCE: The variance of the start times is between 0 and 60 minutes.

AVG_NUM_COMMANDS: The average number of commands is between 6 and 46.

AVG_WAIT_TIME: The average waiting time between telnet commands is between 0 and 2 seconds.

AVG_TYPE_TIME: The average typing time of telnet commands is between 0 and 20 seconds.

PROB_GLOB: The probability of mistyping a file name is between 0 and 0.1.

PROB_MISSTYPE: The probability of mistyping a command is between 0 and 0.2.

PROB_UPARROW: The probability of executing a command using the up arrow key is between 0 and .08.

NUM_COMMANDS: The total number of commands to select from (set at 54).

EDIT_FILES: A list of the files edited by the users. All programmers edit *.c files, all secretaries edit *.tex files, and all other users edit *.text files. There are 10 files, on average.

AVG_EDIT_LINES: The average number of lines to edit is between 5 and 20 lines.

MAIL_RECIPIENTS: A list of the mail recipients. There is a maximum of 20 different recipients.

MAN_PAGES: A list of the man page entries. There is a maximum of 20 different man pages.

DIRECTORIES: A list of directories.

FILES: A list of files.

PROB_TELNET_COMMANDS: The probability of executing each command within telnet. Only programmers will use gcc and only secretaries will use latex. For simplicity, no users will use both gcc and latex. Users on virtual machines will not use ps or netstat. Only system adminstrators will use su. Secret users are not allowed to use telnet but use ssh instead. Figure 7-20 shows the probability of each command being executed for two programmers and illustrates that they have a probability of executing many commands, but use cd, gcc, and vi most often. "Executable" indicates when a user executes a program that they have compiled, rather than a system binary or shell command. A unique array of programs is maintained for programmers. The command associated with each number on the x-axis is given in Table 7-2.

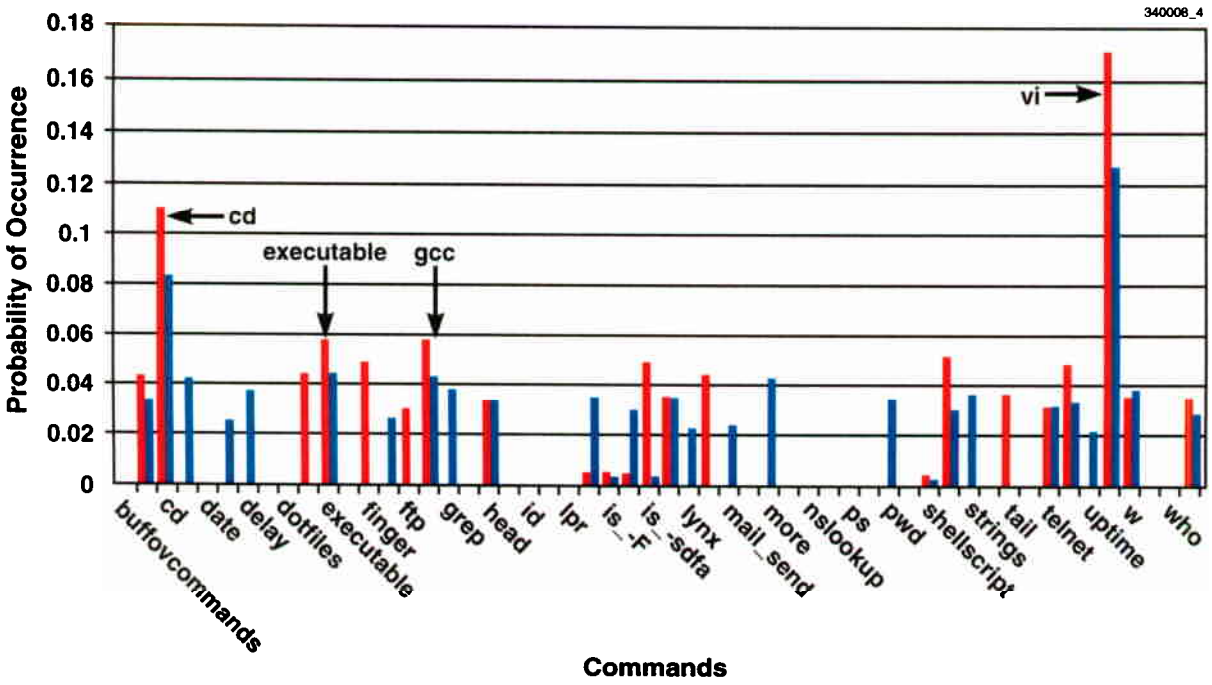


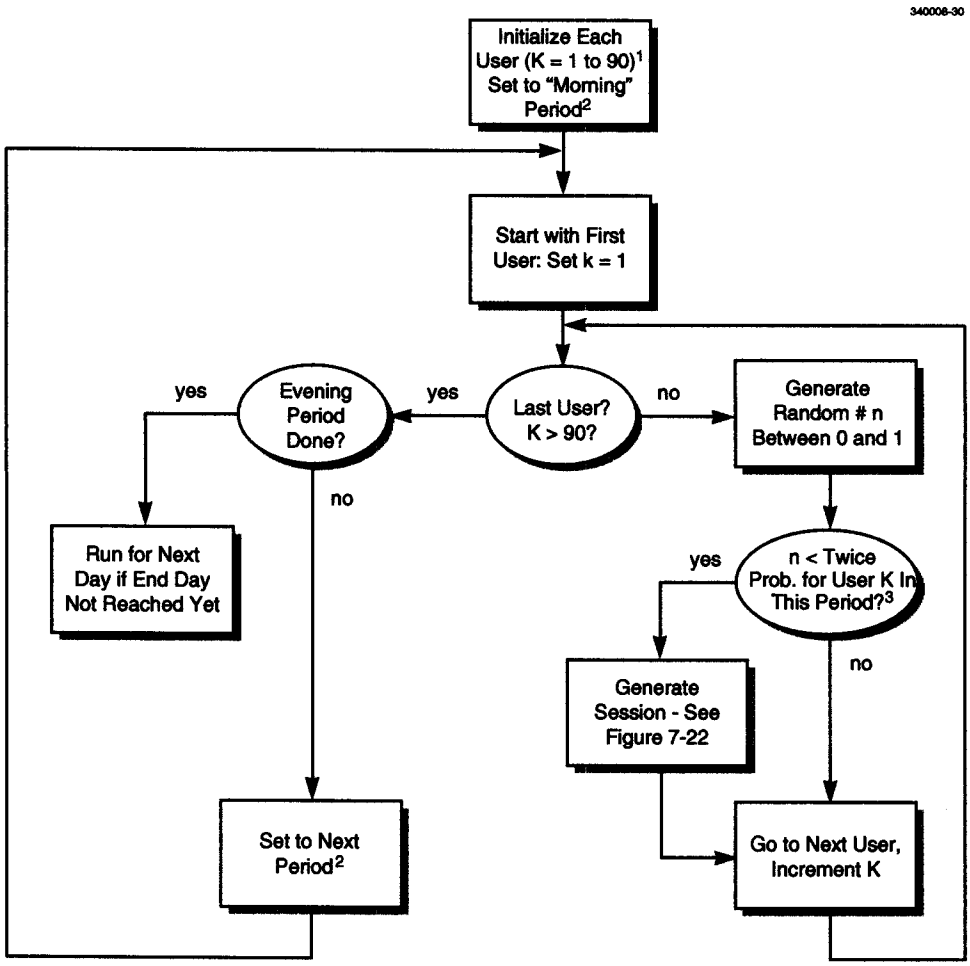
Figure 7-20. Probability of executing commands, for two programmers, during telnet sessions. Executable indicates the execution of a program compiled by that user.

TABLE 7-2

COMMAND ASSOCIATED WITH NUMBERS ON X-AXIS OF FIGURE 7-20

1	buffovcommands	19	head	37	ps
2	cat	20	history	38	ps_-aux
3	cd	21	id	39	pwd
4	chmod_+x	22	latex	40	secret
5	date	23	lpr	41	shellscript
6	dc	24	ls	42	shellvariables
7	delay	25	ls_-f	43	strings
8	df_-k_.	26	ls_-l	44	su
9	dotfiles	27	ls_-sdfa	45	tail
10	du_-s_.	28	ls_*	46	tcsh
11	executable	29	lynx	47	telnet
12	find	30	mail	48	uname
13	finger	31	mail_send	49	uptime
14	frrom	32	man	50	vi
15	ftp	33	more	51	w
16	gcc	34	netstat	52	wc
17	grep	35	nslookup	53	who
18	gzip	36	ping	54	whoami

Figure 7-21 gives a general block diagram of generating telnet sessions. To determine whether a telnet session is generated, a random number between 0 and 1 is generated. If twice the probability of a user telnetting to a computer during a certain period (defined as morning, afternoon, or evening) is greater than this random number generated, then a telnet session is generated. This check is performed for each of the 90 users for each of the three periods of the day.



[1] With information from user profiles.
[2] Periods are morning, afternoon, and evening.
[3] This is the probability of a telnet session being generated in the current period, from variables in user profile.

Figure 7-21. Telnet session synthesis.

If a user is chosen to have a telnet session in a given period of the day, the start time of the telnet session is calculated from the average start time for a given period and the start time variance. This time is also checked against the attack schedule and if there is a conflict, the session is cancelled. The following circumstances would result in a telnet session being cancelled:

1. Either the session is within 10,800 seconds (3 hours) after an attack or the session is under 4 hours before an attack and the telnet source is the same as the attacker source.
2. Either the session is within 10,800 seconds (3 hours) after an attack or the session is under 4 hours before an attack and the telnet destination is the same as the attacker source.

There are 10 chances to find a suitable start time before the telnet session is cancelled.

Figure 7-22 shows details of telnet session synthesis, the "generate session" box. Telnet sessions can be started from either calvin (outside traffic generator) or hobbes (inside traffic generator). There is a one in 20 chance that the telnet session goes to a Windows NT machine. The number of telnet commands executed is randomly generated (between 0 and twice the average number of commands as given in the user profile. The Windows NT commands to execute are chosen randomly from 16 different commands. The average waiting time between telnet commands is randx(average waiting time) where the average waiting time is given in the user profile and randx(W) is an exponentially distributed random variable with mean W. Likewise, the average time to type the command is randx(average type time).

If the telnet session is not going to an NT machine, then it is going to a UNIX machine. If the user is a "secret user," i.e., one that is allowed access to the secret files on each server, then the session has a 90% chance of being conducted via SSH rather than via telnet. The number of telnet commands executed is randomly generated between 0 and twice the average number of commands as given in the user profile. To determine which telnet command to run, a random number between 0 and 1 is first generated. This number is used to choose one of the 54 UNIX commands to run, using a uniform distribution weighted with each command's relatively probability of being chosen. The average waiting time and average typing time is calculated in the same way as for the NT case.

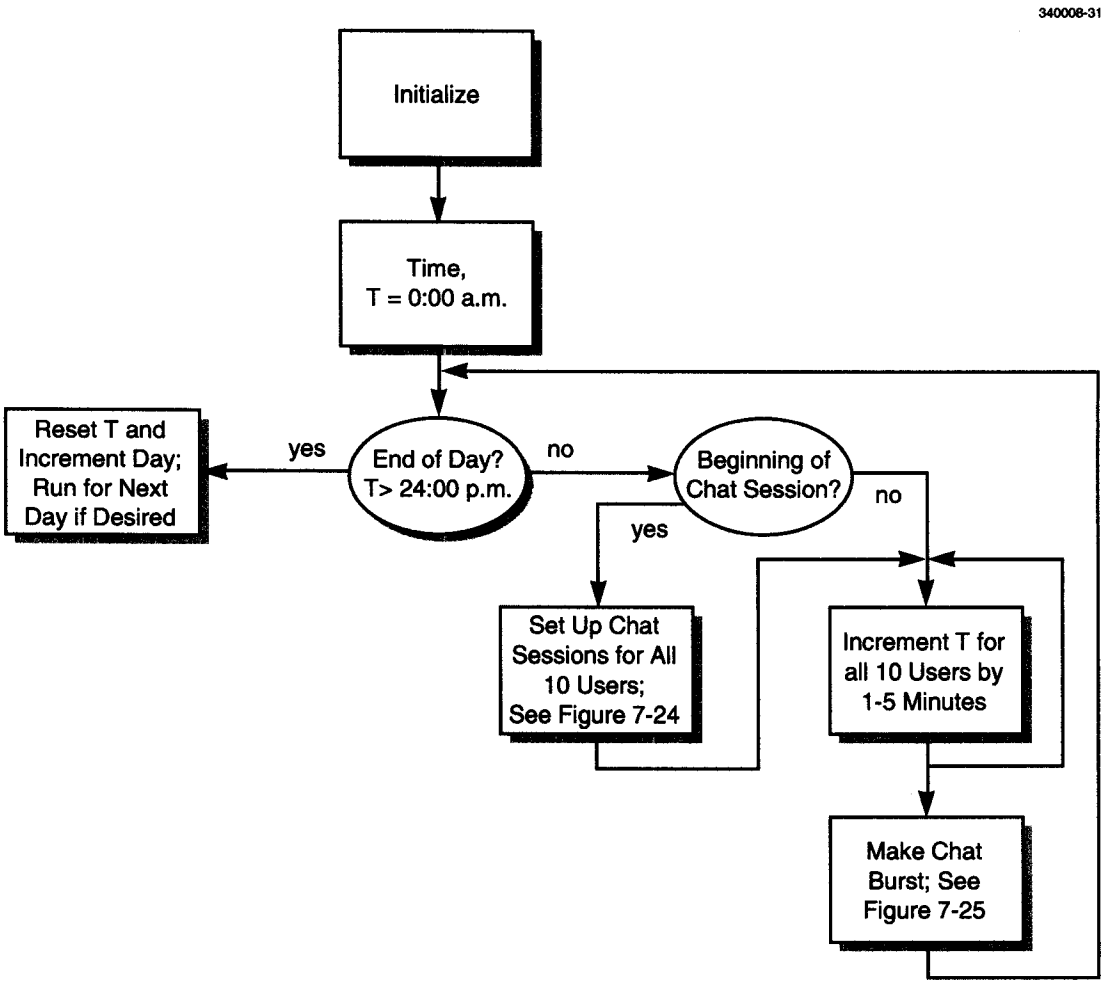


Figure 7-22. Telnet-details of session generation.

7.4.10 IRC

IRC sessions simulate 15 users participating in Internet Relay Chat sessions during the workday. For IRC traffic sessions, there are 10 Air Force users and five hacker users. Traffic is simulated from both the inside and outside traffic generators, calvin and hobbes. Air Force users log in sequentially, between five and 13 minutes apart, starting at 8:00 a.m. Communication happens in bursts of messages sent in succession between IRC users. The next user to talk is randomly generated, with a wait time of less than 10 seconds between different users talking. Content of Chat messages are chosen from a possible list of 4600 messages, but those with two words or less are omitted. Burst occur 1-5 minutes apart. IRC sessions continue for Air Force users until 5:00 p.m. Figure 7-23 shows a general block diagram for generating IRC traffic sessions. Figure 7-24 shows details of setting up the chat session in the morning and Figure 7-25 shows details of creating burst communication.

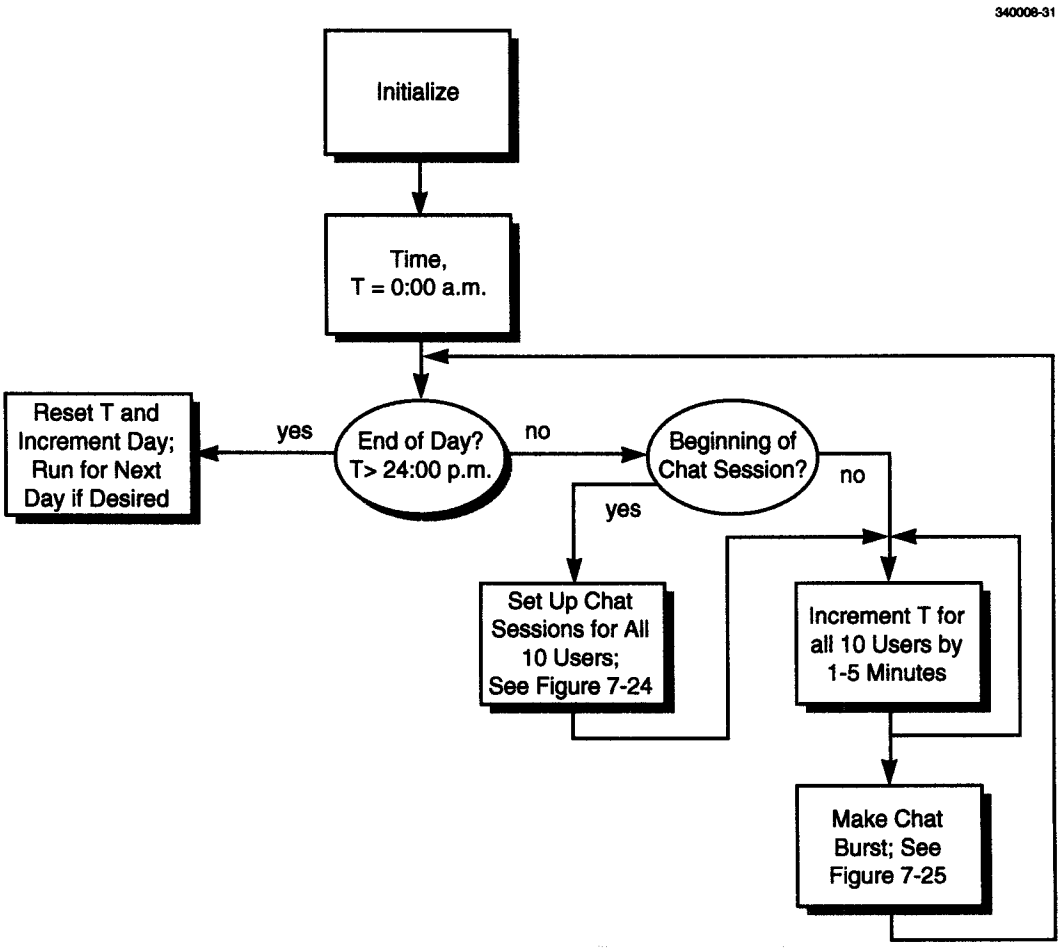


Figure 7-23. IRC session synthesis.

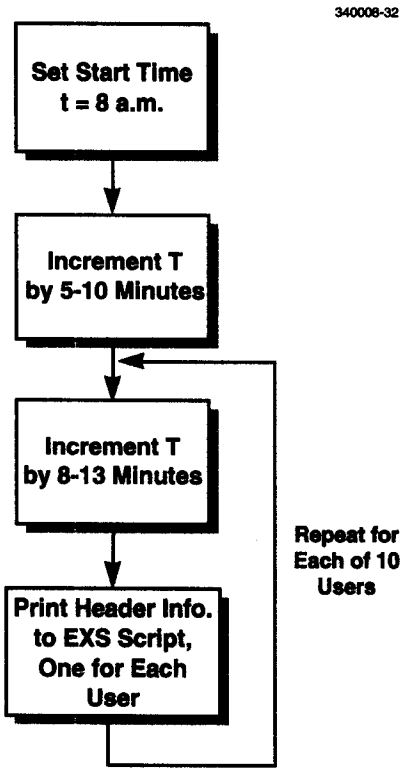


Figure 7-24. Details of setting up chat sessions at the beginning of the day.

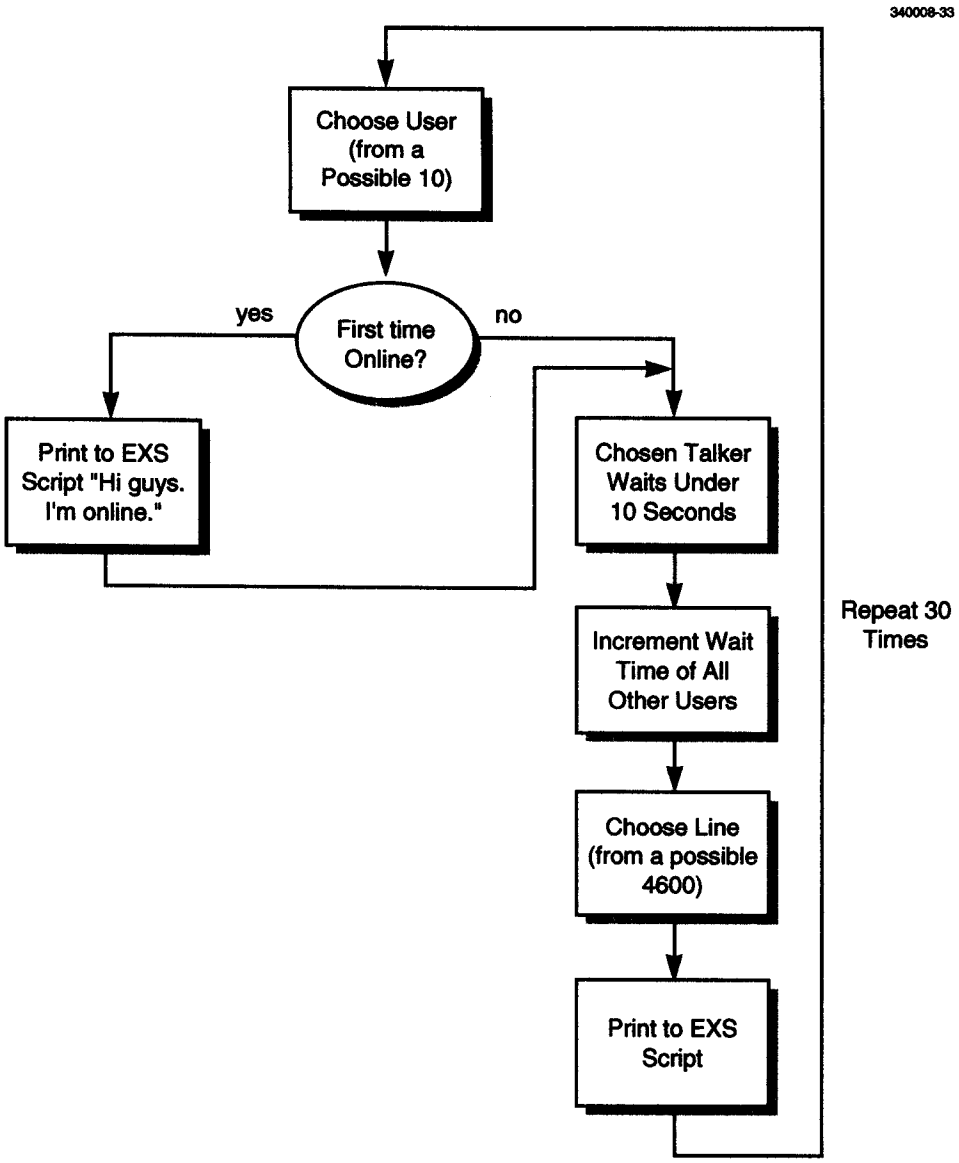


Figure 7-25. Details of creation of traffic bursts in IRC sessions.

The IRC traffic sessions for hackers are very similar to that of Air Force users, except for some minor differences. Hackers login after 10:00 a.m. and their interarrival times between logins is 5–13 minutes. The wait time between different hackers talking is less than five seconds and the wait time between bursts is between five and 10 minutes. Talking continues for hacker users until 7:00 p.m., and all other aspects of the sessions are the same as that for Air Force users.

7.4.11 Human Actors

It was not possible to automate every desired simualation action. Human actors performed some of the more complex tasks on the simulation network. They upgraded software, added users, changed passwords, remotely accessed programs with graphical user interfaces, and performed other system administration tasks from time to time. On the windows NT host, one administrator task was to run a batch script that collected statistics from the web server and posted them in a powerpoint presentation to a web page. On the order of one to three manual actions were carried out each simulation day as part of the background traffic. In the future we hope to script these actions, especially those on Windows NT machines, with macro scripts that are able to interact with Windows programs.

7.5 STATISTICS OF GENERATED DATA

This section presents some charts and graphs to illustrate some of the features of the background traffic synthesized by the software described in the previous section. It is not possible to include figures illustrating all facets of the background traffic, as the synthesis takes place at many levels and for many different types of traffic. Therefore, these charts describe some of the more important features. All of these figures are based on weeks one and three of the 1999 evaluation data; these are training data without attacks. Figure 7-26 shows the average number of Megabytes of traffic, per simulation day, for the major protocols, TCP, UDP and ICMP. Figure 7-27 breaks down the amount of traffic for TCP services for each service and shows the average number of connections for each per day. Figure 7-28 shows the total number of TCP connections per day for the major services.

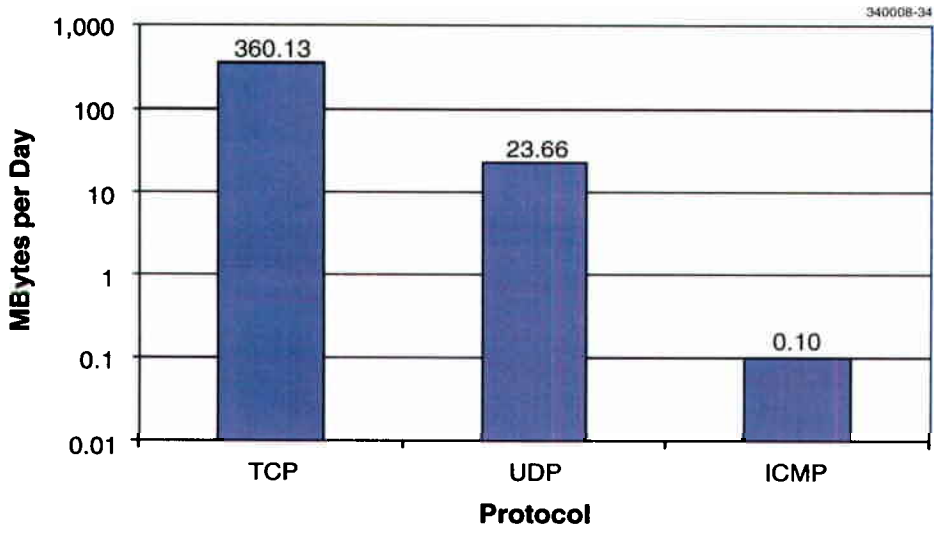


Figure 7-26. Average Megabytes per day for each major protocol in the 1999 training data.

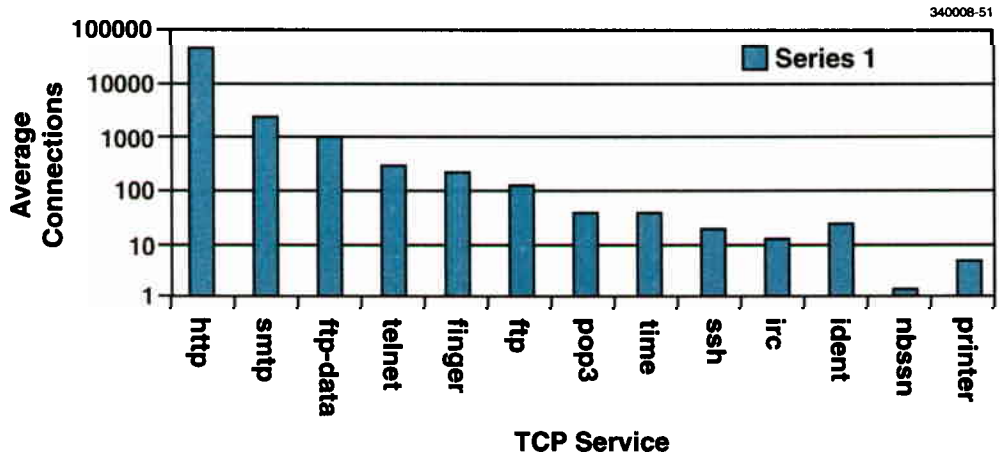


Figure 7-27. Average number of connections per day per TCP service in weeks 1 and 3 (which do not contain attacks).

