IJIS (2001) 1: 14–35 / Digital Object Identifier (DOI) 10.1007/s102070100001

Intrusion and intrusion detection

John McHugh

CERT® Coordination Center*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890, USA
E-mail: jmchugh@cert.org

Published online: 27 July 2001 – © Springer-Verlag 2001

Abstract. Assurance technologies for computer security have failed to have significant impacts in the marketplace, with the result that most of the computers connected to the internet are vulnerable to attack. This paper looks at the problem of malicious users from both a historical and practical standpoint. It traces the history of intrusion and intrusion detection from the early 1970s to the present day, beginning with a historical overview. The paper describes the two primary intrusion detection techniques, anomaly detection and signature-based misuse detection, in some detail and describes a number of contemporary research and commercial intrusion detection systems. It ends with a brief discussion of the problems associated with evaluating intrusion detection systems and a discussion of the difficulties associated with making further progress in the field. With respect to the latter, it notes that, like many fields, intrusion detection has been based on a combination of intuition and brute-force techniques. We suspect that these have carried the field as far as they can and that further significant progress will depend on the development of an underlying theoretical basis for the field.

Keywords: Computer misuse – Intrusion detection – Intrusive anomalies – Intrusion signatures – Intrusion detection systems (IDS) – IDS evaluation

1 Introduction

> The principal unsolved technical problem found by the working group was that of how to provide multilevel resource and information sharing systems secure against the threat from a malicious user. This problem is neither hopeless nor solved. It is, however, perfectly clear to the panel that solutions to the problem will not occur spontaneously, nor will they come from the various well-intentioned attempts to provide security as an add-on to existing systems.

Although these words could have been written today, they come from one of the seminal documents in computer security, a report [3, Preface] prepared by James P. Anderson & Co. for then Major Roger R. Schell of the USAF in 1972. In the nearly 30 years since the preparation of the Anderson report, little has changed, except that we are, perhaps, less sanguine about the solvability of the problem. The line of research and development proposed in the report has produced a few fairly secure systems, but none that have achieved commercial success or wide-scale deployment. The problems posed by malicious users are rampant, and the inability of commodity operating systems to provide more than minimal protection has led to a variety of attempts to secure computing systems through add-on or external means. Firewalls and similar mechanisms form the principle line of defense for many installations. Intrusion Detection Systems (IDSs), the primary topic of this paper, are an attempt to identify both successful and unsuccessful attempts to abuse computer systems.

The paper begins with a discussion of intrusive activities and the kinds of flaws or vulnerabilities in computing systems that enable them. This is followed by a historical overview of the intrusion detection field, illustrated with descriptions of systems of both historical and current interest. With the historical view to give perspective, the paper takes an in-depth look at the technologies involved in intrusion detection, considering both their strengths and weaknesses. This is illustrated with descriptions of current commercial, research and public-domain systems. The problems associated with evaluating IDSs are considered briefly. The paper concludes with a discussion of the current state of the intrusion detection technology and its prospects for future improvement.

* CERT and CERT Coordination Center are registered service marks of Carnegie Mellon University

2 Intrusions, intrusive activities, penetrations, and exploits

From the earliest days of computer security, the possibility that malicious users could defeat protection mechanisms was an area of serious concern. Due to the relatively limited networking of early systems and the prevalence of multiuser batch systems, coupled with the fact that publicly accessible services (such as present-day web servers) were almost unknown, most of the early efforts concentrated on mechanisms that untrusted insiders could use to access sensitive materials on multi-level secure systems.[1] Under this model, the primary threat is from legitimate users of the system who try to gain access to material for which they do not have authorization. By the early 1970s, timesharing and other multiuser systems were well established and there were growing pressures to support multi-level operations on these systems, whether they were actually capable of supporting them or not. In 1970, the Defense Science Board's Task Force on Computer Security supported by Willis H. Ware of the RAND Corporation issued a report [81] that laid the groundwork for the development of adequate computer systems for the processing of multi-level data. As noted in the Anderson report [3], the Ware report did not get the attention it deserved,[2] and systematic approaches to building multi-level secure systems did not emerge until the mid 1970s.

Early multi-user operating systems provided minimal security features and those that were present were often directed towards protecting the data associated with one task from accidental corruption by another. While moderately successful in this respect, such mechanisms were often trivial to defeat. For example, in the mid-1960s, the resident engineer for a site where the author was employed gave him a list of about a dozen techniques that a job could use to gain supervisor mode with complete addressability to all physical memory on systems running the IBM OS-360 MFT operating system.

In the meantime, security flaws that could be exploited by untrusted users were discovered in a number of systems that were considered to be secure. Paul Karger and Roger Schell performed an evaluation [44] of the Multics operating system and discovered flaws that would allow an untrusted user to access or modify protected information. Work at the Naval Research Laboratory [74] uncovered flaws in the sharing mechanisms of Univac's Exec-VIII operating system for the 1108 that would allow users of a language processor such as the FORTRAN compiler to capture files from other concurrent users of the same processor. Experiences such as these led to systematic investigations of flaws and exploitations as well as techniques to design and build secure systems.

Although there are earlier discussions of the issues associated with malicious users, James P. Anderson's 1980 report "Computer Security Threat Monitoring and Surveillance" [5] set up the first coherent framework for an investigation of intrusions and intrusion detection. We will use the following definitions, given by Anderson in this paper, supplementing them later, as necessary.

Threat:	The potential possibility of a deliberate, unauthorized attempt to: (a) Access information (b) Manipulate information (c) Render a system unreliable or unusable
Risk:	Accidental and unpredictable exposure of information, or violation of operations integrity due to malfunction of hardware or incomplete or incorrect software design.
Vulnerability:	A known or suspected flaw in the hardware or software design or operation of a system that exposes the system to penetration or its information to accidental disclosure.
Attack:	A specific formulation or execution of a plan to carry out a threat.
Penetration:	A successful attack; the ability to obtain (unauthorized) access to files and programs or the control state of a computer system.

Note that threat class (c) includes what are commonly called "denial of service" attacks today. Attacks that misappropriate computing resources also fall into this category.

Anderson classifies threats as shown in Fig. 1. The first task faced by an external penetrator is to gain access to the system in question. Note that the true external penetrator may be either an outsider with no connection to the organization that owns or controls the system being attacked or it may be someone associated with the organization who is not authorized to use the system. In today's world of networked systems, it could also be someone who has legitimate access to systems on the network, but not to the target of the attack. While there are certain classes of attacks, notably some denial of service attacks, that do not require the penetrator to actually become a recognized user of the attacked system, most attacks aimed at either accessing or manipulating information require the penetrator to take on a user role, even though it may be limited to using (and abusing) a public service offered by the system. Anderson further characterizes internal penetrators as masqueraders, legitimate users, or clandestine users.

[1] A multi-level secure computing system is one that is capable of supporting a mandatory access control policy that bases access decisions on the classifications assigned to the information objects that it stores and clearances given to users on whose behalf processes seek access.

[2] Although the Ware report contained no sensitive information, it was classified at the CONFIDENTIAL level to control dissemination of its contents. This, combined with its emphasis on stating requirements rather than suggesting solutions probably reduced its impact substantially.

	Penetrator Not Authorized to use Data/Program Resource	Penetrator Authorized to use Data/Program Resource
Penetrator Not Authorized Use Of Computer	Case A: External Penetration	
Penetrator Authorized Use Of Computer	Case B: Internal Penetration	Case C: Misfeasance

Fig. 1. General cases of threats (after [5])

Masqueraders are internal users by definition, but they may be successful external penetrators who have assumed an internal identity or legitimate users who assume the identity of another for whatever reason. Penetrations by legitimate users typically involve abuses of the privileges that are necessary to carry out their regular duties. Clandestine users are typically highly skilled in the technical aspects of penetration and take steps to erase or disguise the traces of their penetrations. They typically achieve complete control of the penetrated computing system and modify its operating system so as to make subsequent discovery very difficult.

Early descriptions of vulnerabilities typically assume that the penetrator has obtained an internal identity and is capable of writing programs to exploit the vulnerability. Two vulnerabilities from the early 1970s serve to illustrate the approaches of the time.

2.1 Multics

In performing a vulnerability analysis of Multics, Karger and Schell found a number of flaws that might be exploited by penetrators. One of these allowed ordinary users to change the access time stamps on segments allowing clandestine users to cover their tracks. Another was more subtle and serves to illustrate ways in which vulnerabilities are introduced and propagated, even in systems that have security as a strong design goal [44, P. 20].

> While experimenting with the hardware subverter,[3] a sequence of code was observed which would cause the hardware of the 645 to bypass access checking. Specifically, the execute instruction in certain cases described below would permit the executed instruction to access a segment for reading or writing without the corresponding permissions in the SDW. . . .
>
> This hardware bug represents a violation of one of the most fundamental rules of the Multics design – the checking of every reference to a segment by the hardware. This bug was not caused by fundamental design problems. Rather, it was caused by carelessness by the hardware engineering personnel.

The bug was introduced as a field change thus escaping the design analysis processes associated with the Multics hardware and software development.

2.2 Exec-VIII

Exec-VIII was a multiprocessing operating system for the Univac 1108 series of mainframes. Univac claimed that Exec-VIII was secure in the sense that it enforced an isolation policy that prevented one job from accessing resources belonging to another job. On the strength of these claims, some government agencies considered use of Exec-VIII in environments with a mix of classified and unclassified computing.

Executing programs on the 1108 used two address segments, one for code and one for data by convention. By making the code reentrant, code for a single program, such as a compiler, could be shared among a number of simultaneous jobs. Univac assumed that the code segments were immutable, but the segmentation hardware provided no mechanism for enforcing this while allowing the necessary write access to the corresponding data segment. Mike Lay of the University of Maryland Computer Science Faculty discovered a mechanism [74] whereby a re-entrant processor (or REP) could be forced to transfer control to an error handler in a user program that would then be able to modify the core image of the REP. Code to carry out the penetration was developed by David Stryker of NRL. The code inserted in the modification could, for example, access or modify any files owned by any user who invoked the REP. Once modified, the REP would persist in its modified state until no jobs were using it. Since REPs are never swapped out, the exploit would have to be repeated periodically, however, this was done by having the breaking code awaken periodically, determine if the target REP was still modified, and repeat the modification, if necessary.

Both the 1108 and the Multics penetrations required deep knowledge of the relevant hardware and software features of the targeted systems. While none of the Multics flaws were carried beyond the demonstration stage, Stryker's penetration appears to have been reduced to a fairly simple tool that could have been (but was not) widely distributed and used by relatively unskilled insiders.

[3] The hardware subverter was a program that periodically probed security sensitive and potentially troublesome hardware instructions on the host hardware. It looked for hardware failures and other anomalies. In some 1100 h of operation, the subverter discovered one undocumented instruction and the access checking failure described in the quotation.

It was not until the mid-to-late 1980s that networked computing became sufficiently ubiquitous for penetrations to become widespread. Network-wide penetrations were a subject of Science Fiction in the 1970s, John Brunner's 1975 novel [12] "The Shockwave Rider" being one of the best known examples. Early research work at highly networked enclaves such as Xerox PARC developed programs [69] that may have served as the prototypes (or at least inspiration) for similar malicious efforts during the late 1980s. Many of the penetrations or intrusions of the early-to-mid-1980s were enabled by gross negligence on the part of vendors and system administrators. Many vendors shipped systems with predefined administration or maintenance accounts (User name: system, Password: manager was used by one vendor) and many system operators neglected to change the passwords on these accounts. The widespread use of the Unix operating system among academic sites contributed to the problem since the BSD Unix networking model created large networks of trusted peer machines, simplifying the spread of compromises from one machine to another. A 1987 note by Brian Reid [65] describes an episode of break-ins that originated with a machine serving as a mail gateway between Unix and IBM systems at Stanford University. This machine had a guest account (User name: guest, Password: guest) as well as a directory `/usr/spool/at` that was universally writable. Commands placed in this directory could easily be made to execute with superuser privileges. The intruder was thus able to assume the identity of any user on the gateway machine, and by taking advantage of peer accounts on other machines was able to masquerade as that user on the peers. Reid estimates that 30 to 60 machines at Stanford were compromised and that at least 15 Silicon Valley companies, nine universities, and three government laboratories were penetrated in this way.

