Marchette, David J.

Working Paper

Network intrusion detection

Papers, No. 2004,34

Provided in Cooperation with:
CASE - Center for Applied Statistics and Economics, Humboldt University Berlin

Suggested Citation: Marchette, David J. (2004) : Network intrusion detection, Papers, No. 2004,34, Humboldt-Universität zu Berlin, Center for Applied Statistics and Economics (CASE), Berlin

This Version is available at:
https://hdl.handle.net/10419/22207

1

Network Intrusion Detection

David J. Marchette

Naval Surface Warfare Center

1.1 Introduction

Attacks against computers and the Internet are in the news every week. These primarily take the form of malicious code such as viruses and worms, or denial of service attacks. Less commonly reported are attacks which gain access to computers, either for the purpose of producing damage (such as defacing web sites or deleting data) or for the opportunities such access provides to the attacker, such as access to bank accounts or control systems of power stations. This chapter will discuss some of the areas in which computational statistics can be applied to these and related problems.

Several books are available that describe the basic ideas in intrusion detection. These include [1], [3], [4], [8], [18], [25] and [28]. Intrusion detection is typically split into two separate problems. Network intrusion detection typically looks at traffic on the network, while host based intrusion detection involves collecting data on a single host. Both involve very large and complex data sets, and both have aspects that lend themselves to statistical solutions. We will only touch on a few such; the reader is encouraged to investigate the references.

There are two basic approaches to network intrusion detection. Most existing systems rely on signatures of attacks. This approach relies on some set of features that can be extracted from the data that indicate the existence of an attack. This is analogous to the virus scanners, which look for a sequence of bytes that are indicative of a virus. In the network realm, this could be attempts to access services that are denied, malformed packets, too many failed attempts to log in, et cetera. The second approach is anomaly detection. The "normal" activity of the network is modeled, and outliers are indicative of attacks. The definition of "normal" is dependent on the type of attacks that one is interested in, and requires statistical models.

This chapter will first describe the basics of the TCP/IP protocol, sufficient to understand the data and the examples given. Then we will look at detecting denial of service attacks, and estimating the number of attacks on the Internet.

Network data is streaming data, and we will discuss this and some areas in which computational statistics can play a part. This will lead to a discussion of simple visualization techniques applied to network data, with some discussion of the types of insights that can be gained from this. We will then take a detour from network data and consider profiling. This will illustrate a type of anomaly detection, which will then be discussed within a network context.

1.2 Basic TCP/IP

When you visit a web site, your request and the response data are sent as a series of packets, each consisting of a header containing addressing and sequencing information, and a payload or data section in which the information resides. Packets are typically relatively small (less than 1500 bytes). In order to analyze the traffic and detect attacks, one needs to collect the packets, and may need to process either the header or the payload. We will (somewhat arbitrarily) denote an attack that can be detected by investigating the header only a "network attack" while leaving those that require investigation of the payload in the "host attack" realm.

One reason for this distinction is encryption. If the data are encrypted (for example, data from a secure web site), the header remains in the clear, and so this information is still available for analysis by the statistician. The payload is inaccessible (assuming a sufficiently strong encryption scheme) and so cannot be used to detect attacks until it is decrypted at the destination host. For this reason (and others), we consider any attack that requires investigation of the data in a packet to be better detected at the host than on the network.

There are several protocols used on the Internet to ensure a level of performance or reliability in the communication. We will briefly discuss TCP (the Transmission Control Protocol), since it is one of the most important ones, and will allow us to discuss a class of denial of service attacks. For more information about the various protocols, see [32].

First, however, it is necessary that we discuss the Internet Protocol (IP). This protocol is not reliable, in the sense that there is no mechanism in place to ensure that packets are received. The IP header contains the source and destination IP addresses, which are 32-bit integers identifying the sending and receiving computer for the packet. There are other fields in the packet that are used to control the routing of the packet, et cetera, but we will not dwell on these here. As always, interested readers should investigate [32] or any of the many books on the TCP/IP protocol suite.

Since IP is unreliable, a packet sent may or may not reach its destination, and if it does not, there is no guarantee that anyone will notice. Thus, a more reliable protocol is required. TCP implements a reliable two way communication channel, and is used for web, email, and many other user applications. The TCP header is shown in Figure 1.1. The important fields, for this discussion, are the ports, sequence numbers and flags.

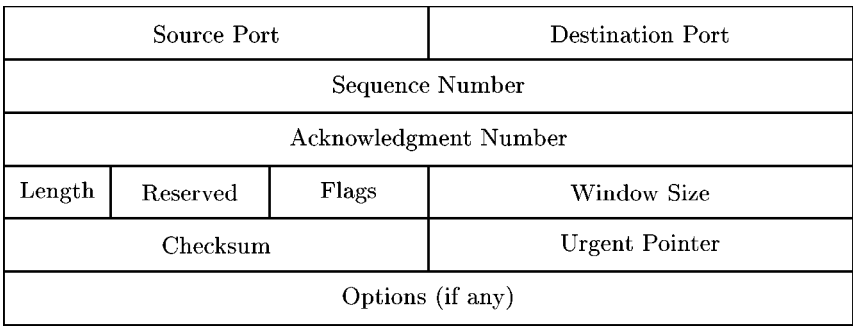

Fig. 1.1. The TCP header. The header is to be read left to right, top to bottom. A row corresponds to 32 bits.

The ports are a method for identifying a specific session, and can be thought of as a 16-bit addition to the IP address that uniquely determines the session. Ports are also used to identify the application requested. For example, port 80 is the standard web port, and web browsers know that in order to obtain a web page from a server they need to make a connection on this port.

To initiate and maintain a connection, the flags and sequence numbers are used. The TCP protocol requires a three-way handshake to initiate a connection. First the client sends a SYN packet (in this manner we will denote a packet with only the SYN flag set; similarly with other flag combinations) to the server. The server responds with a SYN/ACK packet, acknowledging the connection. The client then finalizes the connection with an ACK packet. Sequence numbers are also passed, and tracked to ensure that all sent packets are received and acknowledged, and to allow the reconstruction of the session in the correct order. Packets that are not acknowledged are resent, to ensure that they are ultimately received and processed.

Once a session has been instantiated through the three-way handshake, packets are acknowledged with packets in which the ACK flag is set. In this manner the protocol can determine which packets have been received and which need to be resent. If a packet has not been acknowledged within a given time, the packet is resent, and this can happen several times before the system determines that something has gone wrong and the session is dropped (usually by sending a reset (RST) packet). Note that this means that if there is no response to the SYN/ACK packet acknowledging the initiation of the session there will be a period (of several seconds) in which the session is kept open by the destination host as it tries resending the SYN/ACK hoping for a response. This is the basis of some denial of service attacks, which we will discuss in the next section.

1.3 Passive Sensing of Denial of Service Attacks

The TCP protocol provides a simple (and popular) method for denial of service attacks. The server has a finite number of connections that it can handle at a time, and will refuse connections when its table is full. Thus, if an attacker can fill the table with bogus connections, legitimate users will be locked out.

This attack relies on two fundamental flaws in the protocols. The first is that the source IP address is never checked, and thus can be "spoofed" by putting an arbitrary