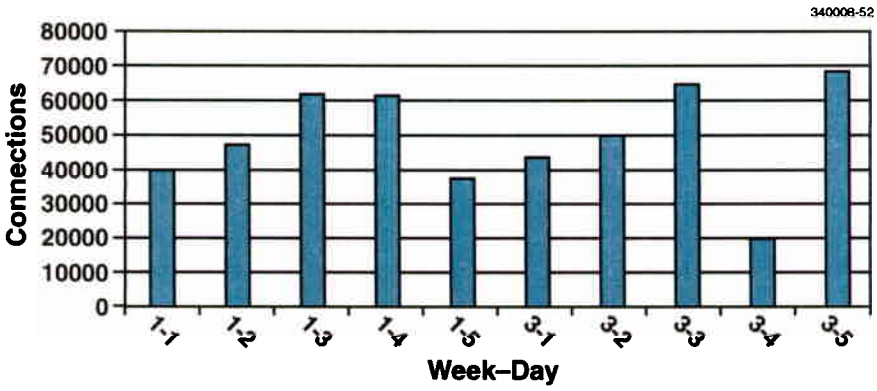


Figure 7-28. Number of TCP connections per day in the 1999 training data. Shown are weeks 1 and 3 which do not contain attacks. On week 3 day 4, there was a problem with the internet webserver, aesop, that resulted in fewer than expected web connections.

Figure 7-29, Figure 7-30, and Figure 7-31 illustrate the inherent inter-day variability in TCP, UDP, and ICMP traffic generated and used for background traffic in the evaluation. Each figure shows the average Megabytes of the three major types of traffic for each day in weeks 1 and 3. These are the weeks of data that do not contain attacks, thus the plots represent background traffic alone. The relatively large variability in amount of UDP and ICMP traffic from day to day illustrates the effect of having a testbed service on a server crash, even for a short period of time, resulting in excessive "ICMP destination unreachable" pockets or excessive DNS traffic, for example. If a problem occurred for a significantly long period of time (greater than a couple of hours) that day was re-run.

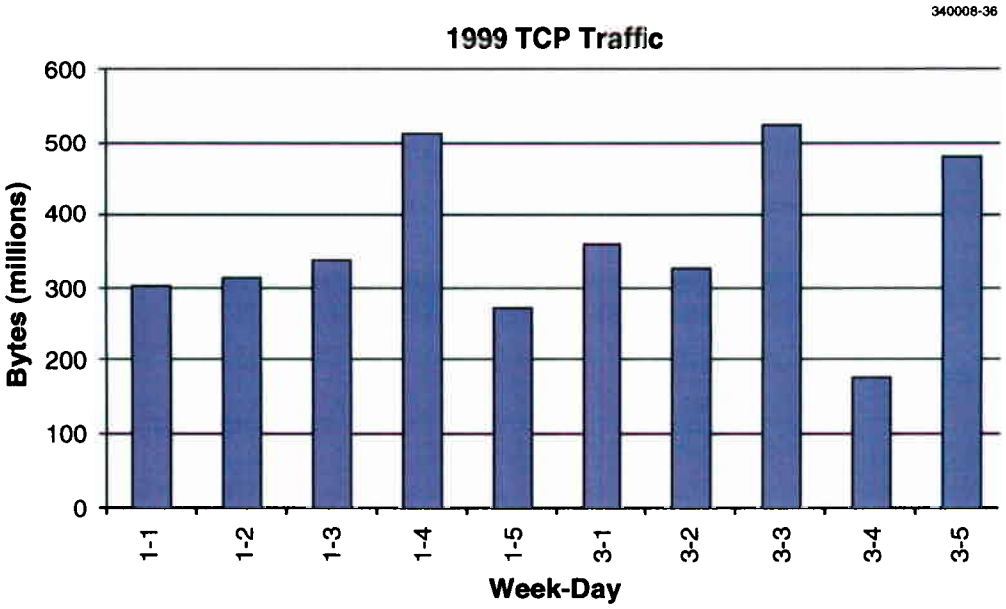


Figure 7-29. Volume of TCP traffic, in bytes, as seen by the inside sniffer. Weeks 1 and 3 are background traffic alone. Week 2 contains some attacks for training and is omitted here.

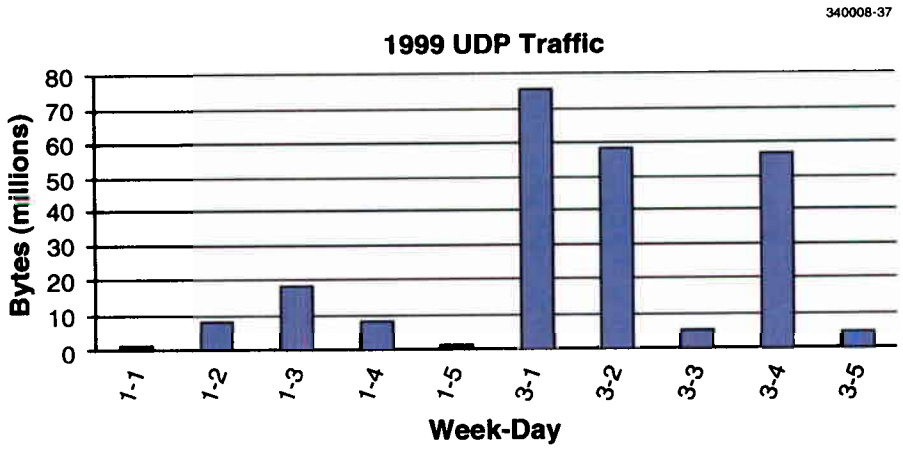


Figure 7-30. UDP traffic volume, in bytes, as seen by the inside network sniffer. Weeks 1 and 3 are background traffic alone. Week 2 contains attacks for training and is omitted here.

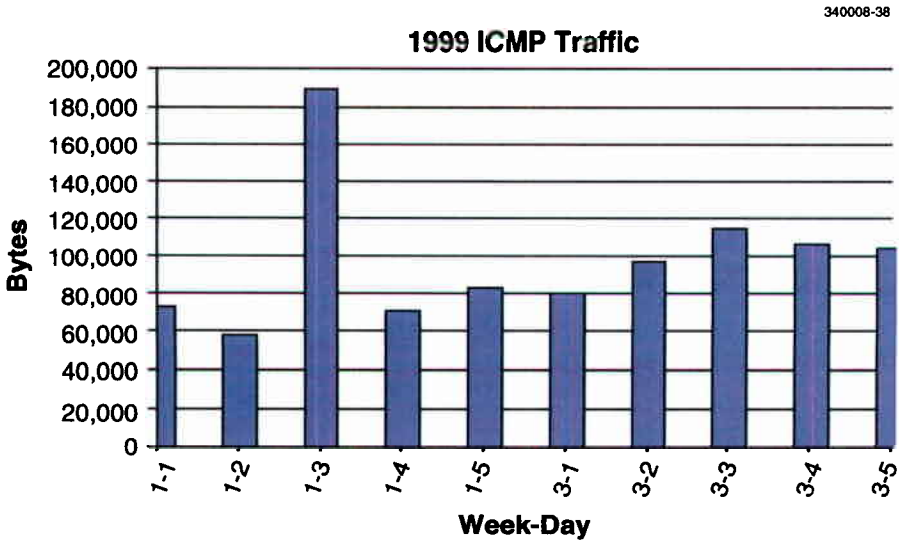


Figure 7-31. ICMP traffic volume, in bytes, as seen by the inside sniffer. Weeks 1 and 3 contain no attacks and thus are only background traffic. Week 2 contains some attacks for training and is omitted here.

Figure 7-32 and Figure 7-33 illustrate how HTTP and telnet connections are distributed throughout each simulation day. Each plot shows the number of TCP connections for port 80 (HTTP) and port 23 (telnet) for each 15 interval of the simulation day, from 8:00 a.m. to 6 a.m. the next day. Both plots are created from Tuesday of Week 3 of the training data, and do not represent averaging across multiple days, to give a good idea of what a single day of traffic might look like and how it varies with the time of day. As would be expected, most traffic is concentrated between 8:00 a.m. and 6:00 p.m., when most Air Force base personnel are at work. There are however some telnet and web connections later in the evening and in the early morning which model the late night, early morning use. Similar plots could be created for other types of traffic; only HTTP and telnet are given here.

Although similar to Figure 7-2, these diagrams show the number of Port 80 or 23 TCP connections, rather than the expected number of user level web (or telnet) "sessions." Naturally each web "session" will contain many port 80 connections, but also each telnet "session" could contain multiple actual telnet connections as users telnet from machine to machine.

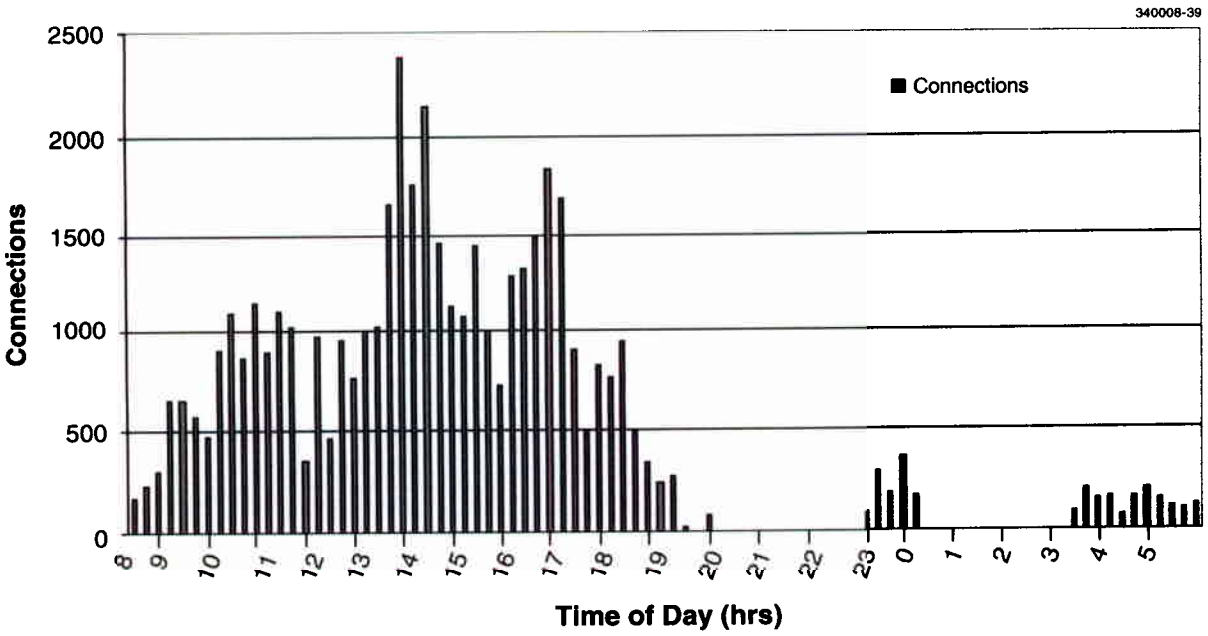


Figure 7-32. Number of HTTP service connections per 15-minute interval throughout week 3, Tuesday. This is proportional but not identical to the expected number of http user-sessions shown in Figure 7-2.

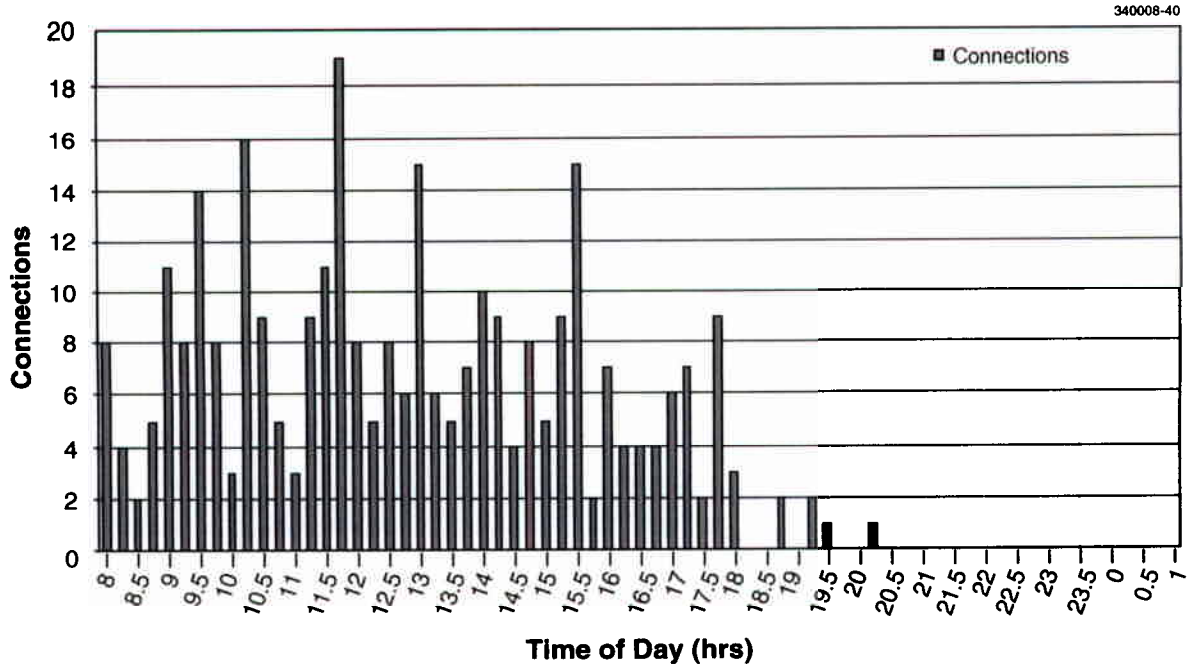


Figure 7-33. Telnet connections per 15-minute interval throughout the day, week 3, Tuesday.

8. SCORING PROCEDURE

This section describes the scoring procedure for the 1999 Off-Line Evaluation and its inputs and outputs. The 1999 Off-Line Evaluation scoring consists of three phases, as illustrated in Figure 8-1. Items on the left, in black, are those that the participant provides, while items and actions in the center column are things the evaluators at Lincoln carried out or provided for the evaluation. Items in gray are some of the products of the evaluation.

- **Detection scoring** is the initial phase of scoring that determines the accuracy of lists of detection alerts, and is discussed in Section 8.1.
- **Participant feedback** is a very important component of detection scoring in which evaluators and evaluation participants analyze evaluation results to verify them and better understand intrusion detection system performance. The feedback phase is discussed in Section 8.2
- **Identification scoring** relies on the output of the detection scoring phase to know which of the more extensive identification entries should be scored. Identification scoring is discussed in Section 8.3.

8.1 DETECTION SCORING

This section describes the detection scoring procedure. Section 8.1.1 lists and describes inputs to the scoring procedure, section 8.1.2 describes the algorithm used in detection scoring, and section 8.1.3 describes outputs received from detection scoring.

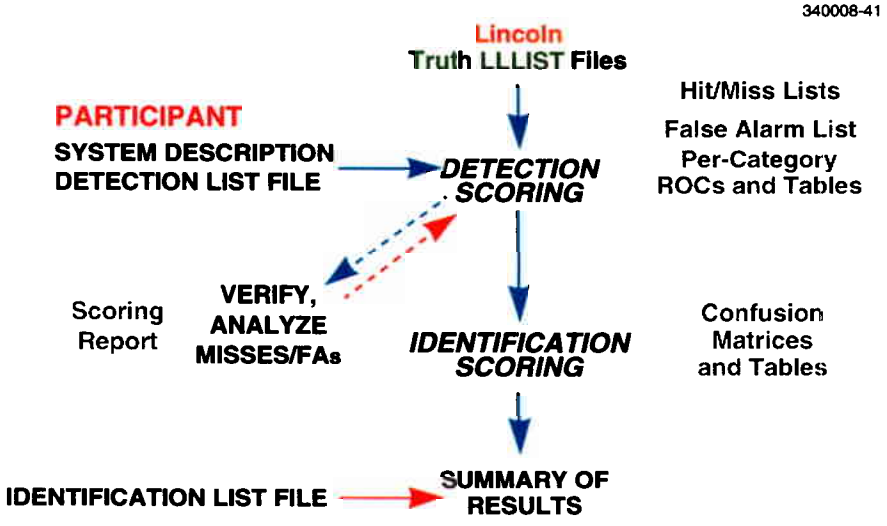


Figure 8-1. The scoring procedure used in the 1999 DARPA Intrusion Detection Evaluation.

8.1.1 Inputs

There are three inputs to the detection scoring procedure, each described in the following subsections:

- The list of detection alerts output by the intrusion detection system when run on the evaluation test data
- A machine-readable, technical description of the intrusion detection system
- The Lincoln Laboratory Detection Truth file

8.1.1.1 Detection List

The list of detection alerts, detections list, or detections.list, is produced by an intrusion detection system as it processes the evaluation test data. It is a list of the putative attacks detected by the intrusion detection system. Each line of the file specifies the time and date of the putative attack and the suspected victim of the attack. The following is an example of the format, illustrated by part of one detections list submitted for the evaluation.

```
17  03/29/1999  08:48:11  172.16.114.50  1.00000  #  R2L   sendmail  (pR2L,pLINUX)
800 03/29/1999  08:50:50  172.16.112.50  1.00000  #  U2R   ps  (pU2R,pSOLARIS)
801 03/29/1999  08:50:50  172.16.112.50  1.00000  #  DATA  secret  (pDATA,pSOLARIS
900 03/29/1999  11:45:40  172.16.0.1     1.00000  #  R2L   snmpgues  (pR2L,pCISCO)
901 03/29/1999  13:58:13  172.16.112.5   1.00000  #  R2L   ftpwrite (pR2L,pSOLARIS)
```

The fields are ordered as follows:

- ID number assigned by the participant; must be unique throughout the two weeks of detections
- Date of the suspected attack
- Time of the suspected attack
- Victim of the suspected attack
- Score, representing the certainty that this detection corresponds to a real attack; can be any positive number, but often ranges from 0 to 1.
- Fields after the # sign were optional: In this case the participant has assigned a category to each detection (U2R, R2L, DOS, Probe, Data), and given the attack name that might correspond to the attack. Items inside parentheses were appended by Lincoln Laboratory to allow the list to be sorted by category and victim operating system. Values assigned by the participant are appended with a "p" ("participant"). Lincoln Laboratory appended this since no requirement for this had been previously specified.

8.1.1.2 System Description

The system description is a machine-readable file composed of several sections, which contain several types of information:

- A machine-parsable table that indicates data sources the intrusion detection system is using for input.
- Two machine-parsable tables that indicate what types of attacks the intrusion detection system should find in the 1999 Lincoln Evaluation.
- Textual descriptions of how the intrusion detection system works and how it attempts to detect new attacks.
- A table indicating what types of attack identification (forensic) information the intrusion detection system will provide to be scored in the identification scoring.

Evaluation participants were required to submit system description files prior to running the evaluation test data through their systems. Here is an example of the input-data table submitted by one intrusion detection system for the 1999 evaluation:

```
1.1 Input Data Table:

TCPDump:              Inside  +   Outside  +

Audit Data:              BSM  +   NT  -

File Listing:       Solaris  -   SunOS  - Linux  -     NT  -

Log Files:          Solaris  -   SunOS  - Linux  -     NT  -

SRI File Info:      Solaris  -
```

The "+" signs indicate that this intrusion detection system takes, as input, both the inside and outside tcpdump files and the Solaris BSM auditing data. Here is an example of the attack-type table from one intrusion detection system's description file:

```
3.1 General Attack Table:

        Solaris    SunOS    Linux      NT      Cisco

Probe      +          +        +         +         +

DOS        +          +        +         +         +

U2R        +          +        +         -         -

R2L        +          +        +         -         -

Data       -          -        -         -         -
```

3.2 Inside Console Attack Table:

	Solaris	SunOS	Linux	NT
Probe	-	-	-	-
DOS	+	-	-	-
U2R	+	-	-	-
Data	-	-	-	-

The "+" signs indicate that the intrusion detection system is supposed to find:

- Probe and Denial-of-Service attacks against all hosts, when those attacks are carried out remotely with respect to the host that was being attacked
- User-to-Root and Remote-to-Local attacks against UNIX hosts, when those attacks are carried out remotely with respect to the host that was being attacked
- Denial-of-Service and User-to-Root attacks against the Solaris host, when those attacks are carried out from the console of the host being attacked

These are the attack types against which the system will be scored. The goal of collecting this information was to not score systems unfairly against attacks types that they were not designed to detect. As it turned out, these broad attack categories were a "good fit" for the abilities of some intrusion detection systems, but were too broad to accurately represent the abilities of others. In the future, finer-grained categories will be necessary for characterizing the abilities of a system. In addition, future evaluations might find it useful to score intrusion detection systems against all attacks in the data to discover the performance of systems on attacks that may be outside the scope of attacks that the designer wants the system scored against. Certain attack actions might be found to manifest themselves in unanticipated ways that could be detected by an intrusion detection system not necessarily designed to detect that type of attack.

8.1.1.3 Lincoln Laboratory Truth

The Lincoln Laboratory Detection Truth file is the truth against which the submitted detection lists will be scored. Although each intrusion detection system is scored against only those attacks that it is trying to detect, the scoring system starts with the same Truth file in each case. A description of the Truth file is given in Section 6.7.1.

8.1.2 Algorithm

Given the three inputs, Detection Truth, System Description, and list of detection alerts, the evaluation is scored. This section describes the scoring algorithm and discusses some of the design considerations, and refers to particular files that are used or generated by the scoring procedure. Often these

file names contain the wildcard term "system-name," which would be replaced with the name of each particular intrusion detection system as run in the evaluation. In addition, the list of detections created by the intrusion detection system and submitted for evaluation is referred to as "detections.list."

In the Lincoln Detection Truth file, each entry is labeled with keywords for the major category andthe victim operating system. Thus the keywords serve to provide an identifier for each entry of an attack.. The system description indicates which attacks (by major category and victim operating system) theintrusion detection system is looking for. The first step of the scoring procedure is to split the Lincoln. Detection Truth file into two files, specifically for the particular intrusion detection system being scored. The grep command is used to create two files:

- A *system-specific* truth list contains only entries for attacks that the particular intrusion detection system is supposed to detect in this evaluation. This file is called system-name.sslist.
- Attacks that the intrusion detection system is not attempting to detect are listed in a separate file called system-name.ignore. A list of sessions (Time, Duration, and Destination IP Address) corresponding to attack instances that did not run as planned is also added to the system-name.ignore list.

The next step is to determine which attacks have been detected by this intrusion detection system. This is a simple process of comparing each entry of the detections.list to each entry of the system-name.sslist. The comparison process consists of two steps:

1) First, the suspected victim of the putative detection alert, from detections.list, is compared to that of the truth entry. This comparison of the IP address or host name supplied with the IP address listed in the truth file. In the case of some network connections, like ftp data connections, the victim and attacker IP addresses in the truth file or submitted detections file would inadvertently be swapped. Thus the submitted IP address or host name is also compared to the attack's source IP address. If a successful match is found, the time comparison is carried out.

2) Next, the time of the putative detection alert, from detections.list, is compared to the time duration of the detection truth entry. If the submitted time is within one minute before the start of and one minute after the end of the detection truth entry, then the putative detection is correct and scored as a "hit." This time frame, illustrared in Figure 8-2, was added to negate the effect of clock skew between data sources used by the various intrusion detection systems, and is described below.

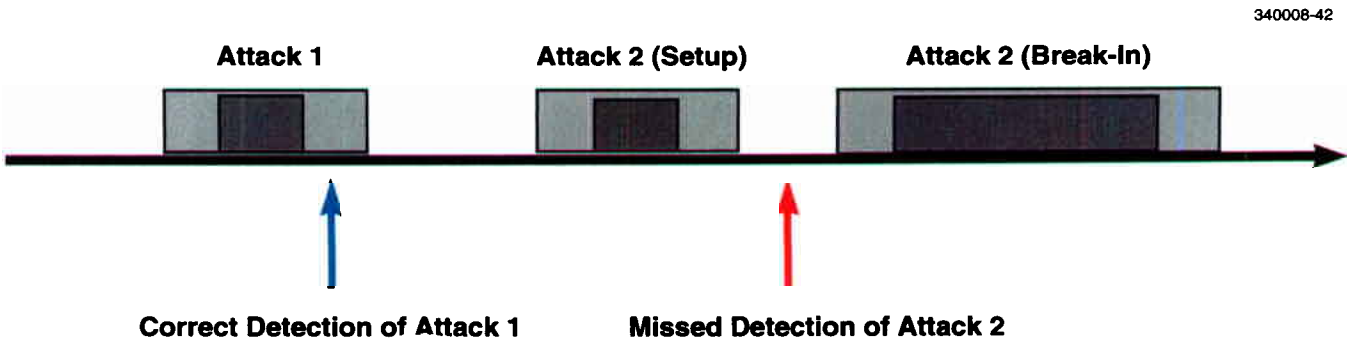


Figure 8-2. A window of one minute before and after each attack component, shown with lighter shading, was allowed for correct attack detection.

In Figure 8-2, two separate attack instances are shown: attack-one consists of a single component and attack-two consists of two components, a setup and break-in phase. The time duration of each attack session is shown with dark shading, the inner boxes, while the one-minute window before and after each component is shown with light shading, the outer boxes. Attack-one is detected when the detection alert's time falls within the one-minute window. An attack (and all components thereof) are successfully detected if a detection alert is submitted that matches just one of the attack components. However, as in the case of attack-two in Figure 8-2, detection alerts falling between two components of the same attack but not matching either one are not counted as "hits."

The one "special case" exception to this scoring procedure is that for certain denial-of-service attacks, the attack itself may consist of a single packet of very short session duration. However, the effects of the attack (a crashed server or service) might last much longer, perhaps until a human administrator is alerted to the problem and fixes it. Attacks like this in the 1999 evaluation included:

- **Teardrop** crashes the Linux machine; service is restored with a manual reboot.
- **Land** crashes the SunOS machine; service is restored with a manual reboot.
- **Crashiis** crashes IIS server on the Windows NT host; service is restored with a manual restart.
- **Syslogd** crashes the syslogd daemon on the Solaris host; service is restored with a manual restart.
- **Dosnuke** crashes the Windows NT machine; service is restored with manual reboot.

There are several ways these attacks could be scored in such an evaluation:

1. Detections could be allowed only in the one-minute window around the relatively small attack duration.
2. Detections could be allowed any time from one minute prior to the attack to the time when the server or service is restarted (potentially hours later).