Another line of intrusive activity began in the mid-1980s. Computer viruses spread through floppy disks apparently appeared on Apple II computers as early as 1981 [70]. In 1985, Fred Cohen published a book [19] entitled "Computer Viruses". The first MS-DOS viruses appeared in 1986 and virus writing (and shortly after the development of anti-virus products as well) became a growth industry that persists to this day. Until fairly recently, most PC-class machines were not networked and early viruses spread primarily through physical contact involving infected floppy disks. Infections via the downloading of infected executables from dial-up bulletin boards was a secondary infection mechanism. Until the advent of Windows macro and scripting viruses in the late 1990s, virus penetrations were largely considered disjoint from more traditional intrusions, and we will not consider them further at this time.

In addition to penetrations enabled by poor administrative practices, direct attacks on system software and services in the spirit of the earlier Multics and Exec-VIII attacks continued to be developed in the late 1980s. The first "race condition" exploit took advantage of the fact that file creation on Unix is not an atomic action but requires separate system calls to create an "i-node" for the file and to link it into the directory structure; this apparently took place in the summer of 1988. It is possible for a process other than the one creating the file to gain access to the i-node if a context switch between the creating and intervening process occurs between the two system calls. Robert T. Morris, Sr., then Chief Scientist at the National Computer Security Center, hypothesized the exploit, but doubted its feasibility. His son, Robert T. Morris, Jr., implemented the exploit [56] on a machine in the Morris home.

On 2 November 1988, the first internet-wide attack and penetration occurred. This attack took the form of a self-replicating program or "worm" that propagated itself from system to system using a variety of transmission techniques. The worm is attributed to Robert T. Morris, Jr., who was prosecuted and convicted of federal crimes in connection with the incident. The "Morris Worm," as it is now known, exploited a misconfiguration in the `sendmail` program that allowed mailed commands to be executed on a remote system. The feature used was intended for debugging the `sendmail` program, but was enabled in binary versions distributed by several vendors. In addition, the worm also spread by overflowing a string variable in the finger daemon, a service widely available on Unix systems that allowed outsiders to obtain information about system users. The worm code also contained an efficient password cracking program that attempted to determine user passwords on systems that it infected. It used the trusted peer relationships reflected in .rhosts files to spread itself as well. Details of the worm and its impact on the internet appear in a number of publications, including papers by Spafford [73] and Seely [68]. It is worth noting that none of the penetration mechanisms used by the Morris Worm can be considered as security flaws, per se. Reid had already commented [65] on the folly of the peer trust represented by the .rhosts mechanism. The `sendmail` misconfiguration was due to the extreme difficulties involved in configuring `sendmail` properly, reducing it to a trial-and-error process that was facilitated by the ability to execute remote commands. The overflow potential in `fingerd` reflects a software engineering failure of a kind that persists to this date.

In response to the Morris Worm, DARPA established the Computer Emergency Response Team, now known as the CERT Coordination Center of CERT/CC at the Software Engineering Institute (SEI). The CERT advisories[4], which started in late 1988, allow major thrusts of intrusive activities on the internet to be traced.[5] From

[4] Available online at `http://www.cert.org`.

[5] More recently, Mitre, in cooperation with the CERT/CC and others, has established a comprehensive listing of Vulnerabilities and Exposures, the CVE, in an effort to foster the use of a standardized vocabulary by the security community. See `http://cve.mitre.org` for more information.

1988 through late 1996, there appear to have been relatively few intrusions enabled by buffer overflows. In late 1996, the online magazine *Phrack* published a cookbook for buffer overflows [59] and the number of such incidents rose dramatically. A search of the CERT advisories shows buffer overflows becoming increasingly important, starting in early 1997. A keyword search of the CVE for the term "buffer overflow" results in about 25% of the entries and candidate entries[6] being identified. A substantial percentage of the incidents reported to the CERT/CC during the past few years have involved a buffer overflow exploit of some kind. There are several reasons for the popularity of these exploits:

- The vulnerabilities are ubiquitous, as seen above. This increases the likelihood that attacks against random targets will succeed.
- A successful attack has a high probability of yielding administrator or superuser privileges on the target, since the subverted process typically runs with these privileges.
- Exploit scripts or programs for these attacks are readily available on the internet and require minimal skills to execute.

As is the case with many other classes of attacks, buffer overflows have evolved in complexity and sophistication. Early attacks involved data explicitly read by the target program. Subsequently, it was discovered that variables used to hold the values of environment variables could also be used as attack vectors. Recently overflows based on format strings have been used.

In 1992, the first widespread appearances of what are now known as "root kits" started to appear [15]. Intruders would gain access to a system as an ordinary user, typically by obtaining a password to a user account through guessing, social engineering, or other means. They would then attempt to become root by exploiting a vulnerability on the system. Once root access was obtained, subverted versions of various system utilities such as `su`, `ftp` and `ftpd` would be installed and remote access permissions enabled to facilitate subsequent reentry by the intruder. This line of intrusion has been refined over time with the addition of numerous modified utilities intended to hide the intruder's activities from other users, administrators and auditors.

Often, substantial time elapses between the discovery of a potential vulnerability and the development of an exploit. Bellovin hypothesized TCP hijacking in 1989 [11], but exploits did not actually appear until 1995 [16]. As noted above, Morris demonstrated a race condition vulnerability in 1988, but the first widespread incident of this kind apparently occurred in 1991 [14].

Figure 2 illustrates the increasing sophistication of attacks from the mid-1980s to the present. As the attacks

[6] The search was made against version 20001013 of the CVE which contains 1077 entries and 678 candidate entries. Of these, 434 or 24.7% contain the terms "buffer" and "overflow".

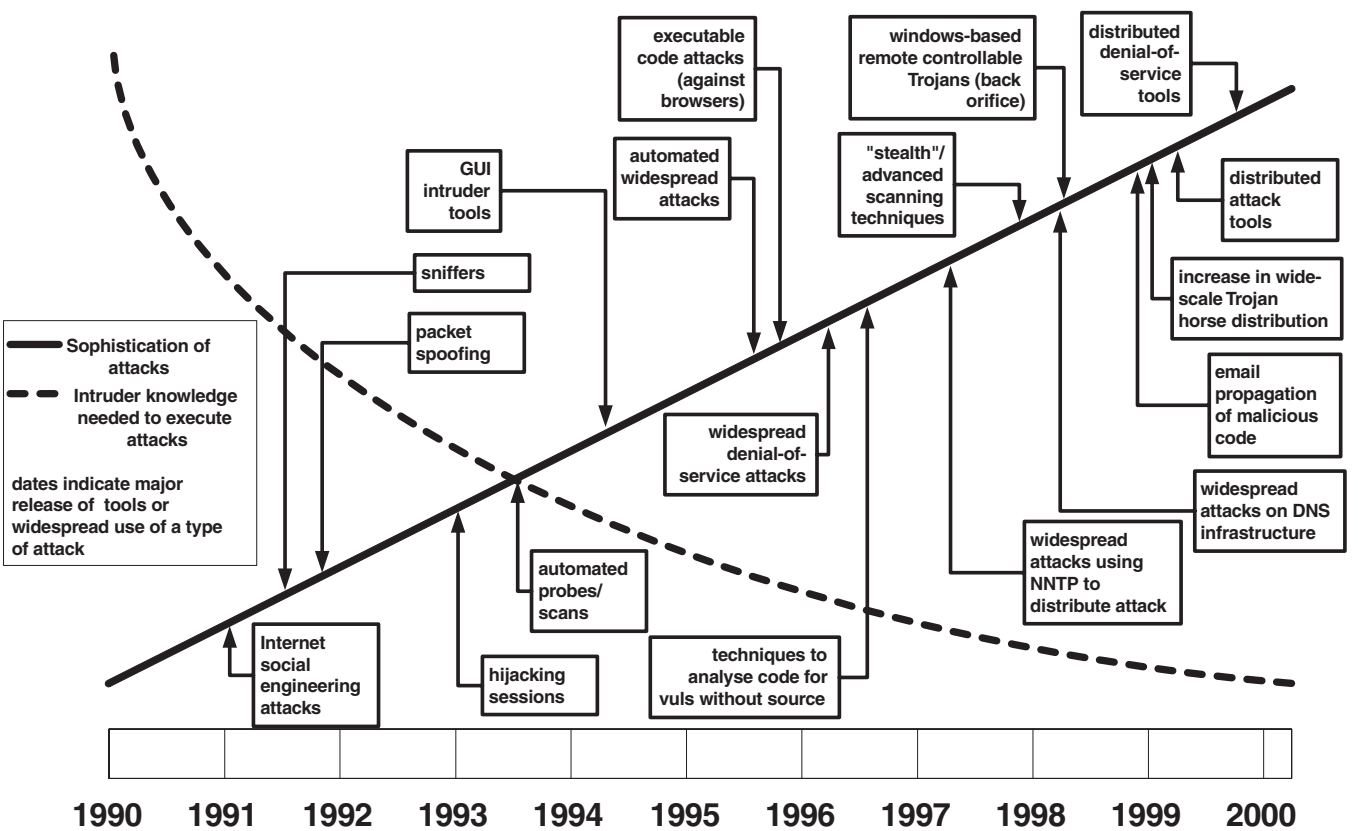

Fig. 2. The evolution of attack sophistication and devolution of attacker skill

have grown in complexity, they have been increasingly automated. This has reduced the skill required to launch the attacks. Recent research using CERT/CC data [7] indicates that this automation may be the trigger for large-scale intrusive activity on the internet.

Over the years, there have been a number of attempts to create taxonomies for classifying vulnerabilities. One of the most complete was developed by Krsul [48]. Krsul notes that about 63% of the vulnerabilities in his database resulted from invalid assumptions made by the programmer about the environment in which the program will run. A recent examination of the CERT/CC vulnerability database found that 58% of the vulnerabilities it contains have a similar origin [37], a figure that is in good agreement with Krusl. Interestingly enough, many of the Multics vulnerabilities found by Karger and Schell fall into this category, indicating the persistence of the problem. Of the remaining vulnerabilities in the CERT database, configuration problems, either with default configurations that are never changed or due to the complexity of configuring the system securely, account for another 10%, while inherently insecure protocols contribute about 5%. Implementation errors account for only about 2%.

3 The early history of intrusion detection

The history of an area of endeavor ultimately becomes a trace of recorded (usually written) documents. In the early stages, scholars may be able to draw on direct accounts from participants as well as on the writings of those who were present or involved in the field from the start. The author is fortunate to know many of the early visionaries and practitioners in the intrusion detection field. In addition to first-hand accounts and the (sometimes obscure) writings of these individuals, recent books, such as Rebecca Bace's excellent "Intrusion Detection" [10] contain a wealth of material on and references to the early developments in the field. In preparing this survey, I have relied on Bace's book and the sources she cites, supplemented by accounts from some of the early participants in the field.

In the beginning, there was audit. Financial systems have been designed to provide information that allows a specialized auditor to inspect the records with an eye to detecting fraud and error. As computers replaced ledgers in the 1950s and 1960s, it was natural to include auditing functionality as part of the programs. By the mid-to-late 1960s financial auditing of computer programs was well established. Since computer time was expensive, it was an obvious and logical extension to collect audit information on the usage of the computer as well. With the advent of multiuser systems, detailed records of user resource consumption were required to determine how to spread the computer costs among users in an equitable fashion.

The Ware report [81] defines requirements for auditing, noting that activities such as accessing files, changing their contents or classification, etc., must be recorded. Ware also notes that trusted individuals such as security administrators, system operators, and maintenance personnel should have their actions audited along with those of normal users. Anderson's 1972 Computer Security Technology Planning study also called for the provision of a security surveillance system [4, PP. 51–52] for the collection of data that may indicate attempted or actual security violations. The questions raised by Anderson – what to detect, how to analyze it, and how to protect the surveillance system and its data from attack – remain at the heart of IDS research today.

The late 1970s produced a number of government-led computer security initiatives, both within DoD (Department of Defense) and at NIST (National Institute of Standards and Technology). Security audit was a consideration in these studies. In 1980, James Anderson produced another report [5], this time for an Air Force customer who processed large amounts of classified data using mainframe computers. The customer's security staff performed a detailed manual review of audit data, looking for information that would indicate security violations. As the volume of processing increased, this task consumed excessive amounts of time. The task was also complicated by the facts that some necessary information was not captured, lots of unnecessary information was captured and some information was captured many times. Anderson introduced the threat taxonomy that is illustrated in Fig. 1 and categorized internal intruders as masqueraders, legitimate users, or clandestine users. The report then goes on to characterize computer usage and to define the sorts of activities that should be recorded for surveillance purposes. He then suggests ways in which the amount of data to be analyzed can be reduced, suggesting that statistical data on user and group behavior can be compared to summary observations, i.e., counts of activities, to detect abnormal behavior. The security officer would be alerted when either a clear security violation (or attempted violation) occurred or when an abnormal statistic was computed. The detailed observations would be available for subsequent evaluation by the security officer. The Anderson report served as a blueprint for early work at SRI and at TRW [6], and much of the work in intrusion detection carried out through the mid-1980s was strongly influenced by his approach.

3.1 The early 1980s

Bace [10, Chapt. 1] provides a summary of the most significant projects of the 1980s. A 1988 survey by Lunt [53] also covers many of the same systems, giving somewhat more detail and supplying additional references. From 1984 to 1985, Sytex conducted an audit analysis project for SPAWAR (U.S. Navy). The project was based on shell-level audit data from a Unix system and demonstrated that this data was capable of discriminating between "normal" and "abnormal" usage. Teresa

Lunt worked on the project at Sytex and later went to SRI, where she joined and led the IDES project which had been started in 1984 by Dorothy Denning and Peter Neumann. The IDES (Intrusion Detection Expert System) project at SRI was one of the most significant (and persistent[7]) of the early IDS research efforts. The SRI work resulted in Dorothy Denning's seminal paper (discussed below) first presented at the IEEE Security and Privacy Symposium in 1986 [24] and subsequently elaborated upon in 1987 [25]. The IDES model is based on the assumption that it is possible to establish profiles to characterize the normal interactions of subjects (typically users) with objects (typically files, programs, or devices). The profiles are actually statistical models of subject behavior and the system attempts to detect anomalous behaviors, i.e., those sufficiently unusual with respect to the profile to be considered suspect. The profiles have both static (long-term stable) and dynamic properties (computed over relatively short intervals) to allow the system to accommodate changing user behavior. The profiles are supplemented by an expert system that uses a rule base that describes activities that represent known security violations. This prevents a user from gradually training the system to accept illegal behaviors as normal.

TRW was involved in two, apparently unrelated, intrusion detection efforts. One, performed for a government customer [53, Sect. 3.4], used an expert system with the knowledge base coming from the expertise of seasoned system security experts provided by the customer. The system was applied to existing audit data and mimicked the behavior of human security officers, building a case against the intruder. In early tests, it performed well, finding the one planted intrusion in a set of actual audit data as well as identifying a previously undetected problem.

While the previous projects were motivated by the problems associated with protecting classified material in a DoD environment, the customer for TRW's Discovery [75] system was internal. Discovery was directed towards detecting intrusions and misuse of the online database that supported TRW's credit system. In particular, the system was intended to discover unauthorized inquiries. Audit data was collected from the some 400 000 inquiries made on the system each day and processed in batch mode using a hybrid system that provided both statistical and expert system rules. Discovery represents the first applications-based IDS where intrusions against an application as opposed to a platform or network are sought.

3.2 IDES and Denning's model

Dorothy Denning's 1987 paper [25] "An Intrusion Detection Model" reflects the work done at SRI in the early-to-mid-1980s.[8] Although the paper has been highly influential, reading it now is somewhat disappointing, primarily because many of the questions that it raises (and leaves unanswered) remain unanswered to this day.

As was the case with many early workers in the IDS field, Denning assumes that intrusive activities are distinct from normal activities and that the IDS task is largely the discovery of appropriate models for normal behavior so that intrusive activity can easily be distinguished. Much of the motivation in the introduction to the paper is speculative and, although intuitively appealing, offered without supporting evidence. The list of examples given includes the following:

> *Leakage by legitimate user:* A user trying to leak sensitive documents might log into the system at unusual times or route data to remote printers not normally used.
>
> *Inference by legitimate user:* A user attempting to obtain unauthorized data from a database through aggregation or inference might retrieve more records than usual.

Denning then goes on to describe the six primary components of the model: subjects, objects, audit records, profiles, anomaly records and activity rules. The profiles characterize subject behavior and are the key to the detection aspects of the model. Activity rules describe the actions to be taken by the system when certain conditions are recognized. They can "...update profiles, detect abnormal behavior, relate anomalies to suspected intrusions and produce reports". Audit records are triggered by an action and record the subject attempting the action, the action, the object targeted by the action, any exception condition which may have resulted, the resource consumption of the action and a uniquely identifying time stamp. Audit records are compared to the profiles and, using the appropriate rules, events that correspond to abnormal conditions are identified. The model is purely operational and has no explicit knowledge of security mechanisms or of vulnerabilities and exploits.

The measures that enter into the profiles are relatively simple consisting of event counters (e.g., logins per hour or commands per session), interval timers (e.g., time between compile commands or time between logins), and resource usage measures (e.g., pages printed per day or cpu usage per session). There are a variety of models based on these measures that can be part of a profile. The simplest may represent fixed limits. For example, more than four password failures for a given account in a 5 min period may indicate a guessing attack. Other models reflect statistical properties of single measures.

[7] IDES led to NIDES, which led to today's Emerald System. These subsequent systems are discussed further in Sect. 4.8.1.

[8] The original submission of the paper to TSE occurred in December 1985. This was probably nearly concurrent with the submission to the 1986 IEEE symposium [24], where the work was presented in May. The TSE version was revised in August 1986, which represents the latest date that can be assigned to the work.

For example, a user may remain logged in an average of 37 min per session with a standard deviation of 10 min. Assuming that the distribution is fairly normal, sessions longer than 57 min or shorter than 17 min should occur less than 5% of the time. Multivariate statistical, Markov process and complex time series models are also possible. The profile mechanism allows models to be kept at the level of individual users and to be aggregated to create classes of users. This allows detection of users whose actions are internally consistent, but at odds with those of peers in similar positions. The aggregations can be performed to aggregate the objects affected by actions as well. Subject-class aggregation might group privileged users while object-class aggregation might group executable files. This would permit a subject-class–object-class profile that considered how privileged subjects as a group interact with executable files as a class.

Profiles can be created in a variety of ways, manually by a security officer, explicitly but automatically when a subject or object is introduced into the system, or automatically from a template when a subject first uses a given object. Since profiles have a per-subject, per-object, per-action granularity, the number of profiles to be maintained is proportional to the product of the numbers of subjects, objects and actions. This is mitigated by the fact that most subjects access only a subset of the system objects and not all actions are applicable to all subject–object pairs, but can still pose problems.

Denning suggests a number of profiles that should be useful for detecting intrusions. These include things such as login and session activity, with models for frequency of login, session duration, login location, etc., command or program execution behavior, with models for execution frequency for various commands, resource usage for commands, permission failures and resource exhaustion records, and file-access activity, with models for access frequencies, read and write behavior, access failures, and resource exhaustion.

Activity rules serve to both detect anomalous behavior and update profiles so that the system can track the evolution of user and system behavior. Rules are triggered by a number of actions. Audit-record rules are triggered by the creation of a new audit record that matches an activity profile. They may result in the update of the profile and or the issuance of an anomaly record. Periodic activity update rules are triggered by the expiration of a counter and result in the update of period-based activity profiles, e.g., logins per hour, and possibly issuance of an anomaly record. Anomaly record rules are triggered by the creation of an anomaly record and bring it to the immediate attention of the security officer. Periodic anomaly analysis rules generate summary reports of anomalous activity for the period covered.

Thus far, the paper seems to be setting the framework for an experimental evaluation of the proposed model, but it ends abruptly with conclusions that offer questions rather than answers.

There are several open questions:

- *Soundness of Approach:* Does the approach actually detect intrusions? Is it possible to distinguish anomalies related to intrusions from those related to other factors?
- *Completeness of Approach:* Does the approach detect most, if not all intrusions, or is a significant proportion of intrusions undetectable by this method?
- *Timeliness of Approach:* Can we detect most intrusions before significant damage is done?
- *Choice of Metrics, Statistical Models, and Profiles:* What metrics, models, and profiles provide the best discriminating power? Which are cost-effective? What are the relationships between certain types of anomalies and different methods of intrusion?
- *System Design:* How should a system based on the model be designed and implemented?
- *Feedback:* What effect should detection of an intrusion have on the target system? Should IDES automatically direct the system to take certain actions?
- *Social Implications:* How will an intrusion-detection system affect the user community it monitors? Will it deter intrusions? Will the users feel that their data is better protected? Will it be regarded as a step towards "big brother"? Will its capabilities be misused to that end?

For the most part, these questions remain unanswered today. In addition to these questions, the issue of false alarms should also be raised, though it may be being obliquely addressed under the soundness issue. As we will see below, the problem is a difficult one. The ad hoc nature of the IDES and other early approaches is indicated by the lack of any sort of underlying theory that might shed light on the first four of Denning's questions. Later research, like the early work, seems to be based almost entirely on intuition.

3.3 The late 1980s and early 1990s

In the late 1980s, a number of other notable systems were developed, mostly relying on a combination of statistical and expert systems approaches. In several cases, notably Haystack [71] and NADIR [38], the analysis engines incorporated commercial database management systems such as Oracle and Sybase, taking advantage of their abilities to organize the underlying audit data and to generate triggers when certain criteria were met by the data. NADIR remains in use at Los Alamos and is under active refinement to accommodate new threats and to adapt it to new target systems.

MIDAS was developed by the National Computer Security Center at NSA to monitor its Multics system,

Dockmaster [67]. It used a hybrid expert system and statistical analysis approach and examined audit log data from the Multics "Answering System", which controlled user logins augmented with data from other sources. MIDAS is notable in being one of the first systems deployed on a system connected to the internet and was in use from 1989 until the mid-1990s when Dockmaster was retired.

Wisdom and Sense, developed at Los Alamos and Oak Ridge, was another hybrid statistical and expert system [77]. It used nonparametric (distribution-insensitive) statistical techniques to derive its rule base from historical audit data. Like many other machine learning approaches, it suffered from a number of problems, including the difficulty of obtaining training data known to be intrusion free, high false alarm rates, and excessive memory requirements for manipulating the rule bases. The first two problems persist to the present time.

Up until this time, IDS systems used some form of audit data collected from the hosts being protected. The Network System Monitor, developed at the University of California at Davis changed this. NSM-monitored network traffic directly on an ethernet segment and used this as its primary source of data for analysis [36]. In the 2-month test reported in their paper, NSM monitored over 100,000 network connections and identified over 300 of them as being intrusive. Only a few of these connections were discovered by the administrators of the affected systems. Today, most commercial IDS systems use directly sensed network data as their primary (or only) data source.