3. Detections could be allowed in a window from one-minute before to a longer duration after the attack.

Since one way to detect a DoS attack is to detect the service requests that were denied service, it was important to allow for this in the scoring. A window of 15 minutes after each such DoS attack, illustrated in Figure 8-3, was allowed and detection alerts falling in that window were counted as "hits." For the selfping attack, which causes the Solaris host to reboot once, a window of three minutes was allowed after the initial attack, as the host took less than three minutes to reboot.

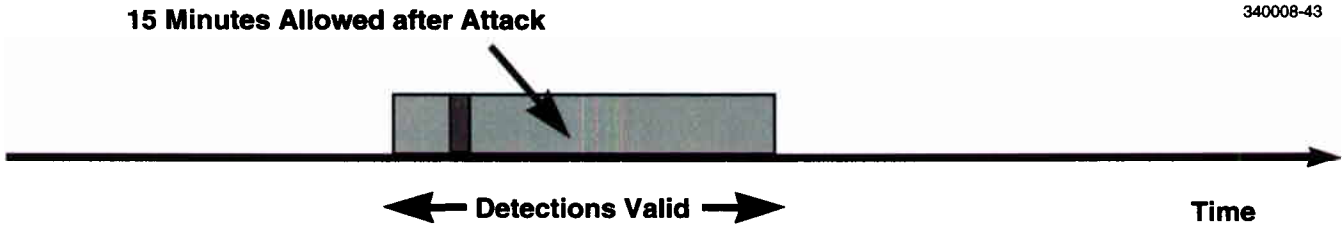


Figure 8-3. A detection window of 15 minutes was allowed after selected denial-of-service attacks that were very short in duration but whose consequences, in terms of denied connection requests, lasted much longer.

Entries of the putative detection alerts list that match an attack in the detection truth list are called "hits" while those that do not match an entry are called "false alarms." Thus two lists are created: hits and false alarms. The list of false alarms is trimmed by matching its entries to those entries in the system-name.ignore file. This eliminates false alarms that might be caused by detection of attacks that this system was not being scored against or that might be caused by attack instances that failed to run as designed. The list of hits is then sorted and trimmed so that it contains only one entry for each attack in the evaluation. In cases where multiple detection alerts match a given attack, the entry with the greatest score is taken. This list will be the input to the identification scoring procedure, as only the identification entries that correspond to detection alerts that match real attacks are scored. In addition, the scoring procedure yields two more important lists, first a list of attacks that were detected by the intrusion detection system, and a list of attacks that the system was supposed to detect but failed to.

The result of the scoring process is that the list of detection alerts is divided among hits, false alarms, and ignored alarms, and the list of attacks to be detected by the intrusion detection system is divided into two categories: "attacks detected" and "attacks missed." These output lists, and the Detection False Alarm (DFA) plots generated from them, are the subject of the next section.

8.1.3 Outputs

Outputs of the scoring procedure are contained in the "detection scoring reports" from the 1999 evaluation and are formatted in HTML. For an index of these reports, for participating intrusion detection systems, please refer to HTTP://ideval.ll.mit.edu/1999test/scoring_report_index.html.

Data included in these reports is described in the following sections.

8.1.3.1 Textual Output

The scoring procedure outputs several types of textual lists from which graphical output was created. The main types of data output are as follows:

- **System-specific list** is a list of all sessions of the attacks that this system should detect.
- **Hits list** is a list of all detection alerts output by the intrusion detection system that are ruled as hits; it is called `system-name.all_hits`. There is also a version of the hits list that gives only one entry per attack; the file is called `system-name.unique_hits`. In the case of an attack being detected by multiple putative detection alerts, the one with the highest score is given in "`unique_hits`."
- **False Alarms list** is a list of those detection alerts that were not ruled as hits, and is called `system-name.fas`.
- **Ignored Alerts list** is a list of detection alerts that were not ruled as hits but did match one of the attack sessions that were to be ignored in scoring this intrusion detection system; this file is called `system-name.ignored`. These are not counted as false alarms.
- **Misses list** is a list of attacks not detected by the intrusion detection system. In the detection scoring reports the list of missed attacks is called `system-name.missed`.

From the hit and false alarm lists, it is possible to draw DFA plots, discussed in the next section, to graphically represent the performance of an intrusion detection system.

8.1.3.2 DFA Plots

Detection score results are presented graphically with DFA plots. These plots show operating points achieved by the intrusion detection system. Points in the DFA plot are realized by varying a threshold from the highest detection score or confidence to the lowest. At each distinct score value the number of attacks detected and false alarms thus far encountered in the combined list of correct detections and false alarms, are tallied and plotted. There are a few important points regarding these plots in the 1999 evaluation:

- DFA plots are not Receiver Operating Characteristics, or ROC, curves. They are not ROCs since the X-axis cannot be interpreted as a probability when the maximum number of false alarms is unknown.
- In some instances points will be connected with lines for clarity, but the lines will not necessarily represent performance of a realizable system.
- In the 1999 DFA plots, convert False Alarms Per Day to Total False Alarms, by multiplying by 10, as there were 10 simulated days in the evaluation.
- These plots only apply to the background data and attacks used in the 1999 evaluation test data and should not be generalized to other sites with different traffic characteristics.

An example DFA plot, from the 1999 Evaluation, is given in Figure 8-4. The particular plot shows detection and false alarm performance for a particular intrusion detection system for all clear attacks in the two weeks of test data. It shows that there were 132 clear attacks that the intrusion detection system was supposed to detect and that the system incurred 13 false alarms over the two weeks. The X-axis gives the number of false alarms per day, while the Y-axis shows the percentage of those 132 attacks detected. The Xs indicate detection/false alarm points at which the intrusion detection system could operate, in the evaluation environment of those two weeks. Since this system output alerts scored with confidences between 0 and 1, as operators of the intrusion detection system, we can choose thresholds in terms of these confidences and only consider alerts scored at or above these thresholds. At a threshold of 0, considering all alerts, the system detected roughly 65% of the 132 attacks at a false alarm rate of just over one false alarm per day. At a scoring threshold of 1, considering only those alerts of highest confidence, the system detected 57% of the 132 attacks with far less than one false alarm per day over the two weeks of test data.

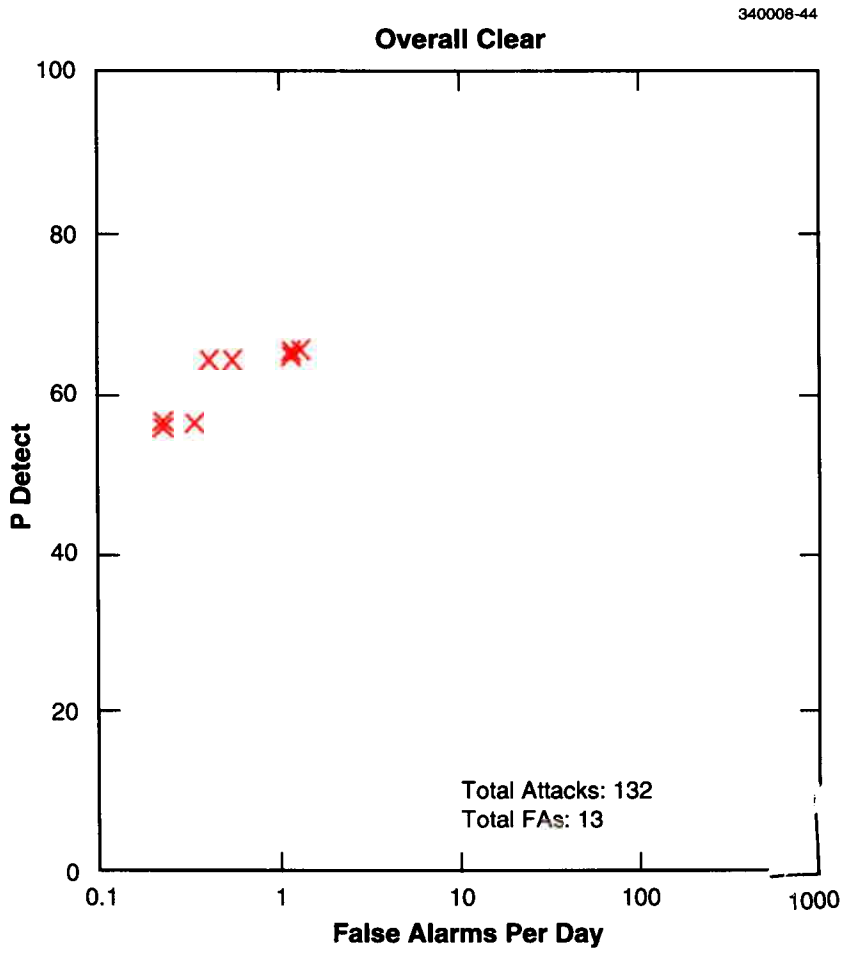


Figure 8-4. Example of a Detection False Alarm (DFA) plot from the 1999 off-line evaluation.

8.1.3.3 Results by Category

All lists (hits, false alarms, and misses) contain labels, in the form of unique text strings with each entry, that classify that entry with respect to the categories used in this evaluation. Making use of this, the detection scoring results are presented for most possible category combinations. Both lists and DFA plots are presented for all combinations of the attack types (DoS, Probe, R2L, U2R, and Data) and victim operating system (Linux, Solaris, SunOS, Windows NT, Cisco, All). Also, these groups are further broken down between Stealthy, Clear, New and Old. Results are generally not presented for a category if for the particular intrusion detection system, there were fewer than three instances of the attack type that the intrusion detection system was trying to detect.

8.2 MISS AND FALSE ALARM ANALYSIS

The final, and perhaps most important, step of the detection scoring procedure was an analysis phase to determine why missed attacks were not detected and why false alarms were incurred by each system. This feedback phase was a result of the Phoenix principle investigators (PI) meeting. Most participants agreed that a more thorough analysis of misses and false alarms was needed to determine why intrusion detection systems miss attacks and generate false alarms. This required participants to work closely with the evaluators to provide detailed analyses of misses and false alarms. To facilitate this analysis, preliminary scoring reports for each intrusion detection system were provided to participants. The remainder of this section contains a detailed description this feedback.

8.2.1 Analyzing Misses

8.2.1.2 Overview and Definition of a Miss

Misses are instances where an intrusion detection system does not detect an attack of a category it was designed to detect. Misses were analyzed to determine reasons for this type of failure. This was accomplished in three steps. In the first step, Lincoln Laboratory identified each miss and attempted to determine whether this miss was due to one of the following simple causes: (1) The intrusion detection system wasn't designed to detect this category of attacks or (2) No evidence of the attack was visible in the input information used by this intrusion detection system. In the second step, a file of misses created was returned to each participant. Each participant then analyzed this file, and confirmed or corrected the Lincoln Laboratory miss analysis, adding further reasons for each missed attack. In the third step, a final list of misses was drawn up, after a careful analysis of attack components and individual system characteristics. The process required many interactions and detailed analyses of individual attacks and intrusion detection system components.

8.2.1.1 Format of the Miss List File

As noted above, the format of the miss list is identical to that of the detection list, with the addition of a field that describes the reason for each miss. The added "reason" field will first contain from one

to three keywords, each corresponding to the answer to one of three questions, separated by dashes. These questions address why the attack was not detected are described in the next section. If there is no simple reason for the miss, then the "reason" field should also contain an open parenthesis, followed by two keywords dictated by the answers to questions 4 to 5 separated by a comma, followed by a close parenthesis. These questions are also described in the next section. Users can also optionally add a reference pointer by adding a bracketed number at the end of the "reason" field pointing to text at the end of the list. The following are examples of valid reason fields:

1. outofspec
2. inspec-invisible
3. inspec-visible-nofeature
4. inspec-visible-feature(bug,day)
5. inspec-visible-feature(variant,day)
6. inspec-visible-feature(new,week)[1]
7. inspec-visible-feature(normal,month)
8. inspec-visible-feature(badlabel)[2]

Using the standard UNIX command description conventions, the "reason" field contained:

```
spec-key-[visible-key]-[feature-key][(why-key,[fix-key])][[n]]
```

where:

spec-key = inspec or outofspec

visible-key = visible or invisible

feature-key = feature or nofeature

why-key = bug or badlabel or new or variant or normal or unknown

fix-key = day or week or month or year or impossible

n = a number referring to a comment after the end of the miss list

Questions and corresponding keywords are discussed in the next section.

8.2.1.3 Keywords and Questions

To assign reasons for misses in a relatively common format, it was necessary to define some keywords and outline the format for appending the reason for each miss to the appropriate line in the miss list text file. The format of the lists is identical to that of detection lists, with the addition of a field that describes the reason for each miss. The added reason field contains an ordered string of multiple keywords describing why the miss occurred with optional footnotes that provide detailed commentary, if necessary. The analysis procedure for each miss will address the following questions in order, stopping

early if a question determines the reason for the miss. Each question results in an additional keyword in the reason field:

Q1. *Was this attack "inspec" (within the specification of) this intrusion detection system?*

This question is used to determine if the intrusion detection system should have detected the attack. If the system wasn't designed to detect the attack, then that is the simple reason for the miss. The types of attacks an intrusion detection system should detect are specified in the system description. Attack types, in 1999, were specified using five broad attack categories (probe, denial-of-service, user-to-root, remote-to-local, data), five target victim host types (Solaris, SunOS, Linux, NT, Cisco), and whether the attack was launched from a victim's console and created no associated network traffic (console-based inside attack) or whether it was launched from a remote host to the victim host (remote attack). Systems should detect any attacks, including new attacks, and attack variants, that are from those types specified in the system description. Any attack of this type that is not detected is a miss.

The first keyword in the reason field of the miss file is "inspec" if the attack should have been detected or "outofspec" if the attack should not have been detected. If the attack should not have been detected, "outofspec" will be the only keyword in the reason field and no further analysis is required. If the attack should have been detected, "inspec" is added to the reason field and the analysis proceeds to question 2. For example, if the attack is a user-to-root attack launched from the console and the intrusion detection system only looks for probes, then the attack is out-of-spec. If the attack is a data attack transporting a secret file off a Linux host, and the intrusion detection system only looks for Linux user-to-root and denial-of-service attacks, then the attack is out-of-spec. However, if the attack is a new stealthy probe of all system types, and the intrusion detection system looks for NT probes, then the attack is in-spec.

Q2. *Was evidence of the attack "visible" in the data used by this intrusion detection system?*

An intrusion detection system cannot detect an attack if no evidence of the attack is visible in its raw, unprocessed, input data. If an intrusion detection system did not examine the input data required to detect an attack, then that is a simple reason for the miss. Input data types used by each intrusion detection system are listed in the system descriptions. Input data types include inside tcpdump data, outside tcpdump data, NT audit data, BSM audit data, daily file system listings for each host, daily log file contents for each host, and daily SRI file analysis data for the Solaris host.

The second keyword in the reason field of the miss file should be "visible" if evidence of the attack was contained in the input data used by the intrusion detection system and "invisible" if not. If no evidence of the attack is contained in the input data used by the intrusion detection system, then the reason field of the miss file will contain only "inspec-invisible" and no further analysis is required. If evidence of the attack is contained in the input data, then the "reason" field should contain "inspec-visible" and the analysis proceeds to question 3. For example, if the attack is a user-to-root attack launched from the Solaris console which uses a buffer overflow of a system program to launch a /bin/csh at root level and, and the intrusion detection system uses Solaris BSM audit data, then the attack is visible. However, if the intrusion detection system looks for user-to-root attacks on the Solaris host, but only uses inside and outside tcpdump data as input, then the above console attack is invisible.

Q3. *Did the "features" extracted from the input data preserve evidence of the attack?*

This question determines if processing of the raw unprocessed input data preserved evidence that could be used to detect the attack. If an intrusion detection system doesn't preserve and extract those features that indicate an attack occurred, then that is a simple reason for the miss. Although the system description provides some information concerning features, only the participant who designed an intrusion detection system can best answer this question.

The third keyword in the reason field of the miss file is "feature" if evidence of the attack is visible in features extracted from the input data and "nofeature" if not. If no extracted feature contains evidence of the attack, then the reason field of the miss file contains only "inspec-visible-nofeature" and no further analysis is required. If evidence of the attack is contained in extracted features, then the reason field should contain inspec-visible-feature and the analysis proceeds to question 4. For example, consider an intrusion detection system that uses BSM audit data, but extracts only the name of the audit record as an input feature and uses N-grams of these names as features. If an attack is a secret attack on the Solaris host where an unauthorized user examines a secret file, then information concerning the files accessed is not used by the intrusion detection system (the file path name is a system call argument and a component of the BSM audit record), and the attack is a "nofeature" attack. However, if the intrusion detection system examines the path argument in the BSM audit record, then this is a feature attack.

Q4. *If the attack was inspec and input features contained evidence of the attack, why wasn't it detected?*

The intrusion detection system designer must provide a "why" keyword by carefully analyzing their system and the attack. Some possible simple "why" answer keywords are:

- *bug* - There was an implementation bug. The system would have detected the attack but wasn't tested thoroughly enough.
- *badlabel* - The 1999 corpus was labeled incorrectly. The attack really did not occur and the Lincoln Laboratory truth labels should be fixed or the 1999 security policy should be modified to clarify this attack.
- *new* - This is a new attack and it wasn't detected because the system does not include a model or signature for this attack.
- *variant* - This is a variant of an old attack that wasn't detected even though there was a signature or model for the old attack.
- *normal* - The features used to detect this attack are too similar to features for normal, non-attack, behavior.
- *unknown* - This is a keyword that Lincoln Laboratory will put in to be replaced by the participant.

Participants were not limited to these reasons and were allowed to provide other keywords to indicate why the attack was missed. In addition, they can provide a detailed footnote indicated by a bracketed number (e.g., [1]) with text at the bottom of the miss list file. Any new keywords used by participants were defined at the bottom of the miss list file. After this keyword is provided, question 5 must be answered.

Q5. *How hard would it be to extend the intrusion detection system to detect this attack?*

A final "difficulty" keyword is used for participants to estimate how difficult it would be to modify their system to detect the missed attack. The following order-of-magnitude keywords are allowed:

- *day* - The modification is trivial and can be performed in minutes to hours. The modification can be made and tested to verify correct detection and measure the effect on false alarm rates for normal traffic in less than a day.
- *week* - The modification is more extensive and requires a few days.
- *month* - The modification is more extensive and requires a few weeks.
- *year* - The modifications required are extremely extensive and would require months of effort.
- *impossible* - The participant thinks that this attack could never be detected by anyone because it is too similar to normal traffic, or for some other reason.

8.2.1.3 Example of a Miss List File

The following is an example of a miss file created first by Lincoln Laboratory and then with reason keywords filled in by a participant. This file explains why some of the attacks in the second week of the 1999 training data were missed. The hypothetical system that missed these attacks looks for probes and denial-of-service attacks against all systems using outside tcpdump data and for user-to-root attacks using outside tcpdump data. It has no signatures to detect NT probes or NT denial-of-service attacks. The identification number and other information about the attack is that assigned by Lincoln Laboratory in the Lincoln Detection Truth list. In the following miss list file, only attacks with identification indices 2,9,10, and 13 overlapped with attacks in the returned detection list file. This miss file also includes one footnote that provides details corresponding to attack-two in the Lincoln detection list file.

IDnum	MM/DD/YYYY	StartTime	Destination	Score	#	Attack	Reason
21.080101	03/08/1999	08:01:01	hume.eyrie.af.mil	1	#	NTinfoscan	inspec-visible-feature(new,week)
21.085015	03/08/1999	08:50:15	zeno.eyrie.af.mil	1	#	pod	inspec-visible-feature(badlabel)[1]
21.172713	03/08/1999	17:27:13	marx.eyrie.af.mil	1	#	secret	outofspec
22.094351	03/09/1999	09:43:51	pascal.eyrie.af.mil	1	#	eject	inspec-invisible
22.100643	03/09/1999	10:06:43	marx.eyrie.af.mil	1	#	back	inspec-invisible
22.105419	03/09/1999	10:54:19	zeno.eyrie.af.mil	1	#	loadmodule	inspec-visible-feature(bug,day)
22.114913	03/09/1999	11:49:13	pascal.eyrie.af.mil	1	#	secret	outofspec
22.142516	03/09/1999	14:25:16	pascal.eyrie.af.mil	1	#	mailbomb	inspec-visible-nofeature
22.130510	03/09/1999	13:05:10	172.016.112.001-172.016.114.254	1	#	ipsweep	inspec-visible-feature(variant,week)

IDnum	MM/DD/YYYY	StartTime	Destination	Score	#	Attack	Reason
22.161115	03/09/1999	16:11:15	marx.eyrie.af.mil	1	#	phf	outofspec
23.235614	03/10/1999	23:56:14	hume.eyrie.af.mil	1	#	crashis	inspec-visible nofeatur (new,week)

[1] This pod attack was implemented incorrectly because the fragments have different id numbers and would not be reconstructed into one packet at the victim machine. This is thus not a valid attack.

8.2.2 Analyzing False Alarms

8.2.2.1 Overview and Definition of False Alarms

False alarms are instances where an intrusion detection system reports that an attack occurred, but the report does not correspond to a true attack. In the evaluation, false alarms are entries in the detection list with high scores that do not correspond to attacks. High scoring false alarms seriously degrade the usefulness of an intrusion detection system. They were analyzed to determine why they were generated. This happened in three steps, similar to the analysis of misses. In the first step Lincoln Laboratory identified each false alarm (according to the scoring procedure) and created a list of false alarms that was returned to each participant as a part of the preliminary scoring report. Participants then analyzed this file to confirm or correct the Lincoln Laboratory false alarm entries, and provide reasons for the false alarms. In the third step, participants and Lincoln Laboratory, along with participants, created a final list of false alarms with reasons after a careful analysis of background traffic and individual system characteristics. This required many interactions and much detailed analysis for all involved.

A high-scoring false alarm will be defined as any entry (putative attack detection) in the detection list file that does not correspond to a true attack and has a score above a threshold which provides 10 false alarms per day across all attack types in the detection list file returned by the participant.

8.2.2.2 Format of the False Alarm List File

The format of the false alarm list is similar to that of the miss list file except the "reason" field is simpler. The "reason" field contains one or two keywords separated by a comma with an optional reference pointer indicated by a bracketed number at the end of the "reason" field pointing to text at the end of the list. The following are examples of valid "reason" fields that could be used in false alarm list files.

1. bug, day
2. badlabel[1]
3. overlapfeatures, month
4. widesignature, week[2]
5. bug, week
6. normalpeak, week[3]

Using the standard UNIX command description conventions, the reason field should contain:

```
why-key, [fix-key]][[n]]
```

- why-key = bug or badlabel or overlapfeatures or widesignature or normalpeak or unknown
- fix-key = day or week or month or year or impossible
- n = a number referring to a comment after the end of the miss list

Questions and corresponding keyword answers are described in the next section.

8.2.2.3 Keywords and Questions

To assign reasons for false alarms in a relatively common format, it was necessary to define some keywords and outline the format for appending the reason for each false alarm to the appropriate line in the false alarm list text file. The format of the false alarm list file is similar to that of the miss list file except the first three keywords in the "reason" field are not required and only the last two keywords are necessary to specify why the false alarm occurred and how difficult it would be to eliminate this false alarm. The two keywords in the "reason" field of the miss list must address the following two questions. Each question results in one keyword in the "reason" field.

Q1. *Why did this false alarm occur?*

The intrusion detection system designers provided a "why" keyword by carefully analyzing their system and the background traffic that caused this false alarm. Some possible "why" answers along with keywords for these answers are listed here:

- *bug* - There was an implementation bug. When fixed, background traffic would not cause this false alarm.
- *badlabel* - The 1999 corpus was labeled incorrectly. This false alarm was triggered by a real attack and the Lincoln Laboratory truth labels should be fixed to label the traffic that triggered this false alarm as an attack.
- *overlapfeatures* - The features selected do not discriminate perfectly between normal traffic and attacks, but feature values for normal and attack traffic overlap.
- *widesignature* - The attack signature was too non-specific and/or wrong and incorrectly labels some normal traffic as attacks.
- *normalpeak* - A short term peak increase in normal traffic to one or more hosts looks like a denial of service or probe attack.
- *unknown* - This is a keyword that Lincoln Laboratory will put in to be replaced by the participant.

Participants could provide other keywords to indicate why the false alarm occurred. Any new keywords used by participants should be defined at the bottom of the false alarm file list file. In addition, they can provide a detailed footnote indicated by a bracketed number (e.g. [1]) with text at the bottom of the false alarm list file. After this keyword is provided, question 2 must be answered unless the keyword is badlabel.

Q2. *How hard would it be to extend the intrusion detection system to eliminate this false alarm?*

A final "hardness" keyword is used for participants to estimate how difficult it would be to modify their system to eliminate the false alarm.

- *day* - The modification is trivial and can be performed in minutes to hours. The modification can be made and tested in less than a day.
- *week* - The modification is more extensive and requires a few days.
- *month* - The modification is more extensive and requires a few weeks.
- *year* - The modifications required are extremely extensive and would require months of effort.
- *impossible* - The participant thinks that this false alarm could never be eliminated by anyone because it is too similar to attack traffic, or for some other reason.

8.2.2.4 Example of a False Alarm List File

The following is an example of a false alarm file created first by Lincoln Laboratory and then with reason keywords filled in by a participant. This file explains why false alarms occurred for traffic from the second week of the 1999 training data. The hypothetical system that reported these false alarms is designed to look for probes, denial-of-service, user-to-root, and secret attacks against all systems. The attack name information was provided in the detection list file only for some attacks, and often only the attack category was provided. The identification index in the false alarm is the index provided by the participant in the returned detection file. Note that the user in this false alarm file has provided two new keywords. One, "bsmbug" specifies a fictitious problem with the BSM audit system. A second keyword "tcpdumpbug" specifies a fictitious problem with the tcpdump program. This file also includes two footnotes.

ID	Date	Start_Time	Destination	Score	#	Attack	Reason
1	03/08/1999	09:15:01	hume.eyrie.af.mil	0.34	#	dos	normalpeak,week
2	03/08/1999	10:40:15	zeno.eyrie.af.mil	0.30	#	pod	widesignature, day[1]
6	03/08/1999	12:17:11	marx.eyrie.af.mil	0.23	#	secret	widesignature, week[2]
9	03/09/1999	17:15:45	pascal.eyrie.af.mil	0.56	#	mailbomb	normalpeak,week

ID	Date	Start_Time	Destination	Score	#	Attack	Reason
10	03/09/1999	19:06:43	marx.eyrie.af.mil	0.7	#	dos	normalpeak,week
11	03/09/1999	16:54:19	zeno.eyrie.af.mil	0.65	#	loadmodule	bug,day
16	03/09/1999	18:01:45	pascal.eyrie.af.mil	0.56	#	u2r	bsmbug
23	03/08/1999	11:25:01	hume.eyrie.af.mil	0.94	#	pod	tcpdumpbug

NEWKEYWORD: bsmbug - There is a bug in BSM auditing which makes it impossible to distinguish between attacks and normal usage of the eject Solaris system program because the path argument for the exec system call is not correct.

NEWKEYWORD: tcpdumpbug - There is a bug in the tcpdump program which improperly interprets fragmented icmp eco packets and makes it impossible to distinguish normal eco packets from attacks.