4 Approaches to intrusion detection

In order to detect intrusions, some source of information in which the intrusion is manifest must be observed and some analysis that can reveal the intrusion must be performed. The first efforts at intrusion detection used operating-system auditing mechanisms to provide the data to be analyzed. This was soon augmented with additional auditing that was believed to be security relevant. In addition, applications can be enhanced to provide audit data that might contain evidence of attacks on the application. Systems that obtain the data to analyze from the operating system or applications subject to attack are called host based. The alternative to host-based sensing is to observe the traffic that goes to and from the system or systems being monitored and look for signs of intrusion in that data. In this section, we will first examine the issues associated with host/applications and network-based sensing. We will then examine the two primary analytical approaches, misuse detection and anomaly-based intrusion detection. This will be followed by brief (and necessarily incomplete) surveys of some current commercial and research systems.

4.1 Host/applications-based data collection

As we have seen, the earliest proposals for intrusion detection were based on the use of audit data from the host being monitored. Initially, the audit data used was that provided by the system to support its administration and to provide for proper user accounting. It was soon realized that additional information was needed. The Trusted Computer System Evaluation Criteria [27] (TCSEC or "Orange Book") set forth audit criteria, starting with class C2, listing the events to be audited. These included the use of identification and authentication mechanisms (logins etc.), introduction of objects into the program's address space (file opens and program executions), deletion of objects, administrative actions, and "other security relevant events." The requirements included protecting the audit trail from unauthorized access or tampering and specified in some detail the information that the audit records should contain. In addition, the audit mechanism was required to support selective auditing of individual users. Additional requirements, primarily having to do with classification labels (B1) and covert channels (B2), and cumulative audit thresholds (B3) appear at higher levels of the TCSEC, but any systems that claim to support C2-level security should have the basic auditing capabilities described above. These include Windows NT and Solaris.[9] It is also possible to supplement the general purpose logging mechanisms of many operating systems with specialized logging. CERT/CC provides guidelines for configuring some of these specialized logging mechanisms as a part of its "Security Improvement Module" series [13, Detecting Signs of Intrusion]. Most host-based systems collect data continuously as the system is operating, but periodic snapshots of the system state can also provide data that has the potential to reveal unexpected changes.

The early intrusion detection literature makes the assumption that some sort of audit data will be sufficient to detect intrusions, but provides no basis, other than intuition, to justify the assumption. Intrusions can manifest themselves in many ways. Even if the intrusive activity does not directly manifest itself in the audit trail,[10] it may be possible to infer something about the activity from the activity that is audited. For example, if a badly formed input to a process (such as the finger daemon) that monitors a network port causes the process to terminate abnormally, we would be much more likely to audit the abnormal termination rather than the message causing it. At the same time, not logging the message reduces

[9] The Solaris BSM audit mechanism provides the ability to collect detailed security relevant data at the system call level. Unfortunately, the data is difficult to use and poorly documented. IDS projects that have attempted to use this source have had to reverse engineer the data and write specialized tools for it [32, 52].

[10] During the Lincoln Lab IDS evaluation, it was noted that certain attacks do not manifest themselves in the Solaris BSM data [52, Sect. 8.2].

our ability to learn its source or to make a determination as to whether it was maliciously constructed or the result of an error in some other software component. On the other hand, unselective logging of messages may greatly increase the audit and analysis burdens, while selective logging requires both detailed knowledge of the correct message formats and possibly substantial computing on each input to determine if the input should be logged. The information necessary to determine this is often not available.

The choice of items to audit and the level of detail to record introduce a bias into the intrusion detection process. If the analysis technique involves looking for particular patterns in the audit trail as indications of intrusion, failing to audit the necessary material will make it impossible to detect certain intrusions. For systems that attempt to detect unusual usage patterns, the effects of this bias may be more subtle, but we know of no research that addresses the problem directly, although the recent work of Maxion and Tan [54] addresses the issue indirectly.

Wherever the information is obtained, there are choices to be made. Most modern operating systems provide for a substantial amount of auditing; however, the more detail collected, the more the collection activity is likely to impact system performance and the more storage space will be required to store the audit data. If the data is analyzed on the collecting host (or on a host that is part of the monitored environment and used for other processing) in real time, the analytical effort may create a substantial performance impact on top of the sensing impact. Offline analysis will also be affected by the amount of data collected. At the same time, collecting too little data raises the risk that attack manifestations will be missed. In the absence of a theoretical basis for characterizing intrusive activity, only ad hoc techniques, guided by experience, are available.

4.2 Network-based data collection

The alternative to host-based sensing is to observe the traffic that goes to and from the system or systems being monitored and look for signs of intrusion in that data. This approach has the advantage that a single sensor, properly placed, can monitor a number of hosts and can look for attacks that target multiple hosts. It has the disadvantage that it cannot see attacks such as those made from a system console, those made from a dial-up modem connected directly to the host, or those arriving over a path that does not traverse the network segment being monitored. Encrypted connections are also difficult to monitor.

Nonetheless, network monitoring is the method of choice for many commercial intrusion detection systems. There are a number of reasons, including the ease of constructing a dedicated platform that combines network sensing with data reduction and analysis. We examine one such platform, Snort, in Sect. 4.7. Constructing a network-based data-collection system is fairly straightforward. All that is required is a network interface device that can be placed in promiscuous mode, so that it will see all the traffic on the network segment to which it is connected, and some mechanism for recording the traffic or portions of it. The `libpcap` [40] library is often used for this purpose. For modest data rates, commodity platforms, such as PCs, suffice as network sensing devices. Unfortunately, network data rates are rising rapidly, with most network segments being upgraded from 10 megabits/s to 100 megabits/s, and gigabit segments are becoming common. While the speed of the segment does not necessarily indicate the amount of traffic carried on the segment, traffic rates are increasing to consume the available bandwidth. Figure 3 illustrates the nature of the problem. It shows the number of memory cycles available to process each packet for a variety of packet lengths (in bytes) and data rates (in megabits/s) for two modes of processing, the first in which only the 68 bytes of header information is input and processed and one in which the entire packet is transferred and processed. These figures assume a 100 MHz processor bus that transfers 32 bits per memory cycle. The processing time available is calculated by subtracting the time required to transfer the packet or packet header into memory from the packet duration and converting to memory cycles at 10 ns per cycle. For the kinds of processing algorithms used in IDSs, memory cycles are more likely than CPU cycles to be the limiting factor. Memory speeds seem to increase at a much slower rate than CPU clock speeds. For reference purposes, 5 megabits/s is near the useful capacity of a 10baseT ethernet while 50 and 500 megabits/s probably represent useful capacities for 100baseT and gigabit network segments.

The shortest packet length, 68 bytes, essentially represents header information only. If only the header is transferred into processor memory, 17 cycles will be required, leaving the remaining cycles available to process the header. This means that some processing can be accomplished for the headers of longer packets, even at network speeds of up to a gigabit. On the other hand, no realistic processing of the packet contents is likely to be possible for data rates above 50 megabits/s, since even a simple pattern match against a single pattern is likely to require more memory cycles per byte than are available.

Network sensors are also subject to attack. It is typical to combine sensing and analysis on a single platform. If the platform performs a stateful analysis that requires it to retain state information associated with many sessions, it can be subject to a resource exhaustion attack. Such platforms may not develop the same "world view" as the systems they are guarding. The TCP/IP protocol suite has a number of ambiguities that are resolved differently by different operating systems. By using overlapping, retransmitted, packet fragments, the intruder may

Data rate (Mb/s)	Packet length (bytes)	Cycles per packet	Processing cycles	
			per packet	per byte
1	68	54 400	54 383	800
	400	320 000	320 000	800
	1600	1 280 000	1 280 000	800
5	68	10 880	10 863	160
	400	64 000	63 900	160
	1600	256 000	255 600	160
10	68	5440	5423	80
	400	32 000	31 900	80
	1600	128 000	127 600	80
50	68	1088	1071	16
	400	6400	6300	16
	1600	25 600	25 200	16
100	68	544	527	8
	400	3200	3100	8
	1600	12 800	12 400	8
500	68	109	92	2
	400	640	540	2
	1600	2560	2160	2
1000	68	54	37	1
	400	320	220	1
	1600	1280	880	1

Fig. 3. Memory cycles per packet processing time (100 MHz, 32-bit bus)

be able to present different results to the IDS and the target system. If an attacker becomes aware of the presence of a network IDS, he may attempt to disable the IDS before attacking the target network. These and other issues associated with network sensing are discussed in detail by Ptacek and Newsham [62].

4.3 Intrusion detection and the signal detection problem

Intrusion detection can be viewed as an instance of the general signal detection problem [9, 31]. In this case, intrusion manifestations are viewed as the signal to be detected while manifestations of "normal" operations are considered to be noise. In classical signal detection approaches, both the signal and the noise distributions are known, and a decision process must determine if a given observation belongs to the signal-plus-noise distribution or to the noise distribution. Classical signal detectors use knowledge of both distributions in making a decision, but intrusion detectors typically base their decisions either on signal (signature-based detectors) or noise (anomaly-based detectors) characterizations. Each approach has strengths and weaknesses. Both suffer from the difficulty of characterizing the distributions. A recent technical report by Stefan Axelsson [9] develops a useful taxonomy of intrusion detection techniques. In our discussions of the primary approaches, we will generally follow the Axelsson taxonomy.

4.4 Anomaly-based intrusion detection – is strange bad?

Anomaly-based intrusion detectors equate "unusual" or "abnormal" with intrusions. Given a complete characterization of the noise distribution, an anomaly-based detector recognizes as an intrusion any observation that does not appear to be noise alone. Characterizing the noise distribution so as to support detection is non trivial. Characterization approaches have ranged from statistical models of component/system behavior to neural networks and other AI techniques to approaches inspired by the human immune system. The primary strength of anomaly detection is its ability to recognize novel attacks. Its drawbacks include the necessity of training the system on noise, with the attendant difficulties of tracking natural changes in the noise distribution. Changes may cause false alarms while intrusive activities that appear to be normal may cause missed detections. It is difficult for anomaly-based systems to classify or name attacks.

The portion of the Axelsson taxonomy that deals with anomaly-based detection is summarized in Fig. 4. In general, the anomaly systems are based on the assumption that intrusive activities are necessarily different from non-intrusive activities at some level of observation. Early work such as that represented in the IDES [26] and others was based on the assumption that statistical models of user behavior were adequate for the purpose. While these systems were apparently able to detect some

Style	Model	Method	Examples
Self-learning	Non time series	Rule-modeling	W&S [77]
		Descriptive statistics	IDES [26] NIDES [41] EMERALD [61] JiNao [42] Haystack [71]
	Time series	Neural net	Hyperview(1) [21]
		Immune inspired	Forrest, et al. [82]
Programmed	Descriptive statistics	Simple statistics	MIDAS(1) [67] NADIR(1) [38] Haystack(1) [71]
		Simple rule based	NSM [36, 57]
		Threshold	ComputerWatch [28] Tripwire [45]
	Default deny	State series modeling	DEPM [46, 47] JANUS [33] Bro [60]

Fig. 4. Anomaly-based intrusion detectors (after [8]). The number in parenthesis indicates the level for a multilevel detection scheme

intrusions, they often raised an excessive number of false alarms. Part of the problem is that neither the noise characteristics (normal usage) nor the signal characteristics (intrusions) have been adequately studied. Maxion and Tan [54] have taken steps towards matching data characteristics to anomaly detector characteristics in ways that are useful in predicting detector behavior. Although most of their results were obtained from artificial data, they performed one experiment on natural data obtained from Solaris BSM audit logs. They noted that the differences in the regularity of the data from user to user in their data set would make it difficult to use the same detector to detect anomalous behaviors across the entire user set. The signal/detector mismatch could translate into a substantial false alarm rate. This is consistent with other experiences with anomaly detection approaches. More work of this kind is needed to determine the effective limits of the anomaly detection approach in practice.

In addition to assuming that anomalies equate to intrusions, the anomaly detection approach to intrusion detection makes another, more subtle assumption. It assumes that intrusions will be accompanied by manifestations that are sufficiently unusual so as to permit detection. It is unclear that this is necessarily the case. As was pointed out by both Anderson [5] and Myers [58], we need to be concerned with the highly skilled intruder who takes steps to understand the system under attack and to avoid detection. We suspect that little appears in the literature concerning the practices of such intruders and that the data used to test or evaluate IDSs does not reflect their approaches.

4.5 Misuse detection – patterns of misbehavior

In order for a signature-based IDS to detect attacks, it must possess an attack description that can be matched to sensed attack manifestations. This can be as simple as a specific pattern that matches a portion of a network packet or as complex as a state machine or neural network description that maps multiple sensor outputs to abstract attack representations. If an appropriate abstraction can be found, signature-based systems can identify previously unseen attacks that are abstractly equivalent to known patterns. They are inherently unable to detect truly novel attacks and suffer from false alarms when signatures match both intrusive and non-intrusive sensor outputs. Signatures can be developed in a variety of ways, from hand translation of attack manifestations to automatic training or learning using labeled sensor data. Because a given signature is associated with a known attack abstraction, it is relatively easy for a signature-based detector to assign names (e.g., Smurf, Ping-of-Death) to attacks.

The portion of the Axelsson taxonomy that deals with signature-based systems is shown in Fig. 5. These systems range from very simple to quite elaborate. At the simple end of the spectrum are systems that do simple string matching or that follow simple rules. Many of these systems are able to base decisions on single network packets. For a large class of attacks, particularly those targeting specific vulnerabilities such as buffer overflow possibilities in programs that read inputs from the network, this is sufficient. Similarly, attacks that involve packet pathologies such as identical source and destination addresses

Style	Model	Method	Examples
Programmed	State modeling	State transition	USTAT [39]
		Petri net	IDIOT [20]
	Expert system		NIDES [41]
			EMERALD [61]
			MIDAS-direct [67]
			DIDS [72]
			MIDAS(2) [67]
	String matching		NSM [36, 57]
	Simple rule-based		NADIR [38]
			NADIR(2) [38]
			ASAX [34]
			Bro [60]
			JiNao [42]
			Haystack(2) [71]

Fig. 5. Signature-based intrusion detectors (after [8]). The number in parenthesis indicates the level for a multilevel detection scheme

can be detected on a single packet basis. Exploits that require multiple steps to be carried out require the IDS to consider multiple data items, whether a series of network packets or a set of audit records. All of the approaches represented in Fig. 5 encode some representation of known attacks into a form that can be compared against the sensed data. An attack is recognized when the sensed data matches a pattern. With relatively few exceptions, signature-based systems operate with fairly concrete data representations. Signatures that are too specific will miss minor variations on a known attack, while those that are too general can generate large numbers of false alarms. Some of the more successful systems, such at the STAT family [80], use a sufficiently abstract representation for some classes of attacks to be able to recognize attacks that are "new" in the sense that they were unknown at the time the signature was created. This behavior is the exception and, in general, signature-based IDSs, like virus detectors, have to be updated when new attacks are discovered. Most commercial systems fall into this category and the ease and frequency of updates is an issue in choosing the system to deploy.

4.5.1 Signatures are hard – an example

To illustrate some of the problems associated with defining and using signatures, we consider an example, based loosely on the Morris Worm's `fingerd` attack [68, 73]. This was one of the first buffer overflow attacks. The finger service listens for requests on port 79 and, when a connection is established, reads a single line, terminated with a CRLF pair. Neither the current standard (RFC-1288 [83]) nor the original (RFC-742 [35]) specify the length of the line. In the original implementation, characters were read with a `gets` input routine which continued to transfer characters until a `null` character was encountered. In describing the attack, Seeley [68] notes:

> Probably the neatest hack in the worm is its co-opting of the TCP finger service to gain entry to a system. Finger reports information about a user on a host, usually including things like the user's full name, where their office is, the number of their phone extension and so on. The Berkeley version of the finger server is a really trivial program: it reads a request from the originating host, then runs the local finger program with the request as an argument and ships the output back. Unfortunately the finger server reads the remote request with gets(), a standard C library routine that dates from the dawn of time and which does not check for overflow of the server's 512-byte request buffer on the stack. The worm supplies the finger server with a request that is 536 bytes long; the bulk of the request is some VAX machine code that asks the system to execute the command interpreter sh, and the extra 24 bytes represent just enough data to write over the server's stack frame for the main routine. When the main routine of the server exits, the calling function's program counter is supposed to be restored from the stack, but the worm wrote over this program counter with one that points to the VAX code in the request buffer. The program jumps to the worm's code and runs the command interpreter, which the worm uses to enter its bootstrap.

The most straightforward signature for detecting this attack would be one that alerts on all messages with a content length greater than 512 that are destined for the finger port, port 79. Now, what if observations of network traffic indicated that due to malfunctioning software elsewhere in the network, malformed packets with lengths greater than 512 destined for port 79 were common? These packets would probably effect a denial of service attack on the finger service if they overwrote the return address with a "random" value, but the precise effect might be unpredictable. If we assume that we only want to deal with serious intrusions rather than accidental denial of service, a more specific signature is needed.

Looking for the hand-crafted code that executes the shell seems to be an obvious approach. We could look for this pattern in the body of the message so that the combination of excessive length and identifiable code would constitute the signature. Since the message will modify the program stack to replace the return address of the affected subroutine with an address that lies within the buffer, i.e., the address of the sequence of instructions that executes the shell, another possibility would be to look for the address of this code in the portion of the message that overlays the return address. This will require some computation, since variations on the attack could place code anywhere in the buffer, and any number that falls between the origin of the buffer and the end of the message is suspect. Since most IDS signature description languages do not have this power, the code itself is probably the best bet.

A clever attacker could attempt to hide the code that calls the shell by adding code to modify the region in which it is stored prior to executing it. A simple loop to "XOR" the code region with a constant would suffice. Since this kind of modification can be performed in arbitrary ways, there is no pattern that can easily detect variations on the original attack. Given that distinguishing malicious messages from garbage is hard, how easy is it to observe the effects of the attack? When the attack code is executed, a command interpreter or shell (`/bin/sh`) inherits port 79 and awaits further input. The shell typically issues a prompt to indicate that it is awaiting input, and since it is operating with root or superuser privileges, by convention, its prompt consists of a "#". By looking for this output from the supplanted `fingerd`, we might be able to detect the fact that `fingerd` has been replaced by a root shell. However, on most systems, the attacker can specify the prompt to be used as an argument to the shell when it is started. The command can be set to some arbitrary value, even perhaps to a response that `fingerd` might emit such as "User list unavailable".

If host-based sensing rather than network-based sensing is used, a different picture is presented. Packets that crash `fingerd` will probably create log entries indicating the abnormal termination of the process, with reasons for the termination that might include illegal instruction execution, segmentation faults, or other illegal address failures. A successful buffer overflow attack results in an exec call which should also be logged. From the description of `fingerd` given above, it is clear that finger uses the exec system call to execute the local `finger` program. If this is the only use of exec, its presence in the log with any other argument is a clear intrusion signature. However, the intruder has the ability to modify log files at this point, and unless the logs are adequately protected from modification, the intruder may be able to erase this signature before it is detected.

This scenario presents a sort of "cat and mouse" game between the intruder and the IDS. While the bulk of present-day intrusions appear to come from scripts [7] that repeat exactly the same set of actions on many machines, a more sophisticated intruder can easily modify the attack to avoid simple signature-based detection as seen above.

4.6 Commercial systems

Beginning in the late 1980s, IDSs moved from the laboratory to the commercial arena. Today, there is a thriving commercial market which is in a constant state of flux. Any attempt at a comprehensive survey is almost certain to be out of date as soon as it is finished. In addition, objective data concerning the performance of commercial systems is often difficult to find, and the information available from vendors is usually more marketing oriented that technical. In many cases, the algorithms, data files, etc., used in commercial systems are proprietary. Over the next few years, we can anticipate further mergers of IDS companies accompanied by rapid evolution of the systems that they field. More and more, IDS is being viewed as a component in a more comprehensive security solution, and companies that build multiple security components such as vulnerability scanners and firewalls are starting to integrate these components into comprehensive system management systems.

There is also a movement towards interoperability of components from multiple vendors. The IETF Intrusion Detection Working Group (IDWG) is in the process of defining a protocol for exchanging alert information among IDS systems. Major vendors in the system management area are now supporting IDS systems as part of their management suites.

In the fall of 1999, CERT/CC published a comprehensive survey [2] of the state of practice in intrusion detection. While the information on particular systems is somewhat out of date, the general commentary concerning commercial systems in general is still valid. In general, the systems examined were somewhat difficult to install and configure. For the most part, the signatures that they used to define intrusions were either not available for the user to inspect and modify or were difficult to understand. Updates of signatures were infrequent, in some cases requiring a new release of the IDS, making

it difficult to react to new attacks quickly. False alarms are a significant problem in many commercial systems. In a recent presentation at RAID 2000, Julisch reports [43] false alarm rates of tens of thousands per month for one commercial system (NetRanger) deployed on an operational network.

In the following paragraphs, several systems are described briefly to give the reader a feeling for the variety of systems currently available. The choices are largely arbitrary.

4.6.1 ISS

RealSecure®from Internet Security Systems[11] is a real-time IDS. It uses a three-part architecture consisting of a network-based recognition engine, a host-based recognition engine and an administrator's module. The network recognition engine runs on dedicated workstations to provide network intrusion detection and response. Each network recognition engine monitors a network segment looking for packets that match attack signatures. When a network recognition engine detects intrusive activity, it can respond by terminating the connection, sending alerts, recording the session, reconfiguring firewalls, etc. It also passes an alarm to the administrator's module or a third-party management console.

The host-based engines analyze log data to recognize attacks. Each host engine examines its system's logs for evidence of intrusions and security breaches. Log data may contain information that is difficult or impossible to infer from network packet data. The host engine may prevent further incursions by terminating user processes or suspending user accounts. It may also take actions similar to those performed by a network engine.

Multiple recognition engines are managed by an administrative module. The result is comprehensive protection, easily configured and administered from a single location. The administrative module is supplied with both recognition engines and is also available as a plug-in module for a variety of network and systems management environments.

ISS also makes a vulnerability scanner which it is in the process of integrating with its intrusion detection system. This will allow the IDS to be configured in such a way that alerts for attacks against vulnerabilities known not to exist in the protected network can be suppressed.

4.6.2 Cisco Systems

In the past several years, Cisco's IDS products have evolved from a stand-alone hardware/software platform known as NetRanger to an enterprise-wide system consisting of sensors and management facilities. Audit trails from Cisco's routing products can also be fed into the management and analysis facilities. In addition to raising alarms or alerts, the Cisco IDS products can also be used to reconfigure routing tables to block attacking sources as seen in the following quotation from the current online product data [18]:

- Reset TCP connections after an attack begins. The Sensor can reset individual TCP connections after an attack to eliminate the threat. Communication between all other connections continues, thereby decreasing the likelihood of denial of service problems that can occur after shunning TCP-based attacks.
- Shun the attack. Shunning is a term that refers to the Sensor's ability to use a network device to deny entry to a specific network host or an entire network. To implement shunning, the Sensor dynamically reconfigures and reloads a network device's access control lists. This type of automated response by the Sensor should only be configured for attack signatures with a low probability of false positive detection, such as an unambiguous SATAN attack. In case of any suspicious activity that does not trigger automatic shunning, you can use a Director menu function to shun manually. **Caution** Shunning requires careful review before it is deployed, whether as a set of automatic rules or operational guidelines for staff. In case an alarm is generated from a critical host or network, NetRanger can be configured to never shun that host or network. This safety mechanism prevents denial of service attacks using the NetRanger infrastructure.

The caution is indicative of an unsolved problem in intrusion response. If the system reacts automatically by refusing connections from the network or host believed to be responsible for the intrusion, attackers have the opportunity to create denial of service attacks by spoofing an attack from a critical source. Accidental false alarms can have a similar effect.