[1] A normal short ping was sent to zeno, and the signature used incorrectly labeled all pings to zeno as ping-of-death attacks.

[2] The program logic or signature was incorrectly set up to detect a secret attack when a secret file was viewed remotely using a secure shell connection, even though this is allowed by the security policy.

8.2.3 Feedback Results

The participant feedback process and analysis of missed detections and false alarms proved to be a very useful part of the 1999 intrusion detection evaluation. The feedback process led to a few changes in the scoring procedure, while the miss and false alarm analysis results will be discussed in the "Results" technical report.

Feedback from participants led to a couple of improvements in the scoring procedure. These improvements are reflected in the discussion of the scoring procedure in Section 8.1, however they are also summarized in the next two paragraphs.

Short-duration DoS attacks with long-lasting effects had initially been scored, similarly to other attacks, with correct detections only allowed in the window of one min. before and one min. after the duration of the attack actions themselves. However it became evident that at least one intrusion detection system was attempting to detect these types of attacks by detecting the denied connection requests. Rather than simply not count any of these detections, and as opposed to counting detections that could be hours after the attack itself, a fifteen-minute window after the initial attack was allowed for detection of some of these denied service requests. Also, the detailed analysis of "a few" attacks revealed instances in which attacks had been placed in the wrong categories or some attack session had been omitted.

8.3 IDENTIFICATION SCORING

In the identification phase of scoring, the procedure was easier than that of the detection phase. Participants, optionally, submitted a list of identification entries, each entry corresponding to a putative detection alert and having the same identification (ID) number. These were used to evaluate the ability of the intrusion detection system to provide information important to understanding and taking action to verify the putative alert and taking defensive action in the face of attack. These identification entries contained more detailed information about the putative detection alert, like start time, duration, source of the attack, etc, in the format:

```
ID:
Date:
Name:
Category:
Start_Time:
Duration:
Attacker:
Victim:
Ports:
  At_Attacker:
  At_Victim:
Username:
Comments:
```

Here are more details on the fields of the identification alert:

1. ID: Each entry must contain an ID# which must match the ID# of the corresponding entry in the detection alert list
2. Date of attack: As for the detection list, the format is MM/DD/YYYY. This is the date the attack begins.
3. Name of attack: Lincoln provided a list of all attack names that appeared in 1998 evaluation or in the 1999 training data (html file of attacks database). For attacks that are new to the 1999 test set (did not appear in the 1999 training data), the name provided in the identification list file was not scored
4. Category of attack: The five categories, dos, probe, u2r, r2l, data, were used to categorize attacks, allowing some attacks to be in more than one category when necessary.

5. Start time: (HH:MM:SS) The hypothesized start time of the attack. This start time was not necessarily identical to the start time provided in the detection list.
6. Duration (HH:MM:SS): this is the amount of time between the attack start and end times. If the attack has several stages (such as a setup, break-in, and actions), then the start time should be when the beginning of the setup occurs and the duration should encompass the full time period until all actions of the attacker are complete. The hour field could be larger than 24 if the attack extends over multiple days.
7. Source machine(s): Either IP addresses or full machine names including the domain were accepted. Multiple machines must be comma separated. The shorthand notation "x.y.z.(1-100)" refers to the 100 machines x.y.z.1, x.y.z.2, ..., x.y.z.100.
8. Destination machine(s): Same format as for source machines.
9. Destination port(s) and number of connections to each: Any well-known ports that are made use of during an attack are listed. In addition, any port that is the destination of a TCP connection, UDP packet, or ICMP packet is listed, unless it is an ftp-data connection. In the case of ftp-data, the well-known port number (20), at that machine on which that port is used in an attack-related connection, is listed.
   - The list of ports is classified according to weather they are ports on/at the attacking machine or the victim machine. For example, if there's a telnet from host "Attacker", port 12345, to host "Victim", port 23 (Attacker:12345 -> Victim:23), refer to port 23 on/at "Victim".
   - The shorthand notation 1-100 can refer to ports 1,2,3,...,100.
   - UDP connections are labeled as port#/u.
   - ICMP connections are labeled as "i". (See examples below).
   - The number of times a single port or a range of ports is listed (connected to) should be represented by the numeral representing number of repetitions, placed within curly braces, after the port or port-range. For Example: (1-100){5} indicates that ports 1, 2, 3, ..., 100 were each used 5 times, for a total of 500 connections. The list should be comma separated.
10. Comments - This field was optional and was not used to score the identification part of the evaluation.

Here is an example identification entry:

```
ID: 1
Date: 03/08/1999
Name: ntinfoscan
Category: probe
Start_Time: 08:01:01
Duration: 00:15:14
Attacker: 206.048.044.018
Victim: hume.eyrie.af.mil
Ports:
   At_Attacker:
   At_Victim: 21{1}, 20{4}, 23{1}, 80{10}, 139{2}
Username: anonymous
Comments:
```

This example entry refers to an NTInfoscan attack, an attack that probes the Windows NT machine for common vulnerabilities and lists of users. The attack started at 8:01am and lasted fifteen minutes. The attack was sourced from 206.48.44.18 and was directed against hume.eyrie.af.mil. During the attack the ftp (21), telnet (23), and http (80) ports on hume were connected to once each, a netbios (139) port was connected to twice, and the ftp-data (20) port was connected to four times. The username "anonymous" was used in an attempt to login to the ftp server anonymously.

This sort of alert format was adequate for the 1999 evaluation, however in the future a more flexible alert might lend itself better to an evaluation that included higher-level fusion and correlation intrusion detection systems. In those cases, it is important that intrusion detection alerts contain as much information as the intrusion detection system can gather. The static alert format of the 1999 evaluation does not adapt as easily to the requirements of the variety of intrusion detection systems. Use of a more flexible alert format, such as Intrusion Detection Message Exchange Format (IDMEF) [23] or Common Intrusion Specification Language (CISL) [24], would require an API for systems to be able to write alerts in similar formats. On the scoring and evaluation side, evaluators would need to ensure that comparable alerts are scored the same – to avoid the problem of alerts with the same content but differing in exact format being scored differently.

Scoring of these simpler 1999 identification alerts proceeded as follows. If an attack was correctly detected, by the intrusion detection system, according to the detection scoring process, then the highest scoring detection alert that matched the attack determines which submitted identification entry would be scored in the identification scoring phase. Only one identification list entry would be selected for each attack detected. The goal was to encourage intrusion detection systems to conglomerate as much as possible into one entry, which could then be referred to by multiple detection entries. This selected

identification entry was then compared, field-by-field with each of the fields of the Lincoln Identification Truth entry that corresponded to the attack instance.

A field-by-field comparison and presentation of results was made to be as fair as possible in the scoring. It would be desirable to give a score to the identification alert as a whole as well but this would require weighting the relative importance or value of individual fields. To do this, one would need to know the relative values of the fields to either a human administrator or to a higher-level detection or correlation device. Then one could produce a single score representing how well an intrusion detection system identified the attack instance. However, these values will vary depending on who or what is depending on the outputs, and an extensive study will be required to determine accepted costs and values for this sort of information.

Results for scoring of identification information are found in the "Results" technical report and can be found online in the Identification Scoring Reports, indexed at

http://ideval.ll.mit.edu/1999test/scoring_report_index.html.

Scoring for those individual fields that were scored is discussed in the following sections. The Category, Name, Start Time, Duration, Source (Attacker), and Ports (at/on the victim) fields were all scored. Attack Destination (Victim) and Ports at/on the attacker were not scored.

8.3.1 Attack Category

Identifying the attack category aids a system administrator in knowing the purpose of an attack. The five predefined attack categories were DoS, Probe, R2L, U2R, and Data. If the participating intrusion detection system specified the category of the attack that was detected then the Category field was marked correct.

Results for this field are given in a confusion matrix, such as the one shown in Figure 8-5. The first column of the matrix is the category assigned by the participant. The first row is the truth category. In the next five rows and columns, matrix entries give the number of times that an attack was identified as belonging to a particular category listed in that column when in fact it belonged to a category in that row. The last column gives the number of false alarms for each category. The seventh row in the confusion matrix gives the number of attacks that were not categorized by the participant. The last row represents the number of attacks that the participant missed.

TABLE 8-1

EXAMPLE OF THE CONFUSION MATRIX

	r2l	u2r	dos	probe	data	false_alarms
r2l	16	0	2	0	0	5
u2r	0	13	0	1	0	1
dos	2	0	41	3	0	4
probe	0	0	0	8	0	5
data	0	0	0	0	2	0
no_cat.	0	0	0	0	0	0
misses	22	10	19	24	4	0

The confusion matrix used to present results for attack category classification in the identification scoring procedure.

Identification entries that listed two categories for an attack were given the benefit of the doubt, and scored as correct if one of the categories listed was correct. If all categories listed were incorrect, the first listed category was used for the confusion matrix.

8.3.2 Attack Name

The ability of an intrusion detection system to accurately identify the attack name for old attacks helps a system administrator know exactly which specific attack was used. Installing patches or following CERT advisory recommendations may prevent further attacks of the same type.

Attack Name fields are scored if the attack that was detected is an "old," for which a name has previously been defined. This scoring was not performed for "new" attacks, for which names had not been defined ahead of time. A field is scored as correct if the name assigned by the intrusion detection system matches the truth. Some "normalization" was performed on submitted Identification files to account for minor formatting differences between the truth list and the submissions.

Results are given in a large confusion matrix. The first column indicates possible names listed by the participant. The first row indicates true names. Numbers along the upper left to lower right diagonal indicate correctly named attacks. New attacks that did not appear in the 1999 training data were not scored. The names dict, guest, guessftp, guesstelnet, and guesspop were considered to refer to the same attack type for this scoring, as they all refer to instances of password guessing of a fairly similar nature.

8.3.3 Detection Latency and Estimated Duration

The ability of an intrusion detection system to quickly detect the presence of an attack allows a system administrator to respond rapidly to an attack. Intrusion Detection Systems provided the time at

which their system detected each attack and a duration over which the attack was suspected to take place. For some systems, these times may represent the actual latencies that would be obtained in a real-time installation. For others (i.e. SRI-DERBI), these times are forensic, determined after the fact, because these systems do not run in real-time.

Detection Latency is scored by comparing the submitted "Start Time" with the True "Start Time" of the attack. Time latency results are presented in hours, minutes, and seconds, and also summarized in terms of a percentage of the attack duration. Start time of attacks was scored separately for DoS, Probes, R2L, and U2R attacks. Data attacks were not scored because of the low number of data attacks that were detected. Summary tables, illustrated in Table 8-2, are created that show the number of attacks detected at various latencies. The table shows the number of attacks in each category for which the detection latency falls in the ranges (in seconds) 0-15, 15-30, 30-45, 45-60, 60-75, and greater than 75. Identification of the Start Time was considered *correct* if the detection latency was 15 seconds or less.

TABLE 8-2

ACCURACY OF DETECTED ATTACK START TIMES

(sec)	r2l	u2r	dos	probe	data
0–15	14	0	36	4	0
15–30	2	0	2	0	0
30–45	0	0	1	0	0
45–60	1	1	1	1	0
60–75	0	0	1	0	0
>75	1	12	2	7	2

The accuracy of one particular intrusion detection system and its ability to identify the start time of detected attack instances. Similar tables for all evaluated systems are given in the "Results" technical report.

A similar scoring procedure and presentation of results was done for the Attack Duration field. The resulting table, illustrated in Figure 8-7, shows the number of attacks in each category for which the error in guessed duration falls in the ranges (in seconds) 0-15, 15-30, 30-45, 45-60, 60-75, and greater than 75.

TABLE 8-3

ACCURACY OF DETECTED ATTACK DURATIONS

(sec)	r2l	u2r	dos	probe	data
0–15	5	0	21	2	0
15–30	0	0	0	0	0
30–45	0	0	2	0	0
45–60	3	0	2	0	0
60–75	0	0	0	0	0
>75	10	13	18	10	2

This table shows the accuracy with which one particular intrusion detection system was able to identify the duration of detected attack instances.

8.3.4 Victim Ports

The ability of an intrusion detection system to accurately identify victim ports aids a system administrator in knowing what ports and services are being used or affected by an attack. Participating intrusion detection systems listed the TCP and UDP ports used in an attack and weather or not ICMP was used, also given was the number of times that each port or service was used. There was one list for ports and services used on the victim machine(s) and another list for ports and services used on the attacking machine. These lists were scored for the number of ports used that were identified correctly, and not penalized for listing "extra" ports. The number of connections per port is not scored.

For attacks in which fewer than 10 ports or services were used, all ports must have been correctly identified (listed) for that identification field to be scored as correct. For attacks with 10 or more ports used, 90% of the ports must have been correctly identified for the attack to be scored as correct. Results are given, in the Results Technical Report, in terms of the number of attacks that were scored correctly according to these criteria.

8.3.5 Source of Attack

The ability of an intrusion detection system to accurately identify the attacker sources aids a system administrator in detecting the origin of attacks. Participating intrusion detection systems listed the source IP address(es) of each attack. Comparing them to the true list of attack sources scored these lists of addresses. In this scoring, forged IP addresses are treated as true attacker source IP addresses that must be identified.

For attacks with less than 10 source machines, all machines must have correctly identified for the attack to be scored as correct. For attacks with 10 or greater source machines, 90% of the machines must have correctly identified for the attack to be scored as correct. Systems were not penalized for listing additional, incorrect source machines. The results technical report gives the number of attacks that were scored correctly according to these criteria.

9. EVALUATION PROCEDURE PLAN

This section discusses the timeline of the evaluation. The purpose is to give an idea of the how the evaluation proceeded for both the evaluations and participants. Figure 9-1 shows many of the tasks involved and who did what to make the evaluation possible. As is evident, the evaluation is expensive for both the developer of the intrusion detection system and for the evaluators. Developers must preparetheir systems, train them, run the evaluation, and work with the evaluators to verify results. These werevery time-consuming processes in the 1999 evaluation and need to be considered in designing future evaluations.

340008_50

	Preparation of IDSs	Preparation of Training Data	Running Training Data	Validation of Training Results	IDSs Tuning	Preparation of Test Data	Running Test Data	Packaging of Results	Processing of Results	Verification	Discussion
Developers	X		X	X	X		X	X		X	X
Evaluators		X		X		X			X	X	X

Figure 9-1. Breakdown of evaluation tasks for participants and evaluators. (adapted from Kemmerer, "Intrusion Detection Evaluation: A Victim's perspective," presented at the DARPA Intrusion Detection Evaluation Rethink Meeting, Lake Geneva, WI, May 2000.)

The remainder of this section describes in more detail the tasks carried out by the evaluators to create the evaluation. The timeline of the evaluation, from the evaluator's point of view, can be broken down as shown in Figure 9-2. A brief discussion of the major phases is included here:

- Configuring and Updating the Testbed and Background Traffic — Prior to the 1999 evaluation many features were added to the testbed environment, in terms of ease of use improvements and the addition of Windows NT, and to the background traffic, in terms of more detailed simulation for UNIX hosts and users and Windows NT traffic. Most of these changes are listed in sections 4 and 5.

- Generating and Distributing the Training Data (and Pretest) — This stage involved finalizing the design of the simulation network, the background traffic generation, and the attack generation. Major components of this phase were running the network testbed to generate fifteen 22-hour days of data, collecting the data, and verifying after each run that the simulation had been carried out correctly. Section 5, and in particular section 5.4, discuss many of the issues involved in

were generated and simply running 15 simulations days. Each day involved much setup, administration, data collection, and verification of the day's run. It was sometimes necessary to re-run a day when vital components of the simulation itself crashed or failed to execute properly. For distribution, the training data was posted on the evaluation's password protected website. The Pretest was an informal phase of the evaluation in which participants were asked to run one day of the training data, week 2, day 4, through their intrusion detection system and submit to us the resulting list of detections in the evaluation format. The goal was to ensure that participants and evaluators understood and agreed on the format for submitting results and to ensure that the evaluation had generated the data required by the intrusion detection systems.

- Generating and Distributing the Test Data — Generating the test data involved all aspects of generating the training data with the added overhead of attacks. Despite the fact that the attacks were often (though not always) carried out automatically with the help of expect-based scripts, each attack instance required a significant amount of effort to setup, test, run, and verify. Attacks are discussed in Section 6. It was not uncommon to have to re-run a day of the Test data when attack instances failed for any number of reasons, or if there were a breakdown of the background simulation itself. For distribution, the test data was posted on the evaluation website under a new and different password. Access to the test data was given only to those participants who had committed to running the evaluation and had submitted to Lincoln a formal description of their intrusion detection system.

- Running the Test Data (Participant task) — The test itself was a two week period in which participants would run the test data through their intrusion detection systems, in an off-line fashion, and collect the output of their intrusion detection system, convert it to the evaluation format, and send the results back to the evaluators. Test data was released and ready for download Friday, September 17, 1999, and participants had 14 days from when they had successfully downloaded the test data to return the results to us. In most instances, this meant that results were returned by Monday, October 4th, 1999, or within a couple days.

- Collecting and Scoring Results — Test results were emailed, as text files in the desired format, back to us. The next major evaluation phase was to score these results. There are, naturally, a number of facets to scoring. First, quite a bit of effort was extended in order to make sure that the "truth" against with submitted entries were scored was accurate as possible. Second, software to actually do the scoring and output results in usable report format needed to be written and tested. Third, submitted lists of alerts sometimes needed to be reformatted when they didn't conform to specifications since not all instantiations of the common format were identical. And finally, the evaluation needed to be scored. The scoring procedure is discussed in section 8.

- Analyzing and Presenting Scored Results — Once the scoring process was solidified and a preliminary scoring run was performed, it was time to begin analysis of the results. The first step was to package up each systems results and send them back to the participant so that they could verify that they agreed, as best as possible, with the truth and the way in which the scoring was carried out. This was an iterative process and required a lot of effort on the part of both participants and evaluators. Discussion of this participant feedback phase can be found in section 8.2.

- Principle Investigators Meeting — In December 1999, at the DARPA Intrusion Detection PI meeting, both evaluators and evaluation participants presented results of the evaluation to DARPA and the research community.
- Analysis with Real-Traffic — The final phase, or first phase of the year 2000 effort, is to collect real network data at Hanscom Air Force Base and perform analysis on it to determine how it compares to the background traffic currently being generated on the testbed. The first phase of this analysis was to attempt to characterize the Hanscom traffic with respect to types and distributions of network traffic seen on the base as well as to quantify any "oddities" like malformed packets, fragments, and other strange traffic seen in this real-world environment. The second phase is to run this real data through one or more of the DARPA research intrusion detection systems, with the goal of seeing what false alarms and false alarm rate these research systems see when run on real data as compared to the simulation data. Results of this traffic and false alarm analysis would be used to update the traffic generated on the testbed for future evaluation efforts.

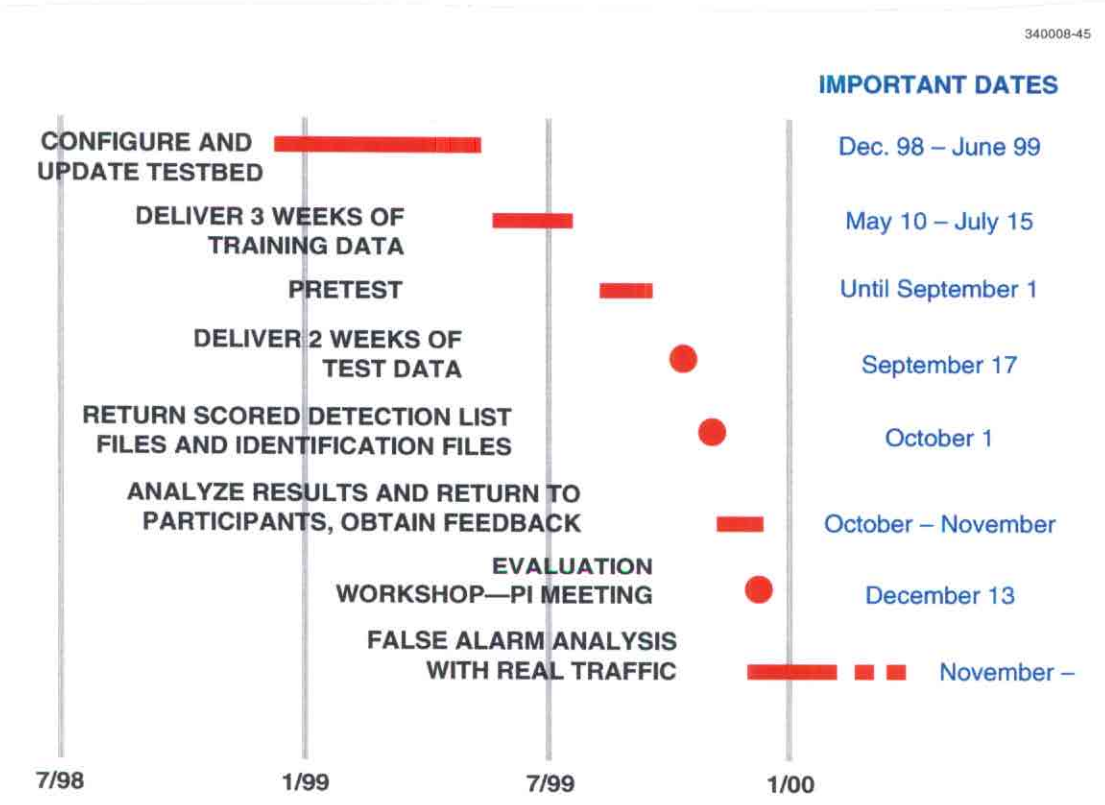


Figure 9-2. Time line for the 1999 evaluation

10. CONCLUSIONS

The 1999 DARPA off-line intrusion detection evaluation promoted development of improved intrusion detection systems by evaluating new intrusion detections systems in 1999 and providing a corpus of data for continued research. Eight research sites participated in this second annual evaluation. A network testbed generated live background traffic, similar to that on a government site, containing hundreds of users on thousands of hosts. More than 200 instances of 58 attack types were launched against victim UNIX and Windows NT hosts in three weeks of training data and two weeks of test data. This off-line data, along with similar data from the 1998 off-line evaluation, is the DARPA intrusion detection corpus. The 1999 evaluation supported algorithm development, performed a blind, off-line evaluation of intrusion detection approaches, and helped DARPA guide research directions. Results of the 1999 evaluation are discussed in a separate technical report, entitled, "Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation." In addition to supporting development of DARPA funded intrusion detection work, the corpus has been downloaded by over 100 sites and has been used to develop and test many intrusion detection techniques and systems.

Intrusion detection research is a field that must continuously change and evolve to keep up with new types of attacks and adversaries and the ever-changing internet environment. Similarly, an evaluation of intrusion detection systems must also evolve. The following are suggestions for updating the evaluation in the future.

First, the testbed, attacks, and adversary models should be updated to ensure that intrusion detection systems are being measured against current threats in a realistic network environment. The testbed and background traffic might be updated to model a modern network that approximates the environment in which intrusion detection systems are currently operating. This will make new types of attacks possible and also provide allow false alarms to be measured. Our experience with the 1998 and 1999 evaluations suggests that more effort should be placed on updating attacks and adversary models than on accurately modeling background traffic. It is relatively easy to measure false alarm rates of high-performing intrusion detection systems on actual internet traffic. It is not so easy to evaluate performance with actual attacks.

Second, some of the procedures used in the1999 evaluation could be updated to make the evaluation easier to run and the results easier to interpret. Many attacks that were run manually in 1999 could be automated for better repeatability of results and replication of experiments. Attacks could be instrumented as they run to automatically verify weather or not they succeed. The categorization of attacks could also be extended to form further subcategories that are fine enough to precisely specify which attacks each intrusion detection system should be able to detect.

Finally, the evaluation could be extended to allow participation from commercial and other real-time intrusion detection systems. Many intrusion detection systems enhance detection capability by incorporating probes of the system being defended. A "real-time" dataset to support real-time detection techniques could take the form of a small testbed and/or a series of scripts that could carry out the attacks and create background traffic.

REFERENCES

1. J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner, "State of the Practice of Intrusion Detection Technologies," Technical Report, Carnige Mellon University, CMU/SEI-99-TR-028, ESC-99-028, January 2000.
2. M. Bishop, S. Cheung, et al., "The Threat from the Net," *IEEE Spectrum*, 38(8), 1997.
3. E. Amoroso, Intrusion Detection, Intrusion.Net Books, 1999.
4. S. Northcutt, Network Intrusion Detection: An Analysis Handbook, New Riders Publishing, Indianapolis, 1999.
5. J. Mariani, "Evaluation," In R.A. Cole, J. Mariani, H. Uszkoriet, A. Zaenen, and V. Zue (eds.), *Survey of the State of the Art in Human Language Technology*, Cambridge University Press, 1997.
6. N. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R.A. Olsson, "A Methodology for Testing Intrusion Detection Systems," *IEEE Transactions on Software Engineering*, 22, 1996, pp. 719–729.
7. C. Ko, G. Fink, and K. Levitt, "Execution Monitoring of Security-critgical Programs in Distributed Systems: A Specification-based Approach," in *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, 1997, pp. 134–144.
8. G. Shipley, "ISS RealSecure Pushes Past Newer IDS Players," *Network Computing*, 17 May 1999, HTTP://www.networkcomputing.com/1010/1010rl.html.
9. R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClurg, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, and M.A. Zissman. "Evaluating Intrusion Detection Systems: the 1998 DARPA Off-Line Intrusion Detection Evaluation," in Proceedings of the 2000 DARPA Information Survivability Conference and Exposition, Vol. 2, 2000.
10. R. Durst, T. Champion, B. Witten, E. Miller, and L. Spagnuolo, "Testing and Evaluating Computer Intrusion Detection Systems," *Communications of the ACM*, 42(7), 1999, pp. 53–61.
11. R.P. Lippmann and R.K. Cunningham, "Improving Intrusion Detection Performance Using Keyword Selection and Neural Networks," Computer Networks, 34(4), 597–603, 2000.
12. R. Lippmann, J.W. Haines, D.J. Fried, J. Korba, and K. Das "Analysis and Results of the 1999 DARPA Off-Line Intreusion Detection Evaluation" in Recent Advances in Intrusion Detection pp. 162–182 (eds.) H. Deber, Me, s. Wo, Springs, Berlin 2000.
13. R. Lippmann, J.W. Haines, D.J. Fried, J. Korba, and K. Das "The 1999 DARPA Off-Line Intrusion Detection Evaluation," Computer Networks, 34(4), 579–595, 2000.
14. G. Schudel and B. Wood, "Modeling Behavior of the Cyber-Terrorist," 2000 IEEE Symposium on Security & Privacy.
15. http://www.schafercorp-ballston.com/id sia (email pwalczak@schafercorp-ballston.com for access).

16. K. Kendall, "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems," Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1998.

17. J. Korba, "Windows NT Attacks for the Evaluation of Intrusion Detection Systems," Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2000.

18. K. Das, "Attack Development for Intrusion Detection Evaluation" Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2000.

19. S. Webster, "The Development and Analysis of Intrusion Detection Algorithms," Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1998.

20. http://www.ietf.org/html.charters/idwg-charter.html, Website of the Internet Engineering Task Force's (IETF) Intrusion Detection Working Group (IDEG).