4.6.3 Tripwire

Tripwire®is a file integrity assessment tool[12] that is useful for detecting the effects of an intrusion. Tripwire creates a database of critical system file information which includes file lengths and cryptographic checksums based on each file's contents. Tripwire compares current information with a previously generated baseline and identifies changed files. Tripwire will report modified files, but the user must decide whether the modifications resulted from an intrusion. Since most monitored files are not

[11] `http://www.iss.net.`

[12] `http://www.tripwire.com`. Both commercial and public-domain versions are available.

expected to change except when new software versions are installed, changes usually indicate an unexpected or unauthorized activity.

For reliable Tripwire results, the database and program must be protected from tampering. This can be accomplished by maintaining them offline or online using read-only storage media. Configuring Tripwire can be problematic, especially for large, multi-use systems since it is not easy to determine which files associated with some services and applications are expected to change and which are not.

4.7 A public-domain system – Snort

Snort [66] is a recent open-source, public-domain effort to build a lightweight, efficient, ID tool that can be deployed on a wide variety of Unix platforms. According to the Snort web site[13]: "Snort is a libpcap-based packet sniffer/logger which can be used as a lightweight network IDS. It features rules-based logging and can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more. Snort has a real-time alerting capability, with alerts being sent to syslog, a separate 'alert' file, or as a WinPopup message via Samba's smbclient".

Snort is currently undergoing rapid development. The user community is contributing auxiliary tools for analyzing and summarizing snort logs, providing additional capabilities. More importantly, there is a large group of users who contribute new signatures. As a result, new attacks are quickly represented in the signature database. The Snort community has embraced the IDWG effort to standardize alert messages so that Snort sensors will be able to interoperate easily with other IDS systems and system management consoles that support the standard. In its present form, Snort is stateless and can only examine single packets; however that could change. Port scan and packet reassembly capabilities are planned for the next major release.

4.8 Research systems of the 1990s

4.8.1 NIDES and EMERALD

SRI's pioneering intrusion detection work began in 1983 when a multivariate statistical algorithm was developed to characterize user behaviors [61]. EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances)[14] builds on this earlier work at SRI and uses both deviations from normal user behavior (anomalies) and known intrusion patterns (signatures).

A major goal of EMERALD is to provide intrusion detection for large, loosely coupled enterprise networks. Such environments are more difficult to monitor and analyze due to the distributed nature of the incoming information. EMERALD structures users into a federation of independently administered domains, each with its own network services and trust relationships with other domains. In this context, a centralized repository or analysis facility is likely to result in significant performance degradation. EMERALD uses a hierarchical "divide and conquer" approach to address these issues.

The hierarchical approach provides three levels of analysis performed by a three-tiered system of monitors: service monitors, domain monitors and enterprise monitors. These monitors have the same basic architecture: a set of profiler engines (for anomaly detection), signature engines (for signature analysis) and a resolver component for integrating the results generated from the engines.

EMERALD is an example of the direction that future IDSs may take. As intruders become more sophisticated in their attacks, they will be increasingly likely to disperse the evidence of their work across networks, making it difficult to sense when a distributed/coordinated attack is occurring. In such situations, the ability to collect, assimilate, correlate, and analyze information emanating from diverse sources in real-time becomes essential.

4.8.2 The STAT family

The State Transition Analysis Technique [80] (STAT)[15] developed at UCSB is a method for representing the sequence of actions that an attacker performs to achieve a security violation. The technique provides a general framework in which host-based (USTAT) [39], network-based (NETSTAT) [78, 79], and distributed multi-host (NSTAT) tools have been built.

State transition analysis uses a graphical notation to represent a penetration, precisely identifying its requirements and the nature of the compromise. Analysis tools use information contained in a system's audit trail or network traffic to compare the state changes represented in the data to the state transition diagrams of known penetrations. State transition analysis assumes that (a) penetrations require the attacker to possess some minimum prerequisite access to the target system (the initial state), and (b) all penetrations lead to the acquisition of some previously unheld ability (the compromised state).

After the initial and compromised states of a penetration scenario have been identified, signature actions are identified. Recent work resulted in the development of STATL [30], a language for specifying intrusive actions. Tools are under development to generate detectors for members of the STAT family from STATL descriptions.

[13] http://www.snort.org.

[14] http://www.sdl.sri.com/emerald/.

[15] http://www.cs.ucsb.edu/~kemm/netstat.html/.

4.8.3 Integrity checkers

Tripwire has been discussed in Sect. 4.6.3. The DERBI (Diagnosis, Explanation and Recovery from computer Break-Ins) system was built at the AI lab at SRI as an ex post facto mechanism for analyzing the residue left by successful break-ins. DERBI can be viewed as giving a semantic interpretation to raw data of the kind produced by Tripwire. As far as we can tell, the DERBI project resulted in no accessible publications, but the materials used in several briefings given on the project are available at the DERBI web site[16]. The system is described briefly in the abstract for a talk given by the PI, Mabry Tyson, in May of 1999 [76]:

> DERBI has three major components. The "head" is the intelligent part that directs the search and processes the evidence. The head is implemented as a PRS graph. The "feet" are the data collectors, or sensors, that scour the file system looking for evidence. The "body" interfaces between the high-level requests of the head and the low-level details of the feet.
> DERBI searches for the fragmentary tracks of the cracker and tries to match these with a model of expected behaviors of crackers. Crackers may camouflage their tracks but in doing so they often leave other, less-obvious traces. DERBI looks for secondary effects of the crackers presence on the system. The evidence found is accumulated and entered into a generic model of an attack. As pieces fall into place, probabilities are calculated to give a system administrator a better idea as to whether an attack has occurred. DERBI's design allows for searching harder for evidence for missing pieces of an attack.
> After accumulating evidence, DERBI uses the attack model to explain to the system administrators what it has found and why this evidence leads to conclusions about the integrity of the system. As it explains what has happened (or what it cannot know), it can also recommend courses of action to be taken to recover from the intrusion.
> Real-time monitoring systems should be better than DERBI at detecting and stopping attacks earlier. However, at a recent DARPA-sponsored evaluation[17], DERBI performed as well as the other systems in detecting attacks.

4.8.4 Data-mining approaches and learning approaches

Data-mining approaches are based on the automatic extraction of features from a large set of data. They develop rules that can describe various relationships among the data items. Lee has applied these techniques [49] to network and host audit data to develop models that support intrusion detection. He reports that several types of algorithms are particularly useful for mining audit data.

Classification: maps a data item into one of several predefined categories. These algorithms normally output "classifiers", for example, in the form of decision trees or rules. An ideal application in intrusion detection will be to gather sufficient "normal" and "abnormal" audit data for a user or a program and then apply a classification algorithm to learn a classifier that can label or predict new unseen audit data as belonging to the normal class or the abnormal class;

Link analysis: determines relations between fields in the database records. Correlations of system features in audit data, for example, the correlation between command and argument in the shell command history data of a user, can serve as the basis for constructing normal usage profiles. A programmer, for example, may have "emacs" highly associated with "C" files;

Sequence analysis: models sequential patterns. These algorithms can discover what time-based sequences of audit events are frequently occurring together. These frequent event patterns provide guidelines for incorporating temporal statistical measures into intrusion detection models. For example, patterns from audit data containing network-based denial-of-service (DOS) attacks suggest that several per-host and per-service measures should be included.

Lee's approach performed very well for detecting known attacks during the 1998 DARPA evaluation, but, like most other approaches, did fairly poorly on several categories of new attacks. The features that the data-mining approach associated with certain attack categories are also somewhat suspect, since they are most likely to be associated with fairly naive attack patterns. For example, obtaining a root shell[18] without a root login or `su` to root seems to be the defining pattern for user to root attacks [49, Sect. 5.2.4]. While this is a sufficient condition for such an attack, it is far from necessary. Attacks are known that can execute specific commands as root,

[16] `http://www.ai.sri.com/~derbi`.

[17] Slides showing the results of the 1998 and 1999 evaluations are available at `http://www.ai.sri.com/~derbi` as presentations at the PI meetings in December of those years.

[18] These techniques assume that obtaining a root shell is indicated by a change in the shell prompt to a string ending in "#".

obviating the need for a root shell. Similarly, it is simple for an ordinary user[19] to spoof a legitimate `su` command, and then perform an exploit that results in an apparently legitimate root shell. While the approach looks promising, it remains to be seen as to whether it can be used in practice. Like all machine learning approaches of which we are aware, it requires a substantial amount of data for which the "ground truth" is known or can be determined. The relatively poor results obtained by Lee on data that was very different from the labeled training data seems to indicate that the approach will not adapt to new environments and attacks without a substantial amount of "hand holding". If mining done in one environment does not transfer to another, the laboratory results may not be of practical use.

5 Evaluating IDSs

There is relatively little work prior to 1998 in the field of evaluating intrusion systems. The work of Puketza and others at the University of California at Davis [63, 64] is the only reported work that clearly predates the Lincoln Lab effort that began in 1998. These papers describe a methodology and software platform for the purpose of testing IDSs. The methodology consists of using scripts to generate both background traffic and intrusions with provisions for multiple, interleaved streams of activity. These provide a (more or less) repeatable environment in which real-time tests of an IDS can be performed. Only a single IDS, the Network Security Monitor [36], seems to have been tested, and the tests reported could not be seen as any sort of a systematic evaluation. The earlier work [63], dating from 1993, reports the ability of NSM to detect several simple intrusions, both in isolation and in the presence of stresses. One form of stress is induced by system loading. Load is measured in terms of the number of concurrent jobs running on the host supporting NSM, and NSM is reported to drop packets under high load averages (42% byte stream loss at a load average of about 14.5). Other forms of stress include background noise (non-intrusive network activity), session volume (the number of commands issued during an intrusive session) and intensity (number of concurrent sessions on the link being monitored). No experimental results are given for these forms of stress. In their later paper [64], the Davis group concentrates on the ability of the test facility to support factoring of sequential attacks into a number of concurrent or overlapping sessions. They report that NSM assigns lower scores to some attacks that have been factored, noting that NMS's independent evaluation of individual network connections may allow attacks to be hidden in this way.

[19] The user in question may be either a misbehaving legitimate user of the system or an intruder who has penetrated the system to the point of assuming an unprivileged user role.

The most comprehensive evaluations of IDSs reported to date were the 1998 [50] and 1999 [51] offline evaluations performed by the Lincoln Laboratory at MIT. The systems evaluated are the results of research funded by DARPA. In these evaluations, investigators were given sensor data in the form of sniffed network traffic, Solaris BSM audit data, Windows NT audit data (added in 1999) and file-system snapshots and asked to identify the intrusions that had been carried out against a test network during the data-collection period. The test network consisted of a mix of real and simulated machines, and the background traffic (noise) was artificially generated by the real and simulated machines, while the attacks were carried out against the real machines. Training data was supplied that contained a variety of attacks that were identified in the corresponding documentation. The data used for evaluation contained a mix of attacks that had been present in the training data and previously unseen attacks.

An analysis of this evaluation shows a number of serious flaws [55], including failures to appropriately validate the background data used (especially with respect to its ability to cause false alarms), the lack of an appropriate unit of analysis for reporting false alarms and the use of questionable or inappropriate data analysis and presentation techniques. The data rate represented by the test data is not given, but calculations based on the data-set sizes indicate an average rate of a few 10s of kilobits per second.

A total of 32 attack types were present in 1998. These were organized, according to an attacker centric taxonomy, into four categories – denial of service, remote to local, user to root, and probing/surveillance – without regard to manifestation. The best system was only able to detect about 75% of the 120 attacks present in the evaluation data. The percentage of the attack types detected is not reported in the 1998 results [50]. False alarms are reported "per day" rather than as a percentage. The best system generated two false alarms per day, while most systems produced some tens of false alarms per day. These figures are problematic for several reasons. The noise component of the data is responsible for the false alarms, but no comparison was made between real and artificial data with respect to false alarm characteristics and any given "natural" environment might produce more (or fewer) false alarms at the data rate used. If the false-alarm characteristics of the data are proportional to the data rate, deploying the evaluated systems on networks carrying a few megabits per second of traffic could result in a hundredfold increase in the number of false alarms reported per day. The 1999 evaluation produced similar, but slightly better, results for detection and false-alarm performance, but over a substantially broader base of attacks. The real improvement is one in breadth of coverage rather than in effectiveness.

Despite its shortcomings, the Lincoln evaluation indicates that even the best of the research IDSs falls far

short of the DARPA goals for detection and false-alarm performance. The Air Force Rome Lab has built a real-time test bed [29] based on the system used by Lincoln to generate its offline evaluation data. This system has been used to evaluate a few of the research systems, with results similar to those obtained in the offline evaluation.

In 1998, while the Lincoln group was developing and carrying out its test methodology, a group at the IBM Research Division in Zurich issued a technical report [22] describing another experimental facility for comparing IDSs. As the previous work, the Zurich group reports on the design and implementation of a real-time test bed. The Zurich test bed consists of several client machines and several server machines under the control of a workstation used as the workbench controller. The report discusses a number of issues associated with the generation of suitable background traffic, noting the difficulties associated with alternatives, including developing accurate models of user and server behavior, using test suites designed by operating system developers to exercise server behavior and using recorded "live" data. The authors tend to favor the test-suite approach, but recognize that it may bias results with respect to false alarms. Attacks are obtained from an internally maintained vulnerability database which makes hundreds of attack scripts available although only a few are applicable to the initial workbench configuration which only supports FTP services. The paper describes several of the attacks on FTP. Considerable attention is given to the controller component of the workbench, which allows the systems under evaluation to be configured and administered from a single console. The controller also allows the results from several IDSs to be compared. Unfortunately, the report does not present any results obtained from the workbench. Several observations from the paper are worth noting. The first is that "generating realistic, normal behavior is a time-consuming task when working in a heterogeneous environment". The second is that "it is worth noting that the definition of a set of criteria to evaluate the effectiveness of existing IDSs remains an open issue which we will have to address if we want to perform pertinent comparative studies".

Both anecdotal evidence and the results from the few evaluations that have been performed indicate that current IDSs are not as effective as could be desired. No comprehensive evaluation of commercial products has been performed so that it is difficult to obtain comparative figures. In 1999 IBM Zurich tested two commercial systems [23], RealSecure 3.0.x and NetRanger 2.1.2, using exploit scripts and tools available in their vulnerability database that were compatible with their test environment and for which the IDS claimed coverage in its documentation. RealSecure detected 30 of 42 attacks, while NetRanger detected 18 of 32. Deployed in an operational setting, RealSecure issued some 8000 alarms in a month, over half of which were due to a weekly scan of the network performed for maintenance purposes. Comparable figures are not given for NetRanger, apparently because of performance problems. Both systems had fairly high false alarm rates, but issued false alarms for different classes of activity.

Recent work at Zurich [1] addresses the potential of IDS systems to detect certain classes of intrusions using an analysis of design principles rather than an evaluation based on an actual implementation. The paper develops a technique for describing activities which may be either intrusive or benign and describes the features that an IDS must have in order to successfully detect the intrusive activities while rejecting benign ones. Although this work is in its early stages, the author claims that it is generic and easily extends to a wide variety of intrusive activities, including those for which signatures have not yet been developed.

Reviews and comparisons of commercial IDSs appear from time to time, usually at the web sites of online publications.[20] The reviews are generally superficial and lack details concerning the test methods used. The rapid rate at which new products are introduced and existing products modified gives these reviews a limited window of utility. This is discussed in a recent SEI technical report [2].

All of the evaluations performed to date indicate that IDSs are only moderately successful at identifying known intrusions and quite a bit worse at identifying those that have not been seen before. This renders automatic response to intrusions, a goal of both the research and commercial communities, a dubious prospect. Blocking an attack by dynamically reconfiguring a firewall to block the intruding source carries with it the risk of a self-imposed denial of service attack if it is done in response to an event wrongly identified as intrusive. Anecdotal evidence suggests that legitimate network diagnostic activities have resulted in the temporary blockage of network traffic on at least one US military application system.

6 The future of intrusion detection

Over the past several years, DARPA has sponsored a substantial number of intrusion detection projects. Their goal has been stated in a number of ways, but in general, it is to develop systems (or aggregates of systems) that can detect more than 99% of the attacks (or cover more than 99% of the potential attack space) with a false alarm rate of less than 1% (sometimes stated as less than 0.1%). These systems are intended to be effective against both known and previously seen attacks as well as against new or unknown attacks. The results of the most recent DARPA-sponsored evaluation fall far short of this goal,

[20] Unfortunately, these reports tend to be transient in nature. One such source cited by a reviewer of another paper by the author was invalid by the time the review reached the author. Only two of the three sites mentioned in the SEI report were still valid as of November 2000.

as noted above. Although commercial systems have not been evaluated using the criteria and test data developed by Lincoln Lab for DARPA, anecdotal data indicates that commercial systems are even further from the goal, especially with respect to false-alarm performance.

While the reasons for this situation are not entirely clear, it is our opinion that they reflect a fundamental problem. Early work in the intrusion detection field was based on two assumptions. One was that the manifestations of certain kinds of intrusions would be so obvious that rules for detecting them could be written and detectors could easily find them. The other was that deviations from "normal" behaviors by users or programs would usually be definite indications of malicious or intrusive activity. While the intuitions behind these assumptions are sound, they have led to increasingly ad hoc approaches to intrusion detection. There is no underlying theory that relates detection approaches to detections or that allows useful predictions to be made concerning the relative powers of various techniques. Even against the relatively naive attacks that dominate current internet incident reports, most of the systems fare poorly. The Lincoln Lab evaluation showed that no system exhibited acceptable performance against new attacks. We suspect that none of the current research approaches will show themselves capable of reaching DARPA's goals and that incremental improvements will be limited.

There are two areas in which substantial work is necessary. One is the characterization of "normal" behavior. It is generally agreed that there is no such thing as a "typical" system; however, it may be possible to develop methods for characterizing a given environment sufficiently well so that optimal anomaly detectors for that environment can be defined. Such work may also allow the limits of a detector, in terms of expected false-alarm rates, to be determined a priori. This would be useful in deciding deployment approaches.

Another area in which work is needed is the development of a suitable theory of intrusive behavior that could be used to guide the design of detectors that can abstract concrete behaviors sufficiently well to detect classes of attacks rather than individual instantiations. It is not clear that such a theory is feasible; but, if it is, it might offer an opportunity for significant progress.

One of the more frustrating aspects of intrusion detection is that many of the common attacks are enabled by easily avoidable vulnerabilities. The kinds of errors that are rampant in commonly used software should not be present at all.

Acknowledgements. I want to thank many members of the intrusion detection community for the insights that have made this paper possible. Roy Maxion of CMU has been a constant source of inspiration and support. Dick Kemmerer and Giovanni Vigna of UCSB and Phil Porras of SRI International have provided useful inputs, as well. Stefan Axelsson of Chalmers University provided valuable insights into intrusion detection as a signal detection problem and has been extremely generous in sharing his extensive bibliography and other works. Julia Allen, Tom Longstaff, and Rich Pethia of CERT/CC have been most supportive. Richard Lippmann and the other members of the evaluation effort at Lincoln Laboratory provided the opportunity for my work on IDS evaluation. I am fortunate in knowing many of the pioneers in the IDS field. Jim Anderson provided additional background on his early reports. Ann Marmor-Squires and Peter Neumann provided additional background on early work at TRW and SRI. Teresa Lunt and Dorothy Denning have been helpful on numerous past occasions. This work was sponsored by the United States Department of Defense.

References

1. Alessandri D (2000) Using rule-based activity descriptions to evaluate intrusion-detection systems. In: Raid 2000, number 1907 in LNCS. Springer, Berlin Heidelberg New York, pp 183–196
2. Allen J, Christie A, Fithen W, McHugh J, Pickel J, Stoner E (2000) State of the practice of intrusion detection technologies. Technical Report CMU/SEI–99–TR-028, Carnegie Mellon University, Software Engineering Institute
3. Anderson JP (1972) Computer security technology planning study, Vol. I. Technical Report ESC–TR–73–51, ESD/AFSC, Hanscom AFB, Bedford. [NTIS AD-758-206] - Available online at `http://seclab.cs.ucdavis.edu/projects/history/CD/ande72.pdf`
4. Anderson JP (1972) Computer security technology planning study, Vol. II. Technical Report ESC–TR–73–51, ESD/AFSC, Hanscom AFB, Bedford. [NTIS AD-758-206] – Available online at `http://seclab.cs.ucdavis.edu/projects/history/CD/ande72.pdf`
5. Anderson JP (1980) Computer security threat monitoring and surveillance. Technical report, James P Anderson Co, Fort Washington. Available online at `http://seclab.cs.ucdavis.edu/projects/history/CD/ande80.pdf`
6. Anderson JP (2000) Personal communication
7. Arbaugh WA, Fithen WL, McHugh J (2000) Windows of vulnerability: A case study analysis. IEEE Comput 33(12):52–59
8. Axelsson S (2000) Intrusion detection systems: A taxomomy and survey. Technical Report 99–15, Dept of Computer Engineering, Chalmers University of Technology, Göteborg
9. Axelsson S (2000) A preliminary attempt to apply detection and estimation theory to intrusion detection. Technical Report 00–4, Dept of Computer Engineering, Chalmers University of Technology, Göteborg
10. Bace RG (2000) Intrusion Detection. Technology Series. Macmillan, London
11. Bellovin SM (1989) Security problems in the TCP/IP protocol suite. Comput Commun Rev 19(2):32–48
12. Brunner J (1975) The Shockwave Rider. Ballantine, New York
13. CERT (2000) Cert security improvement modules. Available online at `http://www.cert.org/security-improvement/`
14. CERT Coordination Center (1991) Cert advisory CA-91.20. rdist.vulnerability. Available online at `http://www.cert.org/advisories/CA-91.20.rdist.vulnerability.html`. Superseded by CA-96-14 [17]
15. CERT Coordination Center (1992) Cert advisory CA-1992-03 Internet intruder activity. Available online at `http://www.cert.org/advisories/CA-1992-03`
16. CERT Coordination Center (1995) Cert advisory CA-1995-01 IP spoofing attacks and hijacked terminal connections. Available online at `http://www.cert.org/advisories/CA-1995-01`
17. CERT Coordination Center (1996) Cert advisory CA-1996-14 Vulnerability in rdist. Available online at `http://www.cert.org/advisories/CA-1996-14.html`
18. Cisco Systems, Inc (2000) Cisco secure intrusion detection system overview. Available online at `http://www.cisco.com/univercd/cc/td/doc/product/iaabu/csids/csids1/csidsug/overview.htm`
19. Cohen F (1985) Computer Viruses. ASP Press, Pittsburgh
20. Crosbie M, Dole B, Ellis T, Krsul I, Spafford E (1996) IDIOT users guide. Technical Report TR-96-050, The COAST