21. http://www.isi.edu/gost/brian/cidf/ Website for the Common Intrusion Detection Format (CIDF) and Common Intrusion Specification Language (CISL).

22. M. Tyson, P. Berry, N. Williams, D. Moran, D. Blei, "DERBI: Diagnosis, Explanation and Recovery from computer Break-Ins," project described in http://www.ai.sri.com/-derbi/, April 2000.

23. Internet Engineering Task Force, Intrusion Detection Working Group, http://www.ietf.org/ html.charters/idwg-charter.html.

24. Common Intrusion Detection Framework, http://www.isi.edu/~brian/cidf.

APPENDIX A
COMPLETE ATTACK DESCRIPTIONS

1. LINCOLN LABORATORY DATABASE OF ATTACKS

The database consists of attacks used from three evaluation sources: the 1998 ID evaluation, the 1999 ID evaluation training data, and the 1999 ID evaluation test data.

Attacks considered to be "new" in 1999 are those that did not appear in 1998 or the 1999 training data and are denoted as such. Many of these attack descriptions are taken directly from Kris Kendall's MIT Master's thesis [A-16]. There are many references included in the text, which we have included at the end of this section. For more details on the notation and terminology used below, please refer to the thesis. Because the text of these attack descriptions has been taken directly from this thesis, some will contain references directly to the 1998 evaluation. It should be noted that these descriptions apply to the attacks as they were run in the 1999 evaluation as well.

Descriptions of Windows NT attacks were written by Jonathan Korba, and many more details are available in his thesis [A-17]. Discussion of the queso, selfping, and ncftp attacks are from Kumar Das's thesis [A-18], and additional details can be found there as well.

1.1 Contents

- Teardrop
- Udpstorm

2. User-to-Root Attacks
   - anypw (New in 1999 test) (Windows NT)
   - casesen (New in 1999 test) (Windows NT)
   - Eject
   - Ffbconfig
   - Fdformat
   - Loadmodule
   - ntfsdos (New in 1999 test) (Windows NT)
   - Perl
   - Ps
   - sechole (New in 1999 test) (Windows NT)
   - Xterm
   - yaga (New in 1999 test) (Windows NT)
3. Remote-to-Local Attacks
   - Dictionary
   - Ftpwrite
   - Guest
   - Httptunnel
   - Imap
   - Named
   - ncftp (New in 1999 test)
   - netbus (New in 1999 test) (Windows NT)
   - netcat (New in 1999 test) (Windows NT)
   - Phf
   - ppmacro (New in 1999 test) (Windows NT)
   - Sendmail
   - sshtrojan (New in 1999 test)

- Xlock
- Xsnoop

4. Probes
   - insidesniffer (New in 1999 test)
   - Ipsweep
   - ls_domain (New in 1999 test)
   - Mscan
   - NTinfoscan (Windows NT)
   - Nmap
   - queso (New in 1999 test)
   - resetscan (New in 1999 test)
   - Saint
   - Satan
5. Data
   - Secret

1.2 Denial-of-Service Attacks

A denial-of-service (DoS) attack is one in which the attacker makes some computing or memory resource too busy or too full to handle legitimate requests, or denies legitimate users access to a machine. There are many varieties of DoS attacks. Some, like a mailbomb, neptune, or Smurf, abuse a perfectly legitimate feature. Other DoS attacks, like teardrop and Ping of Death, create malformed packets that confuse the TCP/IP stack of the machine that is trying to reconstruct the packet. Still others, like apache2, back, syslogd, take advantage of bugs in a particular network daemon.

The following sections describe in detail each of the DoS attacks included in the 1998 and 1999 DARPA intrusion detection evaluation.

Apache2

R-a-Deny (Temp./Admin.)

Description: Apache2 is a DoS attack against an apache web server where a client sends a request with many http headers. If the server receives many of these requests, it will slow down and may eventually crash [A-4].

Simulation details: Apache2 was adapted from C code originally posted to the bugtraq mailing list. A C-shell wrapper was also created which executes the Apache2 C program in a loop until the server being attacked is no longer responsive. As soon as the attack was launched the load average (as reported by the 'top' program) of the victim server jumped to 5 or more. As more and more requests were submitted to the web server, the memory usage and load average of the victim continued to climb until eventually the httpd daemon ran out of memory and crashed. At this point the server no longer responded to http requests and the httpd daemon needed to be restarted by the superuser for service to be restored.

Attack signature: Every http request submitted as part of this attack contains many http headers. Although the exact number and value of these headers could be varied by an attacker, the particular version of Apache2 which was used in the 1998 DARPA evaluation sent http GET requests with the header "User-Agent: sioux\r\n" repeated 10,000 times in each request. The actual content of the header is not important because the exploit is only dependent on the fact that the http request contains many headers. A typical http request contains twenty or fewer headers, so the 10,000 headers used by this exploit are quite anomalous.

arppoison

P-a-Deny (Temp.)

Description: ARP Poison is a DoS attack developed specifically for the 1999 MIT Lincoln Laboratory evaluation. In this attack the goal is to trick hosts on the same ethernet into "learning" the wrong "Mac" address for known IP addresses. The attacker must have access to the victim's Local Area Network.

Simulation details: Arppoison is carried out by running a binary program that takes the victim IP address as a commandline parameter. While running (at root-level) the attack listens on the network for "arp who-has" requests, to which it responds as quickly as possible with a bogus machine-level address. Thus, packets destined for the victim are "re-routed" at the Mac-level and connections/packets to the victim are disrupted. As more than one machine might respond to a given arp request, the attack is not 100% effective. During any attack it is possible that one or more hosts on the ethernet have cached the correct Mac address for a given IP and are not tricked into storing the bogus one.

During the 1999 evaluation, the attacks were manually verified to ensure that at least a few connections to the victim were disrupted as a result.

Attack signature: This attack could be detected by analyzing the ARP protocol, and observing that for a given "arp who-has" request the machine performing the attack consistently responds with the wrong (often completely bogus) machine-level address.

Back

R-a-Deny(Temporary)

Description: Back is a DoS attack against the Apache web server in which the attacker submits requests with URLs containing many frontslashes. As the server tries to process these requests, it will slow down and becomes unable to process other requests [A-55].

Simulation details: Back was implemented as a C shell script that used the Netcat [A-31] tool to generate network traffic. This shell script was adapted from a script originally posted to the Bugtraq mailing list. Although the number of frontslashes in the URL sent by the shell script could be varied, the number of frontslashes that was determined to be optimal for denial of service against Apache running on Linux 4.2 was between six and seven thousand. This attack causes instances of the httpd process on the victim to consume excessive CPU time. This consumption of the CPU slows down all the system's activities, including responses to network requests. The system recovers automatically when the attack stops.

Attack signature: An intrusion detection system looking for the Back attack needs to know that requests for documents with more than some number of frontslashes in the URL should be considered an attack. Certainly, a request with 100 frontslashes in the URL would be highly irregular on most systems. This threshold could be varied to find the desired balance between detection rate and false alarm rate.

Crashiis

R-b-Deny

Description: CrashIIS is a DoS attack against the NT IIS web server. The attacker sends a malformed GET request via telnet to port 80 on the NT victim. The command "GET ../.." crashes the web server and sometimes crashes the ftp and gopher daemons as well, because they are part of IIS. [A-70]

Simulation details: A fully automated Expect script on a UNIX attacker telnets to port 80 on the NT victim and sends the "GET ../.." command. Running "crashiis.exp 172.16.112.100" will crash Hume's webserver (and possibly ftp and gopher as well).

Verification: After the attack has completed, the IIS Web server on the victim should be down. Type the command "telnet hume.eyrie.af.mil 80"—it should no longer connect. Use a browser to access a page on the server (e.g., http://hume.eyrie.af.mil)—it should not load the page.

Cleanup: An administrator must manually restart the victim's web server. The ftp and gopher services usually need to be restarted as well.

Attack signature: Sniffing the network traffic will reveal the malformed GET command. The victim's security audit log will show that Dr. Watson ran when the service(s) crashed. However, Dr. Watson will also run for other reasons. Therefore, using this audit signature for detection will most likely result in false alarms.

dosnuke

R-b-Deny
(Temp./Admin.)

Description: DoSNuke is a DoS attack that sends Out Of Band data (MSG_OOB) to port 139 (NetBIOS), crashing the NT victim (bluescreens the machine) [A-68,69].

Simulation details: A Perl script, dosnuke.pl, runs on an NT attacker. Open dosnuke.pl for editing and set the time of day to run the attack. Then run dosnuke.pl or place it in the startup group. The script takes no arguments (always attacks Hume 172.16.112.100).

Verification: The victim machine will display a bluescreen. To remotely verify the success of the attack, try pinging 172.16.112.100. There should be no response from the victim.

Attack signature: The dosnuke attack creates a NetBIOS connection. The packets are flagged "urg" because of the MSG_OOB flag. The attack can be detected by searching the sniffed data for a NetBIOS handshake followed by NetBIOS packets with the "urg" flag.

The victim's audit logs will indicate a reboot after the system is restarted. Most likely, the attacker had to hard reboot the machine (physically press the reset button) because he did not have a password to login or unlock the machine. A soft reboot audit signature is a "SeShutPrivilege" Privilege Use Event followed by an event stating "Windows NT is starting up." A hard reboot audit signature does not include the "SeShutdownPrivilege" event. A hard reboot can be used to detect but not identify the Dosnuke attack, because other attacks may also result in hard reboots (NTFSDOS, AnyPW, etc.). In addition, a hard reboot may occur without an attack (power outages, system halts, etc).

Note: Originally, the attack sent the string "Hey, I can't help getting these nasty VXD errors!" It now sends a blank string instead. Other versions of the attack may still send the string.

Land

R-b-Deny
(Administrative)

Description: Land is a DoS attack that is effective against some older TCP/IP implementations. The only vulnerable platform used in the 1998 DARPA evaluation was SunOS 4.1. The Land attack occurs when an attacker sends a spoofed SYN packet in which the source address is the same as the destination address [A-17].

Simulation details: The Land attack program used in the DARPA evaluation was adapted from a C implementation found at http://www.rootshell.com. This exploit is quite simple and the code could easily be rewritten in any language with access to the TCP sockets interface. The code sends a single SYN packet with the source address spoofed to be the same as the destination address. Within the simulation, this exploit was run against a Sun SPARC workstation running SunOS version 4.1. When a TCP SYN packet with an identical source and destination address was received by this host, the system completely locked up. In order to restore service, the machine had to be physically turned off and on again.

Attack signature: The Land attack is recognizable because IP packets with identical source and destination addresses should never exist on a properly working network.

Mailbomb

R-a-Deny
(Administrative)

Description: Mailbomb is an attack in which the attacker sends many messages to a server, overflowing that server's mail queue and possibly causing system failure.

Simulation details: This attack was implemented as a Perl program that constructed mail messages and connected to the SMTP port of the victim machine directly. The Mailbomb Perl program accepted the e-mail addresses of victims as well as the number of e-mail messages to send as parameters. Although the Mailbomb exploit was used several times throughout the simulation with different parameters, a typical attack would send 10,000 one kilobyte messages (10 megabytes of total data) to a single user. This volume of messages was not enough to adversely effect the performance of the server or cause system failure. As implemented, this attack was more of a nuisance for a particular user than a real threat to the overall security of a server.

Attack signature: An intrusion detection system looking for a Mailbomb attack can look for thousands of mail messages coming from or sent to a particular user within a short period of time. This identification is a somewhat subjective process. Each site might have a different definition of how many e-mail messages can be sent by one user or to one user before the messages are considered to be part of a Mailbomb.

SYN Flood (Neptune)

R-a-Deny(Temporary)

Description: SYN Flood is a DoS attack to which every TCP/IP implementation is vulnerable (to some degree). Each half-open TCP connection made to a machine causes the 'tcpd' server to add a record to the data structure that stores information describing all pending connections. This data structure is of finite size, and it can be made to overflow by intentionally creating too many partially-open connections. The half-open connections data structure on the victim server system will eventually fill and the system will be unable to accept any new incoming connections until the table is emptied out. Normally there is a timeout associated with a pending connection, so the half-open connections will eventually expire and the victim server system will recover. However, the attacking system can simply continue sending IP-spoofed packets requesting new connections faster than the victim system can expire the pending connections. In some cases, the system may exhaust memory, crash, or be rendered otherwise inoperative [A-13].

Simulation details: The Neptune attack code used in the simulation was compiled from C code originally posted to the bugtraq archive. Neptune allows the user to specify a victim host, the source address to use in the spoofed packets, the number of packets to send, and the ports to hit on the victim machine (including an 'infinity' option that would attack all ports). The Neptune exploit was effective against all three of the victim machines used in the simulation. Every TCP/IP implementation is vulnerable to this attack to a varying degree depending on the size of the data structure used to store incoming connections and the timeout value associated with half-open connections. As a point of reference, sending twenty SYN packets to a port on a Solaris 2.6 system will cause that port to drop incoming requests for approximately ten minutes. During the simulation, a Neptune attack which sent 20 SYN packets to every port from 1 to 1024 of the Solaris server once every ten minutes was able block incoming connections to any of these ports for more than an hour.

Attack signature: A Neptune attack can be distinguished from normal network traffic by looking for a number of simultaneous SYN packets destined for a particular machine that are coming from an unreachable host. A host-based intrusion detection system can monitor the size of the tcpd connection data structure and alert a user if this data structure nears its size limit.

Ping of Death

R-b-Deny(Temporary)

Description: Ping of Death is a DoS attack that affects many older operating systems. Although the adverse effects of a Ping of Death could not be duplicated on any victim systems used in the 1998 DARPA evaluation, it has been widely reported that some systems will react in an unpredictable fashion when receiving oversized IP packets. Possible reactions include crashing, freezing, and rebooting [A-14].

Simulation details: Several implementations of the Ping of Death exploit can be found at http://www.rootshell.com as well as many other sources on the web. This exploit is popular because early versions of the ping program distributed with Microsoft Windows95 would allow the user to create oversize ping packets simply by specifying a parameter at the command line (i.e., ping °l 65510). Thus, many users could potentially exploit this bug without even making the effort to download and compile a program. The Ping of Death attack affected none of the victim systems used in the evaluation. The attack was included as an example of an attempted known attack that fails to have an effect.

Attack signature: An attempted Ping of Death can be identified by noting the size of all ICMP packets and flagging those that are longer than 64000 bytes.

Process Table

R-a-Deny(Temporary)

Description: Process Table is a novel DoS attack specifically created for this evaluation. It can be waged against numerous network services on a variety of different UNIX systems. The attack is launched against network services which fork() or otherwise allocate a new process for each incoming TCP/IP connection. Although the standard UNIX operating system places limits on the number of processes that any one user may launch, there are no limits on the number of processes that the superuser can create, other than the hard limits imposed by the operating system. Since incoming TCP/IP connections are usually handled by servers that run as root, it is possible to completely fill a target machine's process table with multiple instantiations of network servers. Properly executed, this attack prevents any other command from being executed on the target machine. An example of a service that is vulnerable to this attack is the finger service. On most computers, finger is launched by inetd. The authors of inetd placed several checks into the program's source code that must be bypassed in order to initiate a successful process attack. In a typical implementation (specifics will vary depending on the actual UNIX version used), if inetd receives more than 40 connections to a particular service within one minute, that service is disabled for 10 minutes. The purpose of these checks was not to protect the server against a process table attack, but to protect the server against buggy code that might create many connections in rapid-fire sequence. To launch a successful process table attack against a computer running inetd and finger, the following sequence may be followed:

1. Open a connection to the target's finger port.
2. Wait for 4 seconds.
3. Repeat steps 1 and 2.

This attack has been attempted against a variety of network services on a variety of operating systems. It is believed that the imap and sendmail servers are vulnerable. Most imap server software contains no checks for rapid-fire connections. Thus, it is possible to shut down a computer by opening multiple connections to the imap server in rapid succession. The situation is reversed with sendmail. Normally, sendmail will not accept connections after the system load has jumped above a predefined level. Thus, to initiate a successful sendmail attack it is necessary to open the connections very slowly, so that the process table keeps growing in size while the system load remains more or less constant [A-6].

Simulation details: The version of this exploit used in the simulation was implemented as a Perl script that would open connections to a port every n seconds, where the port and the number of seconds n are specified as run-time parameters. The connections were maintained until a user terminated the script. The number of connections that must be opened before denial of service is accomplished depends on the size of the process table in the operating system of the victim machine. By using the Process Table attack on the finger port as described above, the process table of a Solaris server was exhausted after opening approximately 200 connections. (At a rate of one connection every four seconds it took about 14 minutes before the process table was full). After the process table was full, no new process could be launched on the victim machine until the attack was terminated by the attacking program or an administrator manually killed the connections initiated by the attacking script (which is quite difficult to do without launching a new process).

Attack signature: Because this attack consists of abuse of a perfectly legal action, an intrusion detection system trying to detect a Process Table attack will need to use somewhat subjective criteria for identifying the attack. The only clue that such an attack is occurring is an unusually large number of connections active on a particular port. Unfortunately an 'unusual' number of connections is different for every host, but for most machines, hundreds of connections to the finger port would certainly constitute unusual activity.

Selfping

R-b-Deny
(Temporary)

Description: Selfping is a DoS attack in which a normal user can remotely reboot a machine with a single ping command. This attack can be performed on Solaris 2.5 and 2.5.1.
http://www.rootshell. com/archive-j457nxiqi3gq59dv/199707/solaris_ping.txt.html

Simulation details: The ping command broadcasts echo_request packets using the localhost as the multicast interface. Within a couple seconds the system panics and reboots. There are two versions of this attack in the 1999 DARPA evaluation. One version creates an atjob on the victim machine and then logs outs. The other, more malicious version, creates a cronjob which reboots the machine every 5 minutes. The administrator must remove the cronjob in order to keep the machine from rebooting. This attack was originally posted on http://www.rootshell.com

Attack signature: The attack reboots the machine within 10 seconds of executing the ping command. The only signature seen in sniffing data is a ping to the broadcast interface just before the machine dies. A ping to the broadcast interface is a perfectly normal action. In the less harmful version, the atjob can be seen until it executes, whereas in the malicious version, the cronjob remains until it is deleted.

Smurf

R-a-Deny
(Temporary)

Description: In the Smurf attack, attackers use ICMP echo request packets directed to IP broadcast addresses from remote locations to create a DoS attack. There are three parties in these attacks: the attacker, the intermediary, and the victim (note that the intermediary can also be a victim) [A-18]. The attacker sends ICMP 'echo request' packets to the broadcast address (xxx.xxx.xxx.255) of many subnets with the source address spoofed to be that of the intended victim. Any machines that are listening on these subnets will respond by sending ICMP 'echo reply' packets to the victim. The Smurf attack is effective because the attacker is able to use broadcast addresses to amplify what would otherwise be a rather innocuous ping flood. In the best case (from an attacker's point of view), the attacker can flood a victim with a volume of packets 255 times as great in magnitude as the attacker would be able to achieve without such amplification. This amplification effect is illustrated by Figure 6-2. The attacking machine (located on the left of the figure) sends a single spoofed packet to the broadcast address of some network, and every machine that is located on that network responds by sending a packet to the victim machine. Because there can be as many as 255 machines on an ethernet segment, the attacker can use this amplification to generate a flood of ping packets 255 times as great in size (in the best case) as would otherwise be possible. This figure is a simplification of the Smurf attack. In an actual attack, the attacker sends a stream of icmp 'ECHO' requests to the broadcast address of many subnets, resulting in a large, continuous stream of 'ECHO' replies that flood the victim.

Simulation details: Because the simulation network for the DARPA off-line evaluations has a flat network topology with only two physical subnets, the Smurf attack as described above could not be implemented on the simulation network. For this reason, the 'smurfsim' program was developed to recreate the observable effects of a Smurf attack. Smurfsim uses the raw socket API to construct ICMP packets with forged source addresses. Smurfsim takes as parameters the IP address of the victim, the number of packets to send, the average percentage of hosts on a subnet that are alive, and a comma-separated list of subnets. The program then randomly constructs a list of hosts that are alive on each of the subnets in the comma-separated list and starts sending 'echo reply' packets to the victim, that have been spoofed to look like they originating from the hosts in the list. This behavior is exactly what would occur if an attacker had performed an actual Smurf attack in which 'echo request' packets (with the source address spoofed to be that of the victim machine) were sent to the broadcast address of each subnet given in the parameter list. Several different simulated Smurf attacks were included in the evaluation data. In the most extreme case, the smurfsim program was used to simulate a Smurf attack that generated traffic from 100 subnets for a period of one

hour. During this period of time the entire simulation network was unresponsive and other network sessions (such as normal users trying to send e-mail, etc.) would time-out before they could be completed. In all, this particular attack instance generated over two gigabytes of network packets.

Attack signature: The Smurf attack can be identified by an intrusion detection system that notices there are a large number of 'echo replies' being sent to a particular victim machine from many different places, but no 'echo requests' originating from the victim machine.

sshprocesstable

R-a-Deny
(Temporary)

Description: SSHProcesstable is similar to the Processtable attack in that the goal of the attacker is to cause sshd daemon on the victim to fork so many children that the victim can spawn no more processes. This is due to a kernel limit on the number of processes that the OS will allow.

Simulation details: The attack is in binary form, and takes the victim IP as a commandline argument. The attack works by making a number (serveral hundred, commonly) of connections to the victim via ssh, but does not complete the login process. If the sshd login timeout is long enough and the connections happen rapidly enough, the hosts process table will be exausted and no new process will be able to be spawned until others timeout or exit.

Attack signature: This attack will be evident due to the large number of rapid ssh connections to the host, the inability of processes to spawn on the host, and the fact that request for new network logins (requiring child processes) will be denied, for the duration of the attack. There may be other obvious signs as well.

Syslogd

R-b-Deny
(Administrative)

Description: Syslogd is a DoS attack that allows an attacker to remotely kill the syslogd service on a Solaris server. When Solaris syslogd receives an external message it attempts to do a DNS lookup on the source IP address. If this IP address doesn't match a valid DNS record, then syslogd will crash with a Segmentation Fault [A-54].

Simulation details: The Syslogd exploit used in the DARPA off-line evaluations was a C program that was originally posted to the Bugtraq mailing list. This code was compiled to create an exploit program that was used to remotely cause the syslogd program to crash on a Solaris 2.5 server. Once syslogd has crashed it must be manually restarted by an administrator for the logging service to be restored.

Attack signature: The one way to reliably recognize this attack with a network-monitoring intrusion detection system is to notice a packet destined for the syslog port that contains an unreachable source address. Of course, it may not be realistic for an intrusion detection system to check every packet destined for the syslog port to see whether or not the source address is resolvable. If no remote system logging is expected to occur on a particular network, any external syslog messages appearing on this network is likely to be an attack. Finally, a host-based intrusion detection system could be configured to notice the syslog process die because of a segmentation fault.

tcpreset

R-a-Deny
(Temporary)

Description: TCP Reset is a DoS attack that disrupts TCP connections made to the victim machine. That is, the attacker listens (on a local or wide-area network) for tcp connections to the victim and sends a spoofed tcp RESET packet to the victim, thus causing the victim to inadvertently terminate the TCP connection.

Simulation details: This attack exists in the form of a binary written especially for the 1999 evaluation. The binary, "coneos," takes the victim IP as a commandline parameter and must be run with root-level permissions on the attacking machine (in order to listen for connections to the victim and do the spoofing).

Attack signature: One way to detect the attack would be to look at the TCP session setup/takedown process, and note cases in which RESET packets appear to come from the machine that had initially attempted to begin the connection. (This method may not be foolproof because there might be cases when this is a common/normal occurrence.)

Unlike the arppoison attack, tcpreset was 100% effective in resetting (causing termination of) tcp connections as they were being brought up. (In the 1999 evaluation.)

Teardrop

R-a-Deny
(Temporary)

Description: Teardrop is a DoS attack that exploits a flaw in the implementation of older TCP/IP stacks. Some implementations of the IP fragmentation re-assembly code on these platforms do not properly handle overlapping IP fragments [A-17].

Simulation details: The teardrop name is derived from a widely available C program that exploits this vulnerability. This exploit code can be found at http://www.rootshell.com and in the Bugtraq archives. Although many systems are rumored to be vulnerable to the teardrop attack, of the systems used in the DARPA evaluation, only the Redhat Linux 4.2 systems were vulnerable. The teardrop attack would cause these machines to reboot.

Attack signature: An intrusion detection system can find this attack by looking for two specially fragmented IP datagrams. The first datagram is a 0 offset fragment with a payload of size N, with the MF bit on (the data content of the packet is irrelevant). The second datagram is the last fragment (MF == 0), with a positive offset greater than N and with a payload of size less than N [A-5].

Udpstorm

R-a-Deny
(Administrative)

Description: Udpstorm is a DoS attack that causes network congestion and slowdown. When a connection is established between two UDP services, each of which produces output, these two services can produce a very high number of packets that may lead to denial of service on the machine(s) where the services are offered. Anyone with network connectivity can launch an attack; no account access is needed. For example, by connecting a host's chargen service to the echo service on the same or another machine, all affected machines may be effectively taken out of service because of the excessively high number of packets produced. An illustration of such an attack is presented in Figure 6-2. The figure demonstrates how an attacker is able to create a never-ending stream of packets between the echo ports of two victims by sending a single spoofed packet. First, the attacker forges a single packet that has been spoofed to look like it is coming from the echo port on the first victim machine and sends it to the second victim. The echo service blindly responds to any request it receives by simply echoing the data of the request back to the machine and port that sent the echo request, so when the victim receives this spoofed packet it sends a response to the echo port of the second victim. This second victim responds in like kind, and the loop of traffic continues until it is stopped by intervention from an external source [A-10].

Simulation details: Code that exploits this vulnerability was posted to the bugtraq mailing list. This program sends a single spoofed UDP packet to a host. This single spoofed packet is able to create a never-ending stream of data being sent from the echo port of one machine to the echo port of another. This loop created network congestion and slowdown that would continue until the inetd daemon was restarted on one of two victim machines.

Attack signature: This attack can be identified in two ways. First, the single packet that initiates the attack can be recognized because it is a packet originating from outside the network that has been spoofed to appear as if it is coming from a machine inside the network. Second, once the loop of network traffic has been initiated, an intrusion detection system that can see network traffic on the inside of the network can note that traffic is being sent from the chargen or echo port of one machine to the chargen or echo port of another.

1.3 User-to-Root Attacks

User-to-root attacks refer to an exploit in which the attacker starts out with access to a normal user account on the system (perhaps gained by sniffing passwords, a dictionary attack, or social engineering) and is able to exploit some vulnerability to gain root access to the system. There are several different types of user-to-root attacks. The most common is the buffer overflow attack. Buffer overflows occur when a program copies too much data into a static buffer without checking to make sure that the data will fit. For example, if a program expects the user to input the user's first name, the programmer must decide how many characters that first name buffer will require. Assume the program allocates 20 characters for the first name buffer. Now, suppose the user's first name has 35 characters. The last 15 characters will overflow the name buffer. When this overflow occurs, the last 15 characters are placed on the stack, overwriting the next set of instructions to be executed. By carefully manipulating the data that overflows onto the stack, an attacker can cause arbitrary commands to be executed by the operating system. Although programmers may eliminate this problem through careful programming techniques, some common utilities are susceptible to buffer overflow attacks [A-2]. Another class of user-to-root attacks exploits programs that make assumptions about the environment in which they are running. A good example of such an attack is the loadmodule attack, which is discussed below. Other user-to-root attacks take advantage of programs that are not careful about the way they manage temporary files. Also, some user-to-root vulnerabilities exist because of an exploitable race condition in the actions of a single program, or two or more programs running simultaneously [A-27]. Although careful programming may eliminate all of these vulnerabilities, bugs like these are present in every major version of UNIX and Microsoft Windows available today. The following sections describe each of the user-to-root attacks used in the 1998 and 1999 DARPA intrusion detection evaluation in greater detail.

anypw

U--S

Description: NukePW is a console user-to-root attack that allows the attacker to logon to the system without a password. A boot disk is used to modify the NT authentication package so that a valid username can login with any password string. Logins via telnet also work with any password.

Simulation details: Insert the bootdisk containing the attack installed in the boot sector. Reboot the machine. A hexadecimal number will appear in the upper left of the screen. The number will slowly increment as the attack searches for the signature of the authentication package. Wait until an asterisk appears beside the number. Remove the diskette and reboot. Later, telnet to the machine as an administrator and enter any password to log on.

Verification: Any password will be accepted with a valid username.

Cleanup: The file c:\winnt\system32\msv1_0.dll must be replaced with an uncorrupted copy.

Attack signature: The sniffed data will reveal remote logons with incorrect password strings. If the attacker physically logs on to the machine and then locks the machine, only the password used to log on can unlock the machine.

The victim's security audit log will indicate a reboot after the system is restarted. Most likely, the attacker had to hard reboot the machine (physically press the reset button) because he did not have a password to log in or unlock the machine. A soft reboot audit signature is a "SeShutPrivilege" Privilege Use Event followed by an event stating, "Windows NT is starting up." A hard reboot audit signature does not include the "SeShutdownPrivilege" event. A hard reboot can be used to detect but not identify the AnyPW attack, because other attacks may also result in hard reboots (DoSNuke, NTFSDOS, etc.). In addition, a hard reboot may occur without an attack (power outages, system halts, etc).

casesen

U2R

Description: CaseSen is a user-to-root attack that exploits the case sensitivity of the NT object directory. The attacker ftps three attack files to the victim: soundedt.exe, editwavs.exe, and psxss.exe (the names of the files were chosen to make the attack more stealthy). The attacker then telnets to the victim and runs soundedt.exe. A new object is created in the NT object directory called \??\c: which links to the directory containing the attack files. A posix application is started activating the trojan attack file, psxss.exe, which results in the logged in user being added to the Administrators user group. Read the page from NTSecurity.net for a more complete explanation [A-66].

Simulation details: Two fully automated Expect scripts, case_s.exp and case_b.exp, are run on a UNIX attacker. Case_s.exp (the setup script) ftps the attack files, telnets to the victim, and runs the attack. It also deletes the three attack files after they have been used. Case_b.exp (the break-in script) telnets to the victim with the new administrator privileges, runs some commands, and cleans up by removing the user from the administrator's group and deleting other files generated by the attack.

First put the three attack files in /home/, where the username of the attacker is found.
Run case_s.exp from a UNIX attacker.

```
"case_s.exp <host> <user> <password>"
   e.g. "case_s.exp 172.16.112.100 lucyj lucy7"
```

Later run case_b.exp.

```
"case_b.exp <host> <user> <password>"
   e.g. "case_b.exp 172.16.112.100 lucyj lucy7"
```

Verification: After case_s.exp runs, the username specified in the command line of the attack should appear in the Administrators group on the victim machine (check the User Manager). After case_b.runs, the username should no longer be in the Administrator's group.

The tcpdump data can also be used to verify the attack. The dump file for the breakin script should contain the line "command completed successfully." This indicates that the command to remove the user from the Administrators group was successful, which means the entire attack was successful.

Cleanup: The setup script deletes the three attack files. The breakin script removes the user from the Administrator's group and deletes c:\inetpub\ftproot\winnt\, which is created during the attack setup.

Attack signature: Sniffing the network traffic will reveal the transfer of the three files, psxss.exe, editwavs.exe, and soundedt.exe, and the execution of soundedt.exe. However, editwavs.exe and soundedt.exe were a names chosen specifically for the simulation. Other versions of the attack may use different filenames, e.g., dummyapp.exe.

A strong signature is left in the victim's security log. The log shows the execution of the files posix.exe and psxss.exe, filenames that will not differ in other attack versions. Also, a log entry states the user is added to the Administrators group by NT AUTHORITY/SYSTEM because the user is added by an application (very uncommon). Normally, the Administrator would use usrmgr to add the user to a group, and the log entry would indicate that the user was added by ADMINISTRATOR, not AUTHORITY/SYSTEM.

Problems and solutions: Usually the victim machine must be rebooted before the attack can be launched a second time. The attack can be launched no more than two times without rebooting the victim.

Eject
U-b-S

Description: Eject exploits a buffer overflow in the 'eject' binary distributed with Solaris 2.5. In Solaris 2.5, removable media devices that do not have an eject button or removable media devices that are managed by Volume Management use the eject program. Due to insufficient bounds checking on arguments in the volume management library, libvolmgt.so.1, it is possible to overwrite the internal stack space of the eject program. If exploited, this vulnerability can be used to gain root access on attacked systems [A-60].

Simulation details: A truncated version of the eject exploit used in the 1998 evaluation is shown in Figure 7-2. This exploit was originally posted to the bugtraq mailing list. The exploit script, once compiled, can be run in a command line session on a Solaris server to spawn a shell that ran with root privileges. There are several ways that an intrusion detection system might identify this attack. Assuming an attacker already has access to an account on the victim machine and is running the exploit as part of a remote session, a network-based system can look at the contents of the telnet or rlogin session the attacker is using and notice one of several features. First, assuming that an attacker transmits the C code to the victim machine unencrypted, the intrusion detection system could look for specific features of the source code. For example, an intrusion detection system could look for the string 'Jumping to address' on line 45 of the source code or the line 'execl (æ/bin/eject'; æeject', & buf(char *) 0);' from line 47. Even if the attacker encrypts the source code, the attack leaves a distinct signature. A segment of the transcript from an actual instantiation of this attack that was used in the simulation is shown below: pascal> /tmp/162562 Jumping to address 0xefffff6a0 B[364] E[400] SO[704] # An intrusion detection system that saw only these three lines has several clues that an attack has taken place. First, the user's prompt has changed from 'pascal>' to '#' without running the su command. Second, the string 'Jumping to address' is again printed. Of course, a careful attacker would remove this line from the source code, but simply looking for the string 'Jumping to address' would catch the less careful attacker. Finally, a host-based intrusion detection system could catch an eject attack either by noticing the invocation of the eject program with a large argument, or by performing bottleneck verification [A-43] on the transition from normal user to root user and noticing that the user did not make a legal user to root transition.

Ffbconfig
U-b-S

Description: Ffbconfig exploits a buffer overflow in the 'ffbconfig' program distributed with Solaris 2.5. The ffbconfig program configures the Creator Fast Frame Buffer (FFB) Graphics Accelerator, which is part of the FFB Configuration Software Package, SUNWffbcf. This software is used when the FFB Graphics accelerator card is installed. Due to insufficient bounds checking on arguments, it is possible to overwrite the internal stack space of the ffbconfig program [A-61].

Simulation details: This attack is very similar to the eject attack described above. Once again, C code was posted on the Bugtraq mailing list to exploit this vulnerability.

Attack signature: The method for identifying the ffbconfig attack is similar to the Eject exploit. An attacker who is exploiting this vulnerability must first transfer the code for the exploit (either C code to be compiled, or pre-compiled code) onto the victim machine, and then run the exploit. As with the Eject exploit, there are strings within the source code of the ffbconfig exploit script which identify the attack to a network or host based intrusion detection system. A host-based intrusion detection system can perform bottleneck verification or look for the invocation of the command '/usr/sbin/ffbconfig/' with an oversized argument for the '-dev' parameter.

Fdformat
U-b-S

Description: Fdformat exploits a buffer overflow in the 'fdformat' program distributed with Solaris 2.5. The fdformat program formats diskettes and PCMCIA memory cards. The program also uses the same volume management library, libvolmgt.so.1, and is exposed to the same vulnerability as the eject program [A-60].

Simulation details: Exploit code for this vulnerability was posted to the Rootshell Website [A-53] in March, 1997. The exploit code was used unmodified for the DARPA evaluation.

Attack signature: Methods for identifying this attack are nearly identical to those described for the eject and ffbconfig attacks.

Load module

U-b-S

Description: Loadmodule is a user-to-root attack against SunOS 4.1 systems that uses the xnews window system. The loadmodule program within SunOS 4.1.x is used by the xnews window system server to load two dynamically loadable kernel drivers into the currently running system and to create special devices in the /dev directory to use those modules. Because of a bug in the way the loadmodule program sanitizes its environment, unauthorized users can gain root access on the local machine [A-8].

Simulation details: The code for the loadmodule exploit script is widely available on the internet. This code is usually in the form of a shell script—it does not need to be compiled before it is run. The steps of the attack are quite simple:

1. Change the value of the internal field separator (IFS) variable to a slash
2. Add '.' to the front of the PATH variable
3. Copy '/bin/sh' to './bin'
4. Execute '/usr/openwin/bin/loadmodule a'

When the loadmodule shell script (which is setuid root by default) executes, it attempts to run the command 'exec(æ/bin/a');'. Since the IFS variable has been changed to '/' the string '/bin/a' is parsed into two tokens, and the loadmodule script attempts to run the first 'bin'. Since the attacker has conveniently put a copy of '/bin/sh' in the current directory and named it 'bin,' the loadmodule script (which is running as root) will execute './bin'—giving the attacker a shell with root privileges.

Attack signature: This attack can be identified either by performing bottleneck verification with a host-based intrusion detection system, or by keyword spotting with a network-based intrusion detection system. A simple rule could say that any session which contained the strings 'set $IFS='\/" and 'loadmodule' in close proximity was probably a loadmodule attack. Of course, an attacker could quite easily hide from such a simple rule.

ntfsdos

U--S

Description: ntfsdos is a console-based attack that reboots the system from a floppy disk containing NTFSDOS.EXE. This executable is used to mount the hard drives, giving the attacker the ability to read and copy files that would otherwise be protected by Windows NTFS security. The attack may be considered a user-to-root attack because the attacker can access files that only the Administrator has permission to use.

NTFSDOS.EXE is a network file system redirector for DOS/Windows that is able to recognize and mount NTFS drives for transparent access. It makes NTFS drives appear indistinguishable from standard FAT drives, providing the ability to navigate, view, and execute programs from DOS or Windows.

Simulation details: The attack is completely manual. Insert the diskette (a Windows98 boot diskette containing the ntfsdos program) into Hume's A-drive. Push the reset button on the CPU. After the system reboots, type "ntfsdos" at the DOS prompt. Change directories to c:\secret. Copy the secret files to the diskette. Remove the diskette and reboot the machine.

Verification: The secret files will be stored on the diskette.

Attack signature: The attack cannot be sniffed because it does not create network traffic and it cannot be logged because windows NT is not running (and therefore not auditing) during the attack. The victim's security audit log will indicate a reboot after the system is restarted. Most likely, the attacker had to hard-reboot the machine (physically press the reset button) because he did not have a password to log in or unlock the machine. A soft reboot audit

signature is a "SeShutPrivilege" Privilege Use Event followed by an event stating, "Windows NT is starting up." A hard reboot audit signature does not include the "SeShutdownPrivilege" event.

A hard reboot can be used to detect but not identify the NTFSDOS attack, because other attacks may also result in hard reboots (DoSNuke, AnyPW, etc.). In addition, a hard reboot may occur without an attack (power outages, system halts, etc). Another way to detect the attack is to monitor the last accessed time and date for the secret files.

Perl
U-b-S

Description: Perl is a user-to-root attack that exploits a bug in some Perl implementations. Suidperl is a version of Perl that supports saved set-user-ID and set-group-ID scripts. In early versions of suidperl the interpreter does not properly relinquish its root privileges when changing its effective user and group IDs. On a system that has the suidperl, or sperl, program installed and supports saved set-user-ID and saved set-group-ID, anyone with access to an account on the system can gain root access [A-12].

Simulation details: A Perl script that uses this vulnerability to gain root access was made publicly available in August 1996 [A-51]. The code is only two lines long, and can easily be executed from the command-line. Once this Perl script has run, the user will be presented with a new shell that is running with root privileges.

Attack signature: The methods by which an intrusion detection system could identify a Perl exploit attempt are identical to those described above for the loadmodule attack. A host-based intrusion detection system could notice that a root shell was spawned without a legal user to root transition, or a network-based intrusion detection system could look the strings '$>=0; $<=0;' or 'exec (æ/bin/sh');', which have little valid use except in an exploit attempt.

Ps
U-b-S

Description: The Ps attack takes advantage of a race condition in the version of 'ps' distributed with Solaris 2.5 and allows an attacker to execute arbitrary code with root privilege. This race condition can only be exploited to gain root access if the user has access to the temporary files. Access to temporary files may be obtained if the permissions on the /tmp and /var/tmp directories are set incorrectly. Any users logged in to the system can gain unauthorized root privileges by exploiting this race condition [A-9].

Simulation details: This exploit is possible because of a combination of the ps program not carefully managing temporary files and a buffer overflow. A shell script that builds a carefully constructed temporary file, creates a C-file, compiles the code and executes the exploit was found at Rootshell.com [A-52]. Once an attacker has transfers this shell script onto a Solaris victim machine and runs it, a root shell will be spawned for the attacker.

Attack signature: Methods for finding this attack are essentially the same as the methods for finding the eject, ffbconfig, or fdformat attacks.

sechole
U--S

Description: The attacker (a regular user) ftps to the victim and uploads test.exe and testfile.dll (filenames were chosen to be stealthy). The attacker then telnets to the victim and runs test.exe. The result is the attacker is added to the administrator's group [A-72, A-73].

Test.exe locates the memory address of a particular API function (OpenProcess) and modifies the instructions at that address in a running image of the exploit program on the local system. Test.exe requests debug rights that give it elevated privileges. The request is successful because the access check for the rights is typically done in the API that was successfully modified by the exploit program. Test.exe adds the user who invoked test.exe to the local Administrators group. View the file from Microsoft Knowledge Base for more information.

Simulation details: Two fully automated Expect scripts are used. The setup script, sec_s.exp, ftps the files from a UNIX attacker, telnets to run the attack, and cleans up by deleting the attack files. The break-in script, sec_b.exp, telnets to the NT victim with the new Administrator privileges, runs some commands, and removes the attacker from the Administrators group before logging off.

First put test.exe and testfile.dll in /home/, where the username of the attacker is located. Run sec_s.exp from a Unix attacker.

```
sec_s.exp <host> <user> <password>
  e.g.  sec_s.exp 172.16.112.100 lucyj lucy7
```

Later, run sec_b.exp from a Unix attacker.

```
sec_b.exp <host> <user> <password>
  e.g.  sec_b.exp 172.16.112.100 lucyj lucy7
```

Verification: After running sec_s.exp, the username specified in the command line should be in the Administrators group on the victim machine (verify by running usrmgr). After running sec_b.exp, the user should no longer be in the Administrators group.

The tcpdump data can also be used to verify the attack. The dump file for the breakin script should contain the line "command completed successfully." This indicates that the command to remove the user from the Administrators group was successful, which means the entire attack was successful.

Cleanup: The setup script deletes the attack files. The breakin script removes the user from the Administrators group.

Attack signature: Sniffing the network traffic will reveal the uploading of test.exe and testfile.dll. The file transfers result in character strings that may be used in keyword detection. Another indicator of the attack is the security log entry for a user being added to the administrators group. The log entry states that the user is added to the administrators group by NT AUTHORITY/SYSTEM because the user is added by an application (very uncommon). Normally, Administrator would use usrmgr to add the user to a group, and the log entry would indicate that the user was added by ADMINISTRATOR, not NT AUTHORITY/SYSTEM.

Problems and solutions: It is unlikely, but the victim system may lock up after the attack. If this happens, the victim will just reboot the machine. The attack still succeeds.

Xterm

U-b-S

Description: The Xterm attack exploits a buffer overflow in the Xaw library distributed with Redhat Linux 5.0 (as well as other operating systems not used in the simulation) and allows an attacker to execute arbitrary instructions with root privilege. Problems exist in both the xterm program and the Xaw library that allow user supplied data to cause buffer overflows in both the xterm program and any program that uses the Xaw library. These buffer overflows are associated with the processing of data related to the inputMethod and preeditType resources (for both xterm and Xaw) and the *Keymap resources (for xterm). Exploiting these buffer overflows with xterm when it is installed setuid-root or with any setuid-root program that uses the Xaw library can allow an unprivileged user to gain root access to the system [A-21].

Simulation details: C source code that exploits this vulnerability on Redhat Linux 5.0 systems was found at the Rootshell website [A-56]. Once again, an attacker can compile this C code, and when the resulting program is run the attacker is given a shell running with root privileges.

Attack signature: Methods for finding this attack are essentially the same as the methods for finding the eject, ffbconfig, or fdformat attacks.

yaga

U--S

Description: Yaga is a Windows NT user-to-root attack. It adds the attacker to the Domain Admins group by hacking the registry. The attacker edits the victim's registry so that the next time a system service crashes on the victim, the attacker is added to the Domain Admins group. To setup the attack, the attacker must put onto the victim machine a file with the registry edit information. The attacker must also edit the registry. All this can be done via a telnet session. Once the setup is complete, the attacker can remotely crash a service on the victim machine (using CrashIIS for example) to add the user to the Domain Admins group.

Simulation details: The setup expect script, yaga_s.exp, telnets to the NT victim computer and creates the file. It then runs the CrashIIS attack to crash the IIS web server thereby adding the user to the Domain Admins group. The web server remains halted.

The breakin expect script, yaga_b.exp, telnets to the victim as the user with the new Domain Admin permissions, runs some commands, and cleans up by removing the user from the Domain Admins group and restoring the original registry key.

Run yaga_s.exp from a UNIX attacker.

```
yaga_s.exp <host> <user> <password>
  e.g.  yaga_s.exp 172.16.112.100 lucyj lucy7
```

Later, run yaga_b.exp.

```
yaga_b.exp <host> <user> <password>
  e.g.  yaga_b.exp 172.16.112.100 lucyj lucy7
```

Verification: After yaga_s runs, the user will be in the Domain Admins group. Run usrmgr on the victim machine to verify. After yaga_b runs, the user should no longer be in the Domain Admins group.

The tcpdump data can also be used to verify the attack. The dump file for the breakin script should contain the line "command completed successfully." This indicates that the command to remove the user from the Domain Admins group was successful, which means the entire attack was successful.

Cleanup: The breakin script removes the user from the Domain Admins group and restores the original AeDebug registry key. Manually, restart the IIS server(s). Sometimes just the web server crashes, but often times the ftp and gopher servers need to be restarted too.

Attack signature: The creation of the file with registry information, "entry," is done with the cat command. As a result, there is a lot of indicative text being sent over the data line, "Aedebug" for example. Also, the sniffed data and the NT security audit logs for the victim machine will show that regedit.exe was run by the attacker.

Another indicator of the attack is the log entry for a user being added to the Domain Admins group. The log entry states that the user is added to the Domain Admins group by NT AUTHORITY/SYSTEM because the user is added by an application (very uncommon). Normally, the Administrator would use usrmgr to add the user to a group, and the log entry would indicate that the user was added by ADMINISTRATOR, not NT AUTHORITY/SYSTEM.

Finally, because yaga uses the CrashIIS attack, any attack signature left by CrashIIS will also be left by yaga.

1.4 Remote-to-User Attacks

A remote-to-user attack occurs when an attacker who has the ability to send packets to a machine over a network but who does not have an account on that machine—exploits some vulnerability to gain local access as a user of that machine. There are many possible ways an attacker can gain unauthorized access to a local account on a machine. Some of the attacks discussed within this section exploit buffer overflows in network server software (imap, named, sendmail). The Dictionary, Ftp-Write, Guest and Xsnoop attacks all attempt to exploit weak or misconfigured system security policies. The Xlock attack involves social engineering in order for the attack to be successful the attacker must successfully spoof a human operator into supplying their password to a screensaver that is actually a trojan horse. The following sections provide details of each of these attacks used in the 1998 and 1999 DARPA evaluations.

Dictionary

R-a-U

Description: Dictionary is a remote-to-local user attack in which an attacker tries to gain access to some machine by making repeated guesses at possible usernames and passwords. Users typically do not choose good passwords, so an attacker who knows the username of a particular user (or the names of all users) will attempt to gain access to this user's account by making guesses at possible passwords. Dictionary guessing can be done with many services; telnet, ftp, pop, rlogin, and imap are the most prominent services that support authentication using usernames and passwords. Figure 8-2 is a plot of the connections made to the pop3 port of a victim machine during a dictionary attack that is using the pop service to check for valid login/password combinations. The horizontal axis of this plot represents time in minutes, and each line segment in the plot is a single connection to the pop3 service. Lines representing successive sessions are displaced vertically slightly and wrap around (in this figure at roughly 1.5 minutes). The length of the lines represents the length of the pop sessions. Lines begin with a greater-than sign '>' and end with a less-than sign '<' and thus form an 'x' for short sessions. In all, this example dictionary attack consists of 40 attempts to log in, with a 4-second delay between each attempt.

Simulation details: A Perl script that performed automated password guessing on a variety of services was developed specifically for use in our evaluation. The 'Netguess' Perl script could take in a file of possible username/password combinations, or create guesses for the password based on simple permutations of the username. Within the simulation, this script was used to perform between 10 and 100 login attempts on the telnet, ftp, and pop services. When the script was successful in gaining access to the system, it would immediately quit and report success.

Attack signature: An intrusion detection system that finds attempted dictionary attacks needs to know the session protocol of every service that provides username/password authentication. For a given service, the intrusion detection system must be able to recognize and record failed login attempts. Once this functionality is available, detecting dictionary attacks is a matter of setting a detection threshold based on the number of failed login attempts within a given period of time.

FrameSpoofer

R-m-Alter

Description: FrameSpoofer tricks the victim into believing he is viewing a trusted web site, but in actuality the page's main body is spoofed with a frame created by the attacker [A-74].

The attacker sends a fake email to the victim directing the victim to a web page that displays security procedures for Air Force Base computer networks. The page contains a link to a page with security procedures specific to Eyrie Air Force Base. When the user clicks on the link, it actually runs a javascript function that brings up

the trusted web site but then inserts a malicious web page (with misleading information) into its main frame. The URL displayed in the browser remains unchanged.

View the page from NTSecurity.net for more information.

Simulation details: Use sendmail.pl to send the email to Hume. The victim must manually receive the mail and click on the links. On a UNIX attacker: "/.sim/bin/sendmail.pl mail.txt." As the victim, receive the mail and follow the links.

Important: Clear the browser cache on the victim machine before executing the attack. Otherwise, the browser will load cached pages from a previous execution of the attack, and no web traffic will be generated on the network.

Verification: The security procedures page for Eyrie Air Force base will display one page and then switch to the spoofed page.

Attack signature: Audit logs reveal nothing. The sniffed data will reveal the source code for the javascript webpage.

The attack can be detected if one carefully examines the javascript code contained in the sniffed data.

Ftp-write

R-c-U

Description: Ftp-write is a remote-to-local user attack that takes advantage of a common anonymous ftp misconfiguration. The anonymous ftp root directory and its subdirectories should not be owned by the ftp account or be in the same group as the ftp account. If any of these directories are owned by ftp or are in the same group as the ftp account and are not write protected, an intruder will be able to add files (such as an rhosts file) and eventually gain local access to the system [A-7].

Simulation details: This attack was implemented as an expect script which was created explicitly for use in the simulation. This expect script anonymously logged in to the ftp service on the victim machine, created a '.rhosts' file with the string '+ +' in it within the ftp home directory, disconnected from the ftp server, used rlogin to connect back to the server as user 'ftp,' and finally performed some actions on the victim machine. Creating a '.rhosts' file in the ftp home directory with the entry '+ +' in it allows any user from any machine to rlogin to the victim as user 'ftp.'

Attack signature: An intrusion detection system can monitor for this attack by watching all anonymous ftp sessions and assuring that no files are created in the ftp root directory.

Guest

R-c-U

Description: Guest is a variant of the Dictionary attack described in Section 8.1. On badly configured systems, guest accounts are often left with no password or with an easy to guess password. Because most operating systems ship with the guest account activated by default, this is one of the first and simplest vulnerabilities an attacker will attempt to exploit [A-27].

Simulation details: The Guest attack is a simplified version of the Dictionary attack discussed earlier in this chapter. The same 'Netguess' Perl script that was used to simulate a Dictionary attack was used to simulate the Guest attack—the only difference between the implementation of the two attacks was the command-line options that was passed to the Netguess program. Whereas the Dictionary attack would try up to a hundred user names and thousands of username/password combinations, the Guest attack would make only a couple of login attempts, using combinations such as 'guest/,' 'guest/guest,' 'anonymous/' and 'anonymous/anonymous.'

Attack signature: Because the Guest attack is essentially a subset of the Dictionary attack, the methods for finding the two attacks are basically the same. An intrusion detection system that is looking for a Dictionary attack already should need only minor tuning in order to find attempts to log in to the guest account.

HttpTunnel

Description: An Http Tunnel attacker gains local access to the victim machine and then sets up and configures an http client to periodically query a web server that the attacker has set up at some remote host. When the client connects, the server is able to send cookies that could request information be sent by the client, such as the password file on the victim machine. In effect, the attacker is able to "tunnel" requests for information through the http protocol.

Simulation details: This attack is carried out in two phases: first a setup, then use of the tunnel. The setup occurs with a login, by the attacker as a local user, to the victim, and a transfer of the web-client (script, source code, or binary) to the victim. During the setup login, the attacker might also add to their crontab file so that the webclient is periodically started. The tunnel is used when the webclient connects to the attacker's server, cookies requesting information are sent to the victim and the requests are fulfilled. This web server could run on any port the attacker chooses, however, most often a lesser known port above 1024 will be used.

Attack signature: Http Tunnel can be recognized by watching for the setup login session, transfer of the client to the victim, and perhaps setting up a job to be run periodically, or starting a background process to run the client. Using the tunnel could be noticed by periodic connections from the victim to the attacker on non-well-known ports, or ports greater than 1024. (However, port 80 could be used as well.)

Imap

R-b-S

Description: Imap exploits a buffer overflow in the Imap server of Redhat Linux 4.2 that allows remote attackers to execute arbitrary instructions with root privileges. The Imap server must be run with root privileges so it can access mail folders and undertake some file manipulation on behalf of the user logging in. After login, these privileges are discarded. However, a buffer overflow bug exists in the authentication code of the login transaction, and this bug can be exploited to gain root access on the server. By sending carefully crafted text to a system running a vulnerable version of the Imap server, remote users can cause a buffer overflow and execute arbitrary instructions with root privileges [A-16].