Project, Dept of Computer Science, Purdue University, West Lafayette
21. Debar H, Becker M, Siboni D (1992) A neural network component for an intrusion detection system. In: Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland. IEEE Computer Society Press, Los Alamitos, pp 40–250
22. Debar H, Dacier M, Wespi A, Lampart S (1998) An experimentation workbench for intrusion detection systems. Research Report RZ–2998 (# 93044), IBM Research Division, Zurich Research Laboratory
23. Debar H (1999) Testing intrusion detection systems. Presentation to Groupe OSSIR, July 1999. Available online at `http://www.ossir.org/ftp/supports/99/debar/index1.html`
24. Denning D (1986) An intrusion detection model. In: Proceedings of the 1986 IEEE Symposium on Security and Privacy. IEEE Press, pp 119–131
25. Denning DE (1987) An intrusion detection model. IEEE Trans Software Eng SE-13(2):222–232
26. Denning DE, Edwards DL, Jagannathan R, Lunt TF, Neumann PG (1987) A prototype IDES: A real-time intrusion-detection expert system. Technical report, Computer Science Laboratory, SRI International, Menlo Park
27. Department of Defense (1985) Trusted computer system evaluation criteria. DoD 5200.28-STD. Also known as the "Orange Book". Available online at `http://seclab.cs.ucdavis.edu/projects/history/CD/dod85.pdf`
28. Dowel C, Ramstedt P (1990) The computer watch data reduction tool. In: Proceedings of the 13th National Computer Security Conference, Washington, DC, October 1990, pp 99–108
29. Durst R, Champion T, Witten B, Miller E, Spanguolo L (1999) Testing and evaluating computer intrusion detection systems. Commun ACM 42(7):53–61
30. Eckmann ST, Vigna G, Kemmerer R (2000) STATL: An Attack Language for State-based Intrusion Detection. In: Proceedings of the ACM Workshop on Intrusion Detection, Athens, November 2000. ACM
31. Egan JP (1975) Signal detection Theory and ROC Analysis. Academic, New York
32. Flack C, Atallah MJ (2000) Better logging through formality: Applying formal specification techniques to improve audit logs and log consumers. In: Recent Advances in Intrusion Detection, Third International Workshop, RAID 2000, number 1907 in LNCS, Toulouse. Springer, Berlin Heidelberg New York, pp 1–16
33. Goldberg I, Wagner D, Thomans R, Brewer E (1996) A secure environment for untrusted helper applications (confining the wily hacker). In: The Sixth USENIX Security Symposium Proceedings, San Jose, Calif., July 1996. USENIX Association, Berkeley
34. Habra J, Le Charlier B, Mounji A, Mathieu I (1992) ASAX: Software architecture and rule-based language for universal audit trail analysis. In: Proceedings of ESORICS 92, Vol. 648 in LNCS, Toulouse, November 1992. Springer, Berlin Heidelberg New York, pp 435–450
35. Harrenstein K (1997) RFC 0742 – NAME/FINGER. Internet Engineering Task Force Request for Comments, December. Internet Society, Reston
36. Heberlein T, Dias G, Levitt K, Mukherjee B, Wood J, Wolber D (1990) A network security monitor. In: Proceedings of the 1990 IEEE Symposium on Security and Privacy, Oakland. IEEE Computer Society Press, pp 296–304
37. Hernan SV (2000) Personal communication
38. Hochberg J, Jackson K, Stallings C, McClary JF, DuBois D, Ford J (1993) Nadir, an automated system for detecting network intrusion and misuse. Comput Secur 12(3):235–248
39. Ilgun K (1993) USTAT: A real-time intrusion detection system for Unix. In: Proceedings of the 1993 IEEE Symposium on Security and Privacy, Oakland. IEEE Computer Society Press, pp. 16–28
40. Jacobson V, Leres C, McCanne S (1994) libpcap. Lawrence Berkeley National Laboratory. Available at `http://www-nrg.ee.lbl.gov/`
41. Jagannathan R, Lunt TF, Anderson D, Dodd C, Gilham F, Jalali C, Javitz HS, Neumann PG, Tamaru A, Valdes A (1993) System Design Document: Next-generation intrusion-detection expert system (NIDES). Technical report, Computer Science Laboratory, SRI International, Menlo Park
42. Jou YF, Gong F, Sargor C, Wu SF, Cleaveland RW (1997) Architecture design of a scalable intrusion detection system for the emerging network infrastructure. Technical Report CDRL A005, Dept of Computer Science, North Carolina State University, Raleigh
43. Julisch K (2000) Dealing with false positives in intrusion detection. Presentation at RAID 2000. Extended abstract available online at `http://www.raid-symposium.org/raid2000/Materials/Abstracts/50/50.pdf`
44. Karger PA, Schell RR (1974) MULTICS security evaluation: Vulnerability analysis. Technical Report ESD-TR-74-193, Vol. II, ESD/AFSC, Hanscom AFB, Bedford. Available online at `http://seclab.cs.ucdavis.edu/projects/history/CD/karg74.pdf`
45. Kim G, Spafford EH (1993) The design of a system integrity monitor: Tripwire. Technical Report CSD-TR-93-071, COAST-TR 93-01. Dept of Computer Sciences, Purdue University, West Lafayette
46. Ko C (1996) *Execution Monitoring of Security-critical Programs in a Distributed System: A Specification-based Approach*. PhD thesis, Dept of Computer Science, University of California at Davis
47. Ko C, Ruschitzka M, Levitt K (1997) Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In: Proceedings of the 1997 IEEE Symposium on Security and Privacy, Oakland. IEEE Computer Society Press, pp 175–187
48. Krsul IV (1998) Software Vulnerability Analysis. PhD thesis, Purdue University, West Lafayette
49. Lee W, Stolfo S, Mok K (1999) A data mining framework for building intrusion detection models. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy, Oakland. IEEE Computer Society Press, pp 120–132
50. Lippmann R, Fried DJ, Graf I, Haines JW, Kendall KR, McClung D, Weber D, Webster SH, Wyschograd D, Cunningham RK, Zissman MA (2000) Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In: Discex 2000, Vol. 2. IEEE Computer Society Press, pp 12–26
51. Lippmann R, Haines JW, Fried DJ, Korba J, Das K (2000) The 1999 DARPA off-line intrusion detection evaluation. In: Raid 2000, number 1907 in LNCS. Springer, Berlin Heidelberg New York, pp 162–182
52. Lippmann RP, Fried DJ, Haines JW, Graf I, Kendall KR, McClung D, Weber D, Webster SE, Cunningham RK, Wyschogrod D, Zissman MA (2000) Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In: Proceedings of DARPA Information Survivability Conference and Exposition, Vol. 2. IEEE Computer Society Press, Los Alamitos, pp 12–26
53. Lunt TF (1988) Automated audit-trail analysis and intrusion detection: A survey. In: Proceedings of the Eleventh National Computer Security Conference, pp 188–193
54. Maxion RA, Tan KMC (2000) Benchmarking anomaly-based detection systems. In: International Conference on Dependable Systems and Networks. IEEE Computer Society Press, Los Alamitos, pp 623–630
55. McHugh J (2000) The 1998 Lincoln Lab IDS evaluation – a critique. In: Raid 2000, number 1907 in LNCS. Springer, Berlin Heidelberg New York, pp 145–161
56. Morris RT, Sr (1988) Personal communication. This anecdote was told to the author and to George Dinolt of Ford Aerospace when they visited Morris at the NCSC to brief him on some research results. We found it particularly interesting in light of the events of the following November
57. Mukherjee B, Heberlein LT, Levitt KN (1994) Network intrusion detection. IEEE Network 8(3):26–41
58. Myers P (1980) Subversion: The neglected aspect of computer security. Master's thesis, Naval Postgraduate School, Mon-

terey. Available online at http://seclab.cs.ucdavis.edu/projects/history/CD/myer80.pdf
59. Aleph One (1996) Smashing the stack for fun and profit. Phrack 7(47):File 14 of 16. Available online at, for example, http://www.shmoo.com/phrack/Phrack49/p49-14 and http://www.codetalker.com/whitepapers/other/p49-14.html
60. Paxon V (1988) Bro: A system for detecting network intruders in real-time. In: Proceedings of the 7th USENIX Security Symposium, San Antonio. USENIX Association, Berkeley, pp 31–51 (Corrected version, original overstated the traffic level on the FDDI ring by a factor of two)
61. Porras PA, Neumann PG (1997) Emerald: Event monitoring enabling responses to anomalous live disturbances. In: Proceedings of the 20th National Information Systems Security Conference. pp 353–365
62. Ptacek TH, Newsham TN (2000) Insertion, evasion, and denial of service: Eluding network intrusion detection. Report from Network Associates Laboratories, 1998. This paper can often be found online by searching for the first author. As of November, it was available online at http://www.secinf.net/info/ids/idspaper/idspaper.html. It has also been seen in the publications directory of http:www.snort.org
63. Puketza N, Zhang K, Chung M, Mukherjee B, Olsson RA (1996) A methodology for testing intrusion detection systems. IEEE Trans Software Eng SE-22(10):719–729. Authors are with the University of California, Davis. Available online at http://seclab.cs.ucdavis.edu/papers.html
64. Puketza N, Chung M, Olsson RA, Mukherjee B (1997) A software platform for testing intrusion detection systems. IEEE Software 14(5):43–51. Available online at http://seclab.cs.ucdavis.edu/papers.html
65. Reed B (1987) Reflections on some recent widespread computer break-ins. CACM 30(2):103–105 ("Viewpoint" column)
66. Roesch M (1999) Snort – lightweight intrusion detection for networks. In: Proceedings of USENIX LISA 99. USENIX Association, Berkeley, pp 229–238. Also available online at http://www.snort.org (Documents page)
67. Sebring MM, Shellhouse E, Hanna ME, Whitehurst RA (1988) Expert systems in intrusion detection: A case study. In: Proceedings of the Eleventh National Computer Security Conference, Washington, DC, October 1988, pp 74–81
68. Seeley D (1989) A tour of the worm. In: Proceedings of the 1989 Winter USENIX Conference, San Diego, February 1989. Usenix Association, Berkeley, pp 287–304
69. Shoch JF, Hupp JA (1982) The "worm" programs – early experience with a distributed computation. CACM 25(3):172–180
70. Slade RM (1992) History of computer viruses. Available online at http://www.bocklabs.wisc.edu/janda/sladehis.html
71. Smaha SE (1988) An intrusion detection system for the Air Force. In: Proceedings of the Fourth Aerospace Computer Security Applications Conference, Orlando. IEEE Computer Society Press, Los Alamitos, pp 37–44
72. Snapp SR, Smaha SE, Teal DM, Grance T (1992) The DIDS (Distributed Intrusion Detection System) prototype. In: Proceedings of the Summer USENIX Conference. USENIX Association, Berkeley, pp 227–233
73. Spaford EH (1989) The internet worm: Crisis and aftermath. CACM 32(6):678–687
74. Stryker D (1974) Subversion of a "secure" operating system. Memorandum Report 2821, Naval Research Laboratory, Washington
75. Tenner WT (1990) Discovery: An expert system in the commercial data security environment. Comput Secur J 6(1):45–53
76. Tyson M (1999) DERBI (Diagnosis, Explanation and Recovery from computer Break-Ins). Abstract for seminar available online at http://www.ai.sri.com/ai-seminars/abstracts/tyson-990506.html
77. Vaccaro HS, Liepins GE (1989) Detection of annomolous computer session activity. In: Proceedings of the 1989 IEEE Symposium on Security and Privacy, Oakland, IEEE Computer Society Press, Los Alamitos, pp 280–289
78. Vigna G, Kemmerer RA (1998) NetSTAT: A network-based intrusion detection approach. In: Proceedings of the 14^{th} Annual Computer Security Application Conference, Scottsdale
79. Vigna G, Kemmerer RA (1999) NetSTAT: A network-based intrusion detection system. J Comput Secur 7(1):37–71
80. Vigna G, Eckmann ST, Kemmerer RA (2000) The STAT Tool Suite. In: Proceedings of DARPA Information Survivability Conference and Exposition, Vol. 2. IEEE Press, Los Alamitos, pp 46–55
81. Ware W (1970) Security controls for computer systems (U): Report of Defense Science Board task force on computer security. Rand Report R609–1, The RAND Corporation, Santa Monica Available online at http://seclab.cs.ucdavis.edu/projects/history/CD/ware70.pdf
82. Warrender C, Forrest S, Perlmutter B (1999) Detecting intrusions using system calls: Alternative data models. In: IEEE Symposium on Security and Privacy, Berkeley, pp 133–145
83. Zimmerman D (1991) RFC 1288 – The finger user information protocol. Internet Engineering Task Force Request for Comments. Internet Society, Reston