Simulation details: The Imap attack used in the 1998 DARPA intrusion detection evaluation was part of the Impack 1.03 attack toolkit [A-34]. This toolkit contained precompiled binary programs for the Linux platform that would scan for vulnerable machines, as well as send the necessary message to exploit the buffer overflow and gain access to a root shell. The Impack contained detailed instructions on how to use these precompiled programs and took very little skill to use. The release of the Impack made this vulnerability especially dangerous, as any user with a Linux machine and the ability to follow instructions could use this attack to remotely gain root access to any vulnerable hosts.

Attack signature: The Imap attack can be identified by an intrusion detection system that has been programmed to monitor network traffic for oversized Imap authentication strings.

Named

R-b-S

Description: Named exploits a buffer overflow in BIND version 4.9 releases prior to BIND 4.9.7 and BIND 8 releases prior to 8.1.2. An improperly or maliciously formatted inverse query on a TCP stream destined for the Named service can crash the Named server or allow an attacker to gain root privileges [A-19].

Simulation details: The version of the Named exploit used in the simulation was adapted from a C program originally posted to the Bugtraq mailing list. This

program, once compiled, would connect to the named port on a victim machine and overflow a buffer of the named server with instructions that would send an xterm running with root privilege back to the attacker's X console. Because this attack involved interaction with X, all of the Named attacks included in the simulation were run by human actors.

Attack signature: The Named attack can be identified by watching DNS inverse query requests for messages that are longer than the 4096 byte buffer allocated for these requests within the 'named' server.

ncftp

R-b-User

Description: Ncftp is an ascii UI ftp program for Linux. This attack exploits one of the popular features of the program: the ability to get subdirectories recursively. New (sub)directories are created on the local machine using the system() command (e.g. if any directories on the remote host contain an expression in backticks, that expression will be evaluated on the local machine when the directory is created. http://www.rootshell.com/archive -j457nxiqi3gq59dv/199803/ncftp.html.

Simulation details: A directory was created on an outside attacking machine with a expression in backticks. This malicious directory was hidden underneath some other directories. When a user on a victim machine used ncftp to download the top level directory recursively, the malicious command was executed on the victim's host. In the 1999 DARPA evaluation, the malicious command mailed the victim's /etc/passwd file to the attacker. This attack was originally posted on http://www.rootshell.com.

Attack signature: In sniffing data, you can see the names of all the files that are transferred in an ftp session. However, it is difficult to see what the malicious command is because most of the harmful expression in backticks is encoded in octal character codes. In addition, the mail sent back over to the attacker was encrypted by adding in spurious characters using sed so the file doesn't resemble /etc/passwd.

netbus

R--User

Description: NetBus is a remote-to-local attack. The attacker uses a trojan program to install and run the Netbus server on the victim machine. Once Netbus is running, it acts as a backdoor. The attacker can then remotely access the machine using the Netbus client [A-67].

The attacker sends an email with an executable attachment (a game called whackamole). When the victim runs whackamole, it launches the Netbus server (explore.exe) and then launches the whackamole game. It also edits the registry so the Netbus server runs at every login.

The attacker can use the Netbus client program to manipulate files on the victim machine, download screen dumps, move the mouse pointer, etc. The attacker's access privileges are identical to the user currently logged on to the victim machine. So if an administrator is using the victim, the attacker can run a net command to setup a new admin user, resulting in a remote-to-root attack. The Netbus client can also be used as a probe attack to scan IP addresses for NetBus servers.

View the information page from www.netbus.com.

Simulation details: Use Sendmail.pl to send the email to Hume. On a UNIX attacker: "/.sim/bin/sendmail.pl netbus.txt" or send the email with attachment from an NT attacker. Later, on a NT attacker, run the NetBus client with the victim IP address (172.16.112.100).

Verification: After the attack has completed, the victim machine should be remotely accessible via the Netbus client running on a Windows NT machine.

Cleanup: Click the Server Admin button on the NetBus client and choose Remove Server. The registry will remain changed and explore.exe will remain in c:\winnt but the server will no longer be running and will not run until the attack is setup again.

Attack signature: When the attacker uses the Netbus client to access the victim, it creates network traffic that is easy to identify. The word Netbus will show up in the sniffed data and all of the commands are in plaintext.

Explore.exe is the most commonly used filename for the Netbus attack. The NT security log will show that explore.exe ran when the attachment was executed. When the Netbus server is running, it shows up in the victim process table as explore.exe.

netcat

R--User

Description: NetCat is a remote-to-local attack. The attacker uses a trojan to install and run the netcat program on the victim machine on a specific port (53). Once netcat is running, it acts as a backdoor. The attacker can remotely access the machine through the netcat port without a username or password.

The attacker sends an email with a self-extracting executable attachment called y2ktest.exe. The email states that the file will install a program to test the victim machine for y2k compliance. When the victim opens the file, it creates a new folder c:\y2ktest and puts the y2ktest files into it. It also places into the folder the attack files, winlog.bat, winlog.exe, and winlog.txt (filenames were chosen to be stealthy).

The batch file, winlog.bat, automatically runs and controls the attack. It edits the Run key in the registry with winlog.txt so that the command

```
winlog -L -d -p 53 -t  -e cmd.exe
```

runs every time a user logs on to the machine. Finally, all unnecessary attack files are deleted. The y2ktest folder and its contents, and c:\winnt\system32\winlog.exe are what remain.

The attacker later uses the command "nc -v 53" on a remote machine (UNIX or NT with the nc program) to telnet to the victim without a username or password.

View the page from l0pht.com for more information.

Simulation details: A UNIX attacker sets up the backdoor on an NT victim machine. Usually, an NT attacker uses the backdoor, but a UNIX attacker can use it as well.

Sendmail.pl is used to send the preassembled mail message y2katack.txt. The victim must manually open the email and runs the trojan. Then the attacker manually runs the client program.

On a UNIX attacker:

```
/.sim/bin/sendmail.pl y2katack.txt
```

Later, on a UNIX or NT attacker with netcat:

```
nc -v 172.16.112.100 53
```

The files included in the self-extracting zip file were originally called nc.bat, nc.exe, and nc.txt. They were changed to winlog.bat, winlog.exe, and winlog.txt respectively. This way, when the backdoor runs, winlog will appear in the victim's process list instead of the more conspicuous name, nc. Sniffing the email transfer will not detect the attack because the attachment is MIME encoded.

When the self-extracting zip file is run, it tells the user that it puts a total of seven files into c:\y2ktest. Because the attack files are moved or deleted, the batch file

copies one of the y2ktest files three times and renames them, (check1, check2, check3) so there are still seven files in the y2ktest directory.

The attack modifies the registry but does not run netcat (winlog) right away. The backdoor does not take affect until the victim user logs out and logs in again. This increases stealth because the setup of the attack is split into two steps.

The attacker should wait a few hours or days before exploiting the backdoor so that it is more difficult for the victim to connect the breakin with the setup. NetCat can use any port, but if it uses port 23, all telnet sessions to the victim will be unauthenticated.

Verification: After the attack has completed, the victim machine should be remotely accessible without authentication: On a remote machine, type the command
`nc -v 172.16.112.100 53`

Cleanup: Delete the winlog command from the Run registry key and remove winlog from the process table, if it is there.

Attack signature: When the attacker connects to the backdoor, sniffing will reveal what looks like a telnet session, but it does not begin with a login name and password and it occurs on port 53.

The security audit log will show the execution of REGEDIT (trojan runs), later followed by the execution of winlog.exe (backdoor activated).

Phf

R-b-U

Description: Phf abuses a badly written CGI script to execute commands with the privilege level of the http server. Any CGI program which relies on the CGI function escape_shell_cmd() to prevent exploitation of shell-based library calls may be vulnerable to attack. In particular, this vulnerability is manifested by the "phf" program that is distributed with the example code for the Apache web server [A-11].

Simulation details: Phf is quite simple to implement because it requires only the ability to connect to a network socket and issue an http request. Within the simulation, the Netcat [A-31] program was used to generate this http request. Although this vulnerability allows an attacker to run any command on the server, the command used throughout the simulation was '/bin/cat /etc/passwd.' Using Phf with this command reveals the contents of the victim system's password file to users with no account on the victim machine.

Attack signature: To find the Phf attack, an intrusion detection system can monitor http requests watching for invocations of the phf command with arguments that specify commands to be run. Examples of commands that an attacker might attempt to execute by exploiting the phf exploit are: cat /etc/passwd, id, whoami, or xterm.

ppmacro

R--User

Description: ppmacro is a remote-to-local attack uses a trojan PowerPoint macro to read secret files. This attack is based on a particular scenario. The victim user usually receives PowerPoint templates from an outside source via email attachment. He runs a built-in macro that inserts a graph displaying web statistics, saves the presentation as a ppt file, and posts it on the web.

The attacker writes a fake email and sends the template with additional code appended to the macro. The attack code reads secret files from the victim machine (in d:\home\secret\) and inserts them as white text in the background on the master slide of the presentation. When the presentation is posted on the web, the attacker can examine the ppt file to reveal the text of the secret file. The macro also stores a counter in the victim's registry so each time the user runs the macro, a different file from the secret directory is inserted into the presentation.

Simulation details: One expect script, ppmacro.exp, creates and sends the email to an NT victim. The victim must manually open the email, create the PowerPoint presentation, and post it on the web. On a Unix attacker: "/.sim/bin/sendmail.pl ppatack.txt " The victim must then run Wusage (creates web statistics), rename one of the graphs in c:\winnt\reports to graph.gif, run PowerPoint, run the macro, and save the ppt file somewhere in c:\inetpub\wwwroot. The attacker later uses Netscape to download the ppt file.

Verification: After the attack has completed, the attacker should be able to view the secret file by downloading the ppt file from the web. Run strings on the file to see the secret file text.

Cleanup: To fully cleanup, delete the webstats key from the victim's registry. This is not recommended because editing the registry is detectable.

Attack signature: The sniffed data will reveal the secret file text being transferred when the attacker downloads the ppt file from the web. The attacker could modify the macro to encrypt the secret file thereby making the attack more stealthy. Auditing reveals nothing about the attack.

Sendmail

R-b-S

Description: The Sendmail attack exploits a buffer overflow in version 8.8.3 of sendmail and allows a remote attacker to execute commands with superuser privileges. By sending a carefully crafted email message to a system running a vulnerable version of sendmail, intruders can force sendmail to execute arbitrary commands with root privilege [A-15].

Simulation details: Although this vulnerability was widely known about at the time of the evaluation, no code that exploited this vulnerability had been posted to public forums such as Bugtraq, or Rootshell.com. A significant period of time (one person working for more than two weeks) was spent developing the first known implementation of this exploit explicitly for use in the evaluation. The implementation consists of a carefully constructed mail message which, when sent to the victim machine with a vulnerable version of sendmail, adds a new entry with root privilege to the end of the password file on the victim system. Once this new entry has been added, the attacker can log into the machine as this new user and execute commands as a root user. Figure 8-3 provides an illustration of an instantiation of the Sendmail attack as it was implemented in the simulation. In step 1 of this illustration, the attacker sends a carefully crafted e-mail message to the victim machine. In step 2, the sendmail daemon starts to process this message, overflows one of its buffers, and executes the attacker's inserted commands that create a new entry in the password file. In step 3, the attacker comes back to the victim machine and uses the new password file entry to gain root access to the victim machine and perform some malicious actions.

Attack signature: The Sendmail attack overflows a buffer in the MIME decoding routine of the sendmail program. In order for an intrusion detection system to identify a Sendmail attack it must monitor all incoming mail traffic and check for messages that contain a MIME header line that is inappropriately large.

sshtrojan

R-a-Deny
(Temp./Admin.)

Description: The SSH Trojan attacker tricks the system administrator into installing (as a "Y2K Upgrade") a trojan version of the SSH program. This trojan version allows the attacker (or anyone!) to log in to the victim, via ssh, with the login "monkey" and no password. Upon login, a root privilege shell is spawned for the attacker.

Simulation details: During the setup phase of the attack an email is sent to the system administrator advising that he/she should install a particular Y2K-compliant version of the ssh server software that they use. The administrator then installs the software, thereby replacing the good version of sshd with the trojan version.

During the break-in phase, the attacker uses ssh to login to the victim via ssh as user "monkey." The login is successful, does not require a password, and yields a rootshell for the attacker.

Attack signature: The setup phase of the attack could be detected perhaps by identifying the initial email message as a hoax, or by detecting the download of sshd program software from a suspicious site. As the break-in part of the attack is carried out in an encrypted session, there is little evidence on the network. However if host-based auditing is available for the victim host, one might be able to detect the root shell spawned by ssh. (In the 1999 evaluation, however, the attack was run against Linux, where no host auditing was available.)

Xlock

R-cs-Intecept (Keystrokes)

Description: In the Xlock attack, a remote attacker gains local access by fooling a legitimate user who has left their X console unprotected, into revealing their password. An attacker can display a modified version of the xlock program on the display of a user who has left their X display open (as would happen after typing 'xhost +'), hoping to convince the user sitting at that console to type in their password. If the user sitting at the machine being attacked actually types their password into the trojan version of xlock the password will be sent back to the attacker.

Simulation details: This attack was created specifically for use in the evaluation. A special version of the xlock program was created by making small modifications to the publicly available source code for the xlock program. The standard version of xlock will not display on a remote display, and does not save or output the password that the user enters. A modified version of xlock was created that allows the attacker to display the screen saver on a remote display and returns the password entered by the victim. Because of the required interaction with the X server, the Xlock attack was always run by a human actor.

Attack signature: Two factors make this attack quite difficult for an intrusion detection system to identify. First, this attack is a spoofing attack that does not abuse a bug in the system, but relies on assumptions by the person who is currently using the system. Second, this attack is embedded in the X protocol, and an intrusion detection system that is trying to identify this attack must understand and parse these X communications in order to identify the Xlock attack. One non-optimal way of dealing with these difficulties, is to program an intrusion detection system to identify as suspicious any X traffic that is originating from an unknown machine that is destined for a machine being monitored.

Xsnoop

R-c-Intercept (Keystrokes)

Description: In the Xsnoop attack, an attacker watches the keystrokes processed by an unprotected X server to try to gain information that can be used gain local access the victim system. An attacker can monitor keystrokes on the X server of a user who has left their X display open. A log of keystrokes is useful to an attacker because it might contain confidential information, or information that can be used to gain access to the system such as the username and password of the user being monitored.

Simulation details: The Xsnoop program used in the simulation was adapted from C code originally posted to the Bugtraq mailing list. The Xsnoop program runs locally on the attacker's machine. The program connects to the victim's X Server and requests to be notified of all X KeyPress Events. If the attacker has permission to make this request (as would happen if the victim had typed 'xhost +', or if the victim's X Server is configured to allow connections from all hosts by default) the

Xsnoop program will be sent all KeyPress events that occur on the victim X Server. The Xsnoop program then uses the KeyPress events to provide the attacker with a view of all the keystrokes the victim. Because this attack required interaction with the X Server, it was always run by a human actor.

Attack signature: An network based intrusion detection system can identify the Xsnoop attack by parsing the X protocol information in packets destined for remote X clients and noting that X KeyPress events are being transmitted to a remote machine. Because the Xsnoop attack results from bad security policy (leaving an X Server unsecured) and not simply a bug, the presence of these packets alone does not signify that an attack is taking place. The intrusion detection system must know whether the security policy of the machine being monitored allows unauthenticated X connections from anywhere.

1.5 Probes

In recent years, a growing number of programs have been distributed that can automatically scan a network of computers to gather information or find known vulnerabilities [A-27]. These network probes are quite useful to an attacker who is staging a future attack. An attacker with a map of which machines and services are available on a network can use this information to look for weak points. Some of these scanning tools (satan, saint, mscan) enable even a very unskilled attacker to very quickly check hundreds or thousands of machines on a network for known vulnerabilities. The following sections describe in detail each of the probes used in the 1998 and 1999 DARPA intrusion detection evaluation.

insidesniffer

P-a-Probe
(Machines, Users)

Description: Insidesniffer attaches a new machine to an inside ethernet hub, configured with an IP, and begins sniffing traffic.

Simulation details: The attack can be carried out in two ways: a fairly "obvious" way to sniff is to allow or request the sniffing software attempt to resolve (using DNS) the IP addresses to names, while a stealthier way is to disable this DNS lookup.

Attack signature: In the "clearer" case above, the Intrusion Detector could look for traffic to/from a completely new IP address, and that the traffic would consist entirely of dns queries for addresses from which it had just seen packets! In the stealthier case, we have not identified a foolproof way to detect the sniffer, given the "off-line" style of these evaluations. Some abnormalities might exist, in particular, if the sniffer was assigned an IP address, a subsequent ipsweep might reveal the presense of the new machine. However, this might or might not indicate that the interface was in promiscuous-mode.

Ipsweep

R-a-Probe
(Machines)

Description: An Ipsweep attack is a surveillance sweep to determine which hosts are listening on a network. This information is useful to an attacker in staging attacks and searching for vulnerable machines.

Simulation details: There are many methods an attacker can use to perform an Ipsweep attack. The most common method—the method used within the simulation—is to send ICMP Ping packets to every possible address within a subnet and wait to see which machines respond. The Ipsweep probes in the simulation were not stealthy—the sweeps were performed linearly, quickly, and from a single source.

Attack signature: An intrusion detection system looking for the simple Ipsweep used in the simulation can look for many Ping packets, destined for every possible machine on a network, all coming from the same source.

ls_domain

R-a-Probe (Network, Machines)

Description: The ls domain attacker uses the "nslookup" command in interactive mode to "list" all machines in a given DNS domain from a misconfigured primary or secondary DNS server. Thus the attacker can learn what machines (IP addresses) belong to (and perhaps exist in) the domain.

Simulation details: The attacker (scripted with expect) performs the above operation and stores the knowledge in a file. Later, the file is opened, and each IP address is probed in some fashion. In the 1999 evaluation, the subsequent probe was an nmap probe of port 80 on each listed host.

Attack signature: The Detector might look for, first the domain listing process, and then will see the probe of all IPs listed in the DNS record. In particular, the attacker will probe machines that don't exist, but are listed.

Mscan

R-a-Probe (Known Vulnerabilities)

Description: Mscan is a probing tool that uses both DNS zone transfers and/or brute force scanning of IP addresses to locate machines, and test them for vulnerabilities [A-20].

Simulation details: The Mscan program used in the simulation was compiled from source code found at [A-57]. Mscan was easy to run and has several command line options for specifying the number of machines to scan and which vulnerabilities to look for. Within the simulation, mscan was used to scan the entire eyrie.af.mil domain for the following vulnerabilities: statd, imap, pop, IRIX machines that have accounts with no passwords, bind, various cgi-bin vulnerabilities, NFS, and open X servers.

Attack signature: The signature of this attack will vary depending on which vulnerabilities are being scanned for and how many machines are being scanned. In general, an intrusion detection system can find an Mscan attack by looking for connections from a single outside machine to the ports listed above on one or more machines within a short period of time.

Nmap

R-a-Probe (Services)

Description: Nmap is a general-purpose tool for performing network scans. Nmap supports many different types of portscans—options include SYN, FIN and ACK scanning with both TCP and UDP, as well as ICMP (Ping) scanning [A-45]. The Nmap program also allows a user to specify which ports to scan, how much time to wait between each port, and whether the ports should be scanned sequentially or in a random order.

Simulation details: At the time of the evaluation, Nmap was the most complete publicly available scanning tool. During the simulation, Nmap was used to perform portscans on between one and ten computers using SYN scanning, FIN scanning, and UDP scanning of victim machines. Both sequential and random scans were performed in the simulation, and the timeout between packets was varied to be anywhere from one second to six minutes. The number of ports scanned on each machine was varied between three and one thousand.

Attack signature: The signature of a portscan using the Nmap tool varies widely depending on the mode of operation selected. Despite this variance of modes, all portscans share some common features. A portscan can be recognized by noting that network packets (whether via TCP or UDP, or via only FIN packets or only SYN packets) have been sent to several (or more) ports on a victim or group of victims within some window of time. Two factors complicate the identification of portscans. First, a portscan can happen very slowly. A patient attacker could probe one port per day. Current intrusion detection systems do not keep enough state to recognize a portscan happening over a long period of time. Given the amount of network traffic sent over a typical network, a 100-day-long portscan is simply impractical to keep enough active state within the ID system to recognize one connection per day for 100 days. Second, the connections don't necessarily all have to come from the same host. A

group can perform a coordinated scan with each member scanning only a subset of machines or ports. By combining these methods, a group could perform a 'low/slow' portscan that would be very hard to recognize.

NTinfoscan

R-a-Probe (Known Vulnerabilities)

Description: NTInfoScan is a NetBIOS based security scanner. It scans the NT victim to obtain share information, the names of all the users, services running, and other information. The results are saved in an html file named ".html" where "victim" is the victim's hostname [A-71].

Simulation details: A Perl script runs on an NT attacker. Edit the first line of the ntis.pl with the time of day the attack should run and then run ntis.pl or put it in the Startup group and restart the machine. Ntis.pl automatically scans hume.eyrie.af.mil. The attack may take up to 20 minutes to complete.

Verification: There will be a file named hume.eyrie.af.mil.html in c:\.sim\attacks\logs (make sure the last modified date agrees with the date the attack was launched). Open the file to verify that data was collected by the scan.

Attack signature: Sniffing reveals that the attack FTPs to the victim as user anonymous with password guestaccnt@compuserve.com and makes numerous HTML GET requests to files in such directories as /cgi-bin and /scripts. Originally, the ntis ftp'd to the victim with the password "ntinfoscan."

The security audit log can also be used to detect the attack. A login by IUSR via Advapi, followed by the execution of newdsn.exe by SYSTEM indicates a web scan. A login via KsecDD followed by multiple SAM_USER accesses by SYSTEM indicates a netbios scan.

Problems and solutions: Often, ntis will temporarily hang during the web services portion of the attack if it attempts GET request for inaccessible files. There is a timeout of 15min, after which the attack will complete.

queso

R-a-Probe

Description: QueSO is a utility used to determine a what type of machine/operating system exists at a certain IP adress. QueSO sends a series of 7 tcp packets to any one port of a machine and uses the return packets it receives to lookup the machine in a database of responses.

Simulation details: To make the attack more stealthy, we increased the delay between sending the packets. In the 1999 DARPA evaluation machines are sent the 7 QueSO packets with delays of 3, 5, and 10 minutes.

Attack signature: Since the time window in between packets can be fairly large, QueSO can be difficult to detect. The first 4 packets are normal requests to open and close a connection. The remaining 3 packets are abnormal requests looking for anomolous behavoir to help classify the machine and OS. The abnormal packets will flag systems looking for odd combinations of TCP flags or attempts to use TCP reserve bits.

resetscan

--Probe

Description: ResetScan sends reset packets to a list of IP addresses in a subnet to determine which machines are active. If there is no response to the reset packet, the machine is alive. If a router or gateway responds with "host unreachable," the machine does not exist.

Simulation details: A Perl script runs on a Linux attacker. ResetScan.pl reads a file containing a list of IP addresses and runs rscan with each address. After one minute, resetscan.pl runs rscan with each IP address again. The first run and the pause are necessary to give the gateway time to determine active hosts. The second run produces the actual results of the attack.

Verification: The dump file will show reset packets sent to any of the IP addresses scanned that actually exist. In addition the tcpdump file will contain "arp who-has" requests for the range of IPs scanned that don't exist (from the first round of resets).

Attack signature: Sniffing reveals reset packets sent to IP addresses with no previous connections, as well as a numerous "arp who-has" packets sent by the gateway, requesting mac addresses for ranges of non-existant IP addresses.

Saint

R-a-Probe (Known Vulnerabilities)

Description: Saint is the Security Administrator's Integrated Network Tool. In its simplest mode, it gathers as much information about remote hosts and networks as possible by examining such network services as finger, NFS, NIS, ftp and tftp, rexd, statd, and other services. The information gathered includes the presence of various network information services as well as potential security flaws. These flaws include incorrectly setup or configured network services, well-known bugs in system or network utilities, and poor policy decisions. Although Saint is not intended for use as an attack tool, it does provide security information that is quite useful to an attacker [A-58]. Saint is distributed as a collection of perl and C programs and is known to run on Solaris, Linux, and Irix systems. Within the simulation, the Saint program was run from a Linux traffic generator and was used to probe several victim machines for vulnerabilities. Saint's behavior is controlled by a configuration file which allows the user to specify several parameters. The most important parameters are the list of machines to scan, and how heavily to scan these machines (light, normal, or heavy). In light mode, Saint will probe the victim for dns and rpc vulnerabilities and will look for unsecured NFS mount points. In normal mode, Saint will also check for vulnerabilities in fingerd, rusersd, and bootd, and will perform a portscan on several tcp (70, 80, ftp, telnet, smtp, nntp, uucp) and udp (dns, 177) ports. Heavy mode is the same as normal mode except that many more ports are scanned. A Saint scan of a network leaves a distinct signature that will vary depending on the level of scanning being performed. The Saint program performs each scan in a nearly deterministic fashion. To identify a Saint scan, an intrusion detection system needs to be able to recognize the distinct set of network traffic the scan creates. Figure 9-2 is a plot that provides a graphical view of this signature. The horizontal axis of this plot represents time in minutes, and the different services probed are presented along the vertical axis. The names of the services are shown on the left side of this plot, and connections for each service are plotted within each named region. The numbers after the service names are the number of separate tcp connections or of udp or icmp packets. Names ending in '/i' indicate that packets use the icmp protocol and names ending in '/u' indicate that packets use the udp protocol. All other services use the tcp protocol. Each line segment represents a connection to a service. This plot shows the unique signature of a medium level Saint scan. Because this signature does not change significantly across multiple instantiations of the Saint attack, an intrusion detection system that has been trained to recognize the pattern of connections shown in Figure 9-2 will probably detect other Saint attacks.

Satan

R-a-Probe (Known Vulnerabilities)

Description: Satan is an early predecessor of the Saint scanning program described in the last section. While Saint and Satan are quite similar in purpose and design, the particular vulnerabilities that each tools checks for are slightly different [A-24].

Simulation details: Like Saint, Satan is distributed as a collection of Perl and C programs that can be run either from within a web browser or from the UNIX command prompt. Satan supports three levels of scanning: light, normal, and heavy. The vulnerabilities that Satan checks for in heavy mode are: NFS export to unprivileged programs, NFS export via portmapper, NIS password file access, REXD access, tftp file access, remote shell access, unrestricted NFS export, unrestricted X Server access, write-able ftp home directory, several Sendmail vulnerabilities, and several ftp vulnerabilities. Scans in light and normal mode simply check for smaller subsets of these vulnerabilities.

Attack signature: A Satan scan of a network can be recognized by the consistent pattern of network traffic the program creates. The checks for the vulnerabilities listed above are always performed in the same order. Figure 9-3 shows a plot of the services probed during an example medium level Satan scan. The horizontal axis of this plot represents time in seconds and the various services that are probed are presented on the vertical axis. Each line segment in the plot represents a single connection to a service. Every medium level Satan scan will have a signature very similar to that shown in Figure 9-3.

1.6 Data

Data attacks involve someone (user or administrator) performing some action that they may be able to do on a given computer system, but that they are not allowed to do according to site policy. Often, these attacks will involve transferring "secret" data files to or from sources where they don't belong.

Secret

Description: Secret is an attack where the attacker maliciously or mistakenly transfers data to a place where it doesn't belong. For example, transferring data from a classified computer/network to a non-classified computer/network would constitute a "secret" attack.

Simulation details: We simulate these types of attacks by publishing a set of rules indicating that all files in a particular directory are "secret" and that they can not be moved out of that directory (whether by 'cp,' 'cat,' 'ftp,' or whatever.) Then the attacker logs in and performs such an action.

Attack signature: To recognize these attacks, the detection system must know which files are considered secret, what the policies are regarding use of these files, and then simply look for actions carried out involving them. Naturally, attacks such as these can be hard to detect—a legitimate user could "cut-and-paste" information from one desktop window to another.

APPENDIX A
REFERENCES

[A-1] Anderson, D., T. Lunt, H. Javitz, A. Tamaru, and A. Valdes, "Safeguard Final Report: Detecting Unusual Program Behavior Using the NIDES Statistical Component," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report, December 1993.

[A-2] Anonymous. Maximum Security: A Hacker's Guide to Protecting Your Internet Site and Network, Chapter 15, pp. 359-362. Sams.net, 201 West 103rd Street, Indianapolis, IN, 46290. 1997.

[A-3] Bishop, M., S. Cheung, et al., "The Threat from the Net," IEEE Spectrum, 38(8). 1997.

[A-4] Bugtraq Archives (e-mail regarding Apache vulnerability). http://www.geek-girl.com/bugtraq/1998_3/0442.html. August 7, 1998.

[A-5] Bugtraq Archives. http://geek-girl.com/bugtraq/1997_4/0283.html. November 13, 1999.

[A-6] Bugtraq Archives. http://www.geek-girl.com/bugtraq/1999_1/0852.html. February 19, 1999.

[A-7] CERT Advisory CA-93.10. http://www.cert.org/ftp/cert_advisories/CA-93%3a10.anonymous.FTP.activity. July 14, 1993.

[A-8] CERT Advisory CA-93.18,CA-95:12. http://www.cert.org/ftp/cert_advisories/CA-95:12.sun.loadmodule.vul. September 19, 1997.

[A-9] CERT Advisory CA-95.09. http://www.cert.org/ftp/cert_advisories/CA-95%3a09.Solaris-ps.vul. August 20, 1995.

[A-10] CERT Advisory CA-96.01. http://www.cert.org/ftp/cert_advisories/CA-96.01.UDP_service_denial. February 8, 1996.

[A-11] CERT Advisory CA-96.06. http://www.cert.org/ftp/cert_advisories/CA-96.06.cgi_example_code. March 20, 1996.

[A-12] CERT Advisory CA-96.12. http://www.cert.org/ftp/cert_advisories/CA-96.12.suidperl_vul. June 26, 1996.

[A-13] CERT Advisory CA-96.21. http://www.cert.org/ftp/cert_advisories/CA-96.21.tcp_syn_flooding. September 19, 1996.

[A-14] CERT Advisory CA-96.26. http://www.cert.org/ftp/cert_advisories/CA-96.26.ping. December 16, 1996.

[A-15] CERT Advisory CA-97.05. http://www.cert.org/ftp/cert_advisories/CA-97.05.sendmail. January 28, 1997.

[A-16] CERT Advisory CA-97.09. http://www.cert.org/ftp/cert_advisories/CA-97.09.imap_pop. April 7, 1997.

[A-17] CERT Advisory CA-97.28. http://www.cert.org/ftp/cert_advisories/CA-97.28.Teardrop_Land. December 16, 1997.

[A-18] CERT Advisory CA-98.01. http://www.cert.org/ftp/cert_advisories/CA-98.01.smurf. January 5, 1998.

[A-19] CERT Advisory CA-98.05. http://www.cert.org/ftp/cert_advisories/CA-98.05.bind_problems. April 8, 1998.

[A-20] CERT Incident Note. http://www.cert.org/incident_notes/IN-98.02.html. July 2, 1998.

[A-21] CERT Vulnerability Note VN-98.01. http://www.cert.org/vul_notes/VN-98.01.XFree86.html. May 3, 1998.

[A-22] Computer Emergency Response Team Website. http://www.cert.org.

[A-23] Cisco Systems, Inc. "NetRanger Intrusion Detection System Technical Overview," http://www.cisco.com/warp/public/778/security/netranger/ntran_tc.htm.

[A-24] COAST FTP site. ftp://coast.cs.purdue.edu/pub/tools/unix/satan/. 1998.

[A-25] Cunningham, R. K., R. P. Lippmann, D. J. Fried, S. L. Garfinkel, I. Graf, K. R. Kendall, S. E. Webster, D. Wyschogrod, and M A. Zissman, (1999) "Evaluating Intrusion Detection Systems without Attacking your Friends: The 1998 DARPA Intrusion Detection Evaluation," In Proceedings ID'99, Third Conference and Workshop on Intrusion Detection and Response, San Diego, CA: SANS Institute.

[A-26] Durst, S. and T. Champion. Packet Address Swapping for Network Simulation. Patent application, Air Force Research Laboratory. March 1999.

[A-27] Garfinkel, S. and G. Spafford, Practical Unix & Internet Security. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol CA, 95472, 2nd edition, April 1996.

[A-28] Heberlein, T., "Network Security Monitor (NSM)—Final Report," University of California/Davis, February 1995, http://seclab.cs.ucdavis.edu/papers/NSM-final.pdf.

[A-29] Heiberlein, L. T., G. V. Dias, K. N. Levit, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. In Proceedings of the 1990 Symposium on Research in Security and Privacy, pages 296–304, Oakland, CA, May 1990. IEEE.

[A-30] Heberlein, T. L., G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. "A Network Security Monitor," in 1999 IEEE Symposium on Research in Security and Privacy. pp. 296–304.

[A-31] *hobbit* hobbit@avian.org. Netcat README file. http://www.l0pht.com/~weld/netcat/readme.html. 1998.

[A-32] Icove, D., K. Seger, and W. VonStorch. Computer Crime: A Crimefighter's Handbook. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol CA, 95472, August 1995.

[A-33] Koral Ilgun. USTAT: A real-time intrusion detection system for UNIX. Master's thesis, University of California Santa Barbara, November 1992.

[A-34] Impack binaries on Rootshell.com. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199804/impack103.tar.gz.html. April 13, 1998.

[A-35] Internet Security Systems X-Force. http://www.iss.net.

[A-36] Jagannathan, R., T. Lunt, D. Anderson, C. Dodd, F. Gilham, C. Jalai, H. Havitz, P. Neumann, A. Tarnaru, and A. Valdes. System design document: Next-generation intrusion detection expert system (NIDES). Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA 94025, March 1993.

[A-37] Javitz, H. S. and A. Valdes, "The NIDES Statistical Component Description and Justification," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report, March 1994.

[A-38] Kemmerer, R. A. "NSTAT: A Model-based Real-time Network Intrusion Detection System," Computer Science Department, University of California, Santa Barbara, Report TRCS97-18, http://www.cs.ucsb.edu/TRs/TRCS97-18.html.

[A-39] Ko, C., M. Ruschitzka, and K. Levitt. "Execution Monitoring of Security-Critical Programs in a Distributed System: A Specifications-Based Approach," In Proceedings 1997 IEEE Symposium on Security and Privacy, pp. 134-144, Oakland, CA: IEEE Computer Society Press.

[A-40]Lawrence Berkeley National Laboratory, Network Research Group Homepage. http://www-nrg.ee.lbl.gov/. May 1999.

[A-41] Lawrence Livermore National Laboratory. "Network Intrusion Detector (NID) Overview," Computer Security Technology Center, http://ciac.llnl.gov/cstc/nid/intro.html.

[A-42] Lincoln Laboratory ID Evaluation Website, MIT, http://www.ll.mit.edu/IST/ideval/index.html. 1999.

[A-43] Lippmann, R. P., R. K. Cunningham, S. E. Webster, I. Graf, and D. Fried. Using Bottleneck Verification to Find Novel New Computer Attacks with a Low False Alarm Rate. Unpublished Technical Report. 1999.

[A-44] Lunt, T.F. "Automated Audit Trail Analysis and Intrusion Detection: A Survey," in Proceedings 11th National Computer Security Conference, pp. 65–73. 1988.

[A-45] NMAP homepage. http://www.insecure.org/nmap/index.html. 1998.

[A-46] Parras, P.A. and P.G. Neumann. "EMERALD: Event Monitoring Enabling Response to Anomalous Live Disturbances," In Proceedings 20th National Information Systems Security Conference, Oct 7, 1997.

[A-47] Paxon, V., "Bro: A System for Detecting Network Intruders in Real-Time," In Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, January 1998, http://www.aciri.org/vern/paper.html.

[A-48] Nicholas, P., K. Zhang, M. Chung, B. Mukherjee, and R. Olsson. A Methodology for Testing Intrusion Detection Systems. Technical report, University of California, Davis, Department of Computer Science, Davis, CA 95616, September 1995.

[A-49] Real Secure 2.5 User Manual, Chapter 6. Internet Security Systems, Atlanta, GA. (http://download.iss.net/manuals/rs25.tar.Z).

[A-50] Rootshell web site. http://www.rootshell.com. 1999.

[A-51] Rootshell web site. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199707/perl-ex.sh.html. August 26, 1996.

[A-52] Rootshell web site. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199707/solaris_ps.txt.html. June 11, 1997.

[A-53] Rootshell web site. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199707/fdformat-ex.c.html. March 24, 1997.

[A-54] Rootshell web site. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199711/sol_syslog.txt.html. November 6, 1997.

[A-55] Rootshell web site. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199801/beck.tar.gz.html. Jan 1, 1998.

[A-56] Rootshell web site. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199805/xterm_exp.c.html. June 4, 1998.

[A-57] Rootshell web site. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199806/mscan.tgz.html. June 24, 1998.

[A-58] Saint web site. http://www.wwdsi.com/saint. 1998.

[A-59] Staniford-Chen, S., S. Cheung, R. Crawford, M. Dilger, J. Frank, K. Levitt, C. Wee, R. Yip, D. Zerkle, and J. Hoagland. GrIDS—a graph based intrusion detection system for large networks. 1996. (available at http://seclab.cs.ucdavis.edu/arpa/grids/welcome.html).

[A-60] Sun Microsystems Security Bulletin: #00138. http://sunsolve.Sun.com/pub-cgi/us/sec2html?secbull/138. 17 April, 1997.

[A-61] Sun Microsystems Security Bulletin: #00140. http://sunsolve.Sun.com/pub-cgi/us/sec2html?secbull/140. 14 May, 1997.

[A-62] Sun Microsystems, Solaris Security Website. http://www.sun.com/solaris/2.6/ds-security.html. May 1999.

[A-63] Sundaram, A. "In Introduction to Intrusion Detection," Crossroads: The ACM Student Magazine, 2(4). 1996.

[A-64] Weber D. A Taxonomy of Computer Intrusions. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 02139, 1998.

[A-65] Webster, S. The Development and Analysis of Intrusion Detection Algorithms. Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, 02139, 1998.

[A-66] "Case Sensitivity Vulnerability," NT Security News, http://www.ntsecurity.net/scripts/loader.asp?iD=/security/casesensitive.htm.

[A-67] Netbus web site, http://www.netbus.com.

[A-68] Net Security web site, http://www.net-security.sk/bugs/NT/oob.html.

[A-69] Next Stop Software web site, http://nssoft.hypermart.net/.

[A-70] NT Security News, http://www.ntsecurity.net/.

[A-71] NTInfoScan home page, http://www.infowar.co.uk/mnemonix/ntinfoscan.htm/.

[A-72] "Sechole Lets Non-administrative Users Gain Debug Level Access to a System Process," Microsoft Product Support Services, http://support.microsoft.com/support/kb/articles/Q190/2/88.ASP?LN=EN-US&SD=gn&FR=0/.

[A-73] "Security Hole #1," Cybermedia Software Private Limited, http://www.cybermedia.co.in/espl21/nt_security/Sechole.htm.

[A-74] "Windows spoofing security bug," http://www.whitehats.com/browsers/b14/b14.html.

APPENDIX B
EXS SCRIPTS

1. OVERVIEW

An Expect program is used to automatically generate normal background traffic and attacks for the evaluation. This program takes what we have called "EXS scripts" as input. Each EXS script specified the session and commands that are to be run. This Appendix describes the format of these EXS script files, as well as giving some details about the simulation environment in which the scripts are run. This appendix summarizes date and time conventions for the simulation runs, naming conventions for the different file types, directories used for traffic generation, and the format of .exs files containing commands to execute.

2. FILE TYPES AND NAMING CONVENTIONS

The following file types are used for traffic generation.

*.exs—Contains "S"ource commands that are issued from the originating or source side of a TCP/IP connection. The format of these files is described below. Names of these files should indicate their function and be unique. For example, .exs scripts that generate mail from a PC vendor on one day could be named "vendor_mail001.exs," "vendor_mail002.exs," ... "vendor_mail345.exs." Scripts that generate normal user telnet sessions could be named "telnet001.exs," "telnet002.exs," ... "telnet043.exs." Scripts that generate system administration telnet sessions could be named "sysadmin_telnet001.exs," "sysadmin_telnet002.exs," ... "sysadmin_telnet007.exs."

***.exd**—Contains "D"estination responses that are fed back from the destination side of a TCP/IP connection. The format of these files is undefined.

***.exp**—Contains "P"rocedures that can be executed from .exs files when the comment before a command line is "# execute."

***.exm**—"M"aster file contains a list of all the .exs files to run on a given day. This is typically named "master.exm."

***.ex**—Normal ex scripts that are interpreted and run by the expect program.

3. DIRECTORY STRUCTURE ON LINUX ARTIFICIAL TRAFFIC GENERATOR MACHINES

Artificial traffic is generated on Linux workstations where the kernel has been modified such that the machine can simulate many different originating IP addresses. Traffic generation is organized on a day-by-day basis. All .exs and .exm scripts and other data required to generate one day of traffic are separated and then used during that day of traffic generation.

Each machine will have a /.sim/ main directory and all files required for traffic generation will be in subdirectories below this main directory. One idea was to make this directory invisible to artificial users who log into the traffic generator machines during the simulation by modifying commands such as df and ls in the /bin directories used by these users, however this was not implemented for the 1999 evaluation.

The following subdirectories will be provided underneath "/.sim."

/.sim/exs/week-day/—This directory contains all .exs and one master.exm file required to generate traffic for one day. The directory name will indicate the week and day of the simulation run where the week starts at 1 and the day goes from 1 to 7, where 1 is Monday. Some examples follow.

/.sim/exs/1-1/—This directory contains all .exs and the master.exm files for the first day (Monday) of simulated traffic.

/.sim/exs/3-5/—This directory contains all .exs and the master.exm files for Friday of the fourth week of simulated traffic.

/.sim/expect/—This directory contains all .ex scripts and .exp procedures that are used to generate traffic. These currently include launcher.ex, master.ex, random.exp, and mailreader.exp.

/.sim/data/—Subdirectories contain data used by .exs scripts to generate mail, ftp, and other types of traffic. Some examples follow.

/.sim/data/ngrams/—This directory contains Ngram files used by .exs scripts to generate mail, ftp, and other types of traffic.

/.sim/data/mail_lists/—This directory contains mail files sent to real mailing lists used by .exs scripts to generate mail, ftp, and other types of traffic.

/.sim/attacks/—This directory contains attack .ex scripts and programs used by .exs files to run attacks.

/.sim/home/—This directory contains the home directories of all real users on the traffic generation machines. Only the subdirectory of "/.sim/" will be backed up to tape each night.

4. FORMAT OF .exs FILES

The .exs files contain source commands that are issued from the originating or source side of a TCP/IP connection. At the beginning of each TCP/IP connection, the program which interprets .exs files takes on the identity of a user on the local machine specified in the header of the .exs file. Telnet sessions and local shells then are run as if that local user was typing in commands. Commands in .exs files start being issued either at the beginning of a telnet session that is started to a remote machine or to a local shell session. After each command is issued, the response from the remote machine is examined to see if it contains a desired prompt string. If the desired prompt string is returned, the next command is issued, otherwise the session terminates. The response from the remote machine is also examined to determine whether the command succeeded by looking for a small number of error return strings such as "No such file" or "cannot access." The behavior of the program when these error strings are returned is determined for each command by a flag which specifies whether to terminate the program when errors occur on the remote machine (e.g., the error wasn't expected), or to continue when an error occurs on the remote machine (e.g., the error was expected), or to blindly keep going even if errors occur on the remote machine (e.g., I don't give a hoot). This last option is not recommended.

The general format of the .exs file is a header which includes a list of prompts followed by a long list of commands. The .exs file is a UNIX text file using spaces to separate tokens on each line. Each .exs file must contain the following components:

Day and Time To Start

Duration

Local User Name and Password

Real Source IP

Real Destination IP

Virtual Source IP

Virtual Destination IP

Service

Prompt 1

Prompt 2

. . .

Prompt N

END_OF_PROMPTS

{ Prompt Index

{ Command

{ Time Of First Character

{ Time Of Last Character

{ Success/Fail/Ignore

END_OF_COMMANDS

5. DETAILS CONCERNING LINES IN .exs FILES

Day and Time to Start:

The simulation day and time during that day that the first command in this .exs file should be issued.

The format of this line is

<week (0-N)>/<day of week (0-6)>/<hr(0-23)>/<min(0-59)>/<sec(0-59)>.

Note that numbers less than 10 do not have to be preceded by a leading zero, that the week is unbounded and simply has to be zero or a positive integer, and that Monday is 0 for the day of week.

Some examples follow.

0/0/2/0/0—2:00 a.m. Monday morning, the first day

0/1/2/15/30—2:15 a.m. + 30 seconds Tuesday morning, second day

1/2/14/15/30—2:15 p.m. + 30 seconds Wednesday afternoon, second week

4/0/7/30/45—7:30 a.m. + 45 seconds Monday morning, fifth week

7/6/20/59/59—8:59 p.m. + 59 seconds Sunday night, eighth week

198/4/22/33/1—10:33 p.m. + 1 second Friday night, 199th week

Duration:

The expected duration of this .exs script taking into the account the time to run all commands.

The format of this line is

<hr(0-23)>/<min(0-59)>/<sec(0-59)>.

The format of these fields are the same as in the previous line except the week and day fields are omitted.

Some examples follow.

0/0/28—28 seconds

7/15/30—7 hours, 15 minutes, 30 seconds

14/59/1—14 hours, 59 minutes, 1 second

Local User Name and Password:

This is the account name of a local user. When the expect script is interpreted, commands will be issued as if this local user logged in and typed the commands.

Real/Virtual Source/Destination IP:

These are the IP addresses in normal notation (e.g. 192.168.0.20) which specify the real IP address of this machine (Source), the real IP address of the destination machine (Destination), the fake IP address to be used by this machine when it initiates the TCP/IP connection (Virtual Source), and the fake IP address which is the destination of the TCP/IP connection established for this .exs file (Virtual Destination).

Kernel modifications on this machine are used to create the Virtual Source IP address and IP aliasing on the Destination machine is used to create the Virtual Destination. When the TCP/IP connection is established only the Virtual Source and Destination IP addresses are used.

Service:

A name or number of the service being connected to. Numbers are more general, but names should be used for the common services to increase human readability of the .exs file. To date, we have only used "telnet" to establish a telnet connection to a remote machine and "local" to start up a shell on the local machine and issue commands to that shell.

The shell that starts up is that specified in the user's "$SHELL" environmental variable.

Prompt N:

A regular expression that matches a prompt returned by the destination machine. This is often simply the name of the machine followed by a colon or greater-than sign for telnet connections.

END_OF_PROMPTS:

A tag to signal the end of the prompt list.

Prompt Index:

A number from 0 to N. 0 indicates a "null prompt;" that is, the regenerator will not wait for any specific string. The numbers 1 thru N correspond to the first thru Nth prompts in the prompt list. If the prompt index is greater than zero, and the prompt is not in the string returned from the destination end of the TCP/IP connection after a timeout, the TCP/IP connection is closed, interpretation of the script stops, and an error message is written in the log file.

Command:

The command that is typed. Characters are escaped as described below. The command is assumed to end with a CR. If the command should not end with a CR, then the line should end with a single backslash.

Time of First Character:

Time of Last Character:

Both real numbers in seconds. The first is the pause between the regenerator seeing the prompt and starting to type. The second is the approximate time it will take the regenerator to type in the command.

Success/Fail/Ignore:

The regenerator will read the first character of this line for an "S," an "F," or an "I." An "S" implies that the command did not fail; an "F" implies that it did. "I" implies that the regenerator should not attempt to determine success or failure. A command is considered to have "failed" if the response back from it contains "o such file" or "cannot open." The regenerator will report differences. If a command fails when it was supposed to succeed, or if it succeeds when it is supposed to fail, the TCP/IP connection is closed, interpretation of the script stops, and an error message is written in the log file.

END_OF_COMMANDS:

A tag to signal the end of both the command list and the .exs file.

Escape Sequences:

All non-printable characters (those less than 32 or greater than 126) should be replaced by "\aaa" where "aaa" is the octal code. DEL (decimal 127) is replaced by "\177." A literal backslash is replaced by a double backslash.

That is, every backslash (not immediately preceded by another backslash) shall be followed either by a backslash, three octal numbers, or a carriage return.

Comment:

One or more comment lines can be added on the line immediately before the "prompt index" line. All prompt lines must begin with a "#" as the first character.

Comment lines serve three different purposes.

1. The first comment in a file provides side information to other programs which determine whether the simulation ran correctly.

2. General comments help a user understand what is going on in a .exs file.

3. A special comment containing only the word "execute" indicates that the following command is not a command. Instead it is an expect procedure defined in a .exp procedure file. Expect will call this procedure instead of sending the command out normally.

The first comment in each file should contain two tokens indicating first whether this is normal background traffic, an attack, or an anomaly ("normal," "attack," or "anomaly"). The second token should be a string representing the primary TCP/IP service used by the .exs file (e.g., telnet, rlogin, smtp, http, finger). These two tokens should be followed by the attack name for attacks, by the anomaly type for anomalies, and and by other descriptive information for normal traffic.

The following are some examples of the first comment.

normal telnet programmer reads outside mail

normal smtp programmer sends mail to manager

attack http phf grab passwd file from external web site

normal telnet secretary edit files

anomaly telnet secretary behaves like programmer

The special comment containing only the word "execute" indicates that the following command is not a command. Instead it is an expect procedure defined in a .exp procedure file. The first token on the command line must be the name of the procedure and the remaining tokens are optional arguments. All arguments are passed to the expect procedure one at a time.

6. AN EXAMPLE OF A .exs FILE

1/2/14/15/30

0/2/32

172.16.112.10

172.16.112.50

31.168.25.47

31.168.193.58

rpl

telnet

login:

```
password:
fred%
subject:
END_OF_PROMPTS
# normal telnet a simple demonstration .exs file
# the user name for login
1
fred
0.5
0.5
Succeed
# a weak password for login
2
gandalf
0.2
0.7
Succeed
# look at a file that is really there
3
cat random.c
3.4
0.86
Succeed
# this should fail because the file isn't there
3
cat gobblygook.abc
8.2
5.6
Fail
# all done, but it takes a while to decide to logout
3
```

```
    logout
    134.6
    0.80
    Ignore
END_OF_COMMANDS
```

APPENDIX C
EYRIE AFB SECURITY POLICY

The security policy adopted by the base eyrie.af.mil is a loose one. No services are blocked for connections between the air force base and the outside. Most users inside and outside the base have accounts on only one machine. Some users have accounts on several machines both inside and outside the base. Some of these users are usually situated on the outside and may connect to their inside accounts if they want access to the inside network. Others users are situated inside the base and sometimes connect to their outside accounts. A subset of these users are system administrators who know the root password. These system administrators often conduct their administrative work and monitoring remotely. They typically log in from .mil machines on the outside, but they may come from other machines on occasion. It is recommended policy that system administrators telnet with their user names and then run su from the telnet session. However, it is not a violation of security if an administrator telnets as root. System administrators have the right to add users or delete users, and the right to add machines and take machines away. Users may download publicly available material from anonymous ftp sites and browse any web sites they wish. One outside machine will be running SNMP on a regular basis to monitor the health of systems on Eyrie Air Force Base. The internal subnet with IP addresses 172.16.112.* is protected. The only valid IP addresses allowed on this subnet are those specified in the List of Simulation Network Hosts. It is illegal to add another host onto this subnet.

There is a secret directory set up on the machines under /home/secret which contains highly confidential information. Only users in the secret group (abramh, elmoc, quintond, orionc) are allowed access to the secret files under this directory. When reading or writing to these files over the network, these users must use ssh rather than telnet in order to prevent the files being sent as clear text. Any transferring of these files to another computer is strictly illegal.

Typical hacks are not allowed by any users. All of the following activities are illegal and fall under the category of an attack:

- Disrupting system or network functioning
- Probing for vulnerabilities
- Obtaining root privileges through dubious means
- Gaining access that the user is not privileged to have
- Illegally modifying or accessing data not owned by the user
- Installing or using previously installed back doors or trojan horses
- Unauthorized uploading of data to anonymous ftp sites
- Downloading of illegally uploaded data from anonymous ftp sites
- Unauthorized using of SNMP
- Transferring any information that has been illegally obtained
- Transferring any data that is accessible only by the secret group, even by a member of the secret group
- Preparations required to carry out any of the above illegal actions
- Actions made possible by any of the above illegal events
- Installing or running a sniffer to capture network traffic

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	26 February 2001	Technical Report

4. TITLE AND SUBTITLE

1999 DARPA Intrusion Detection Evaluation: Design and Procedures

5. FUNDING NUMBERS

C — F19628-00-C-0002

6. AUTHOR(S)

J.W. Haines, R.P. Lippmann, D.J. Fried, M.A. Zissman, E.Tran, and S.B. Boswell

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02173-9108

8. PERFORMING ORGANIZATION REPORT NUMBER

TR-1062

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DARPA, ISO
2701 North Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ESC-TR-99-061

11. SUPPLEMENTARY NOTES

None

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

Recent DARPA Intrusion Detection (ID) and Strategic Intrusion Assessment (SIA) programs have funded development of new approaches to intrusion detection. The Information Systems Technology Group at MIT Lincoln Laboratory assisted this research with off-line evaluations of these new systems in 1998 and 1999. These evaluations measured detections and false alarm rates of the intrusion detection systems. Eight research sites participated in the second annual evaluation. A network testbed was developed for this evaluation. It included host computers that were attacked and recently-developed traffic generators that produced live traffic modeled after a small Air Force base. This traffic appears as if it were generated by hundreds of users and thousands of hosts. More than 200 instances of 58 attack types were launched against victim UNIX and Windows NT hosts in three weeks of training data and two weeks of test data. Objectives of this effort were to support algorithm development, perform a blind, off-line evaluation of intrusion detection approaches, and help DARPA guide research directions. This technical report describes the testbed design and operation, background traffic modeling and generation, attack modeling and automation, and the scoring procedure. Results of the 1999 evaluation are discussed in a separate technical report entitled "Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation."

14. SUBJECT TERMS	15. NUMBER OF PAGES
	198
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Same as Report	Same as Report	Same as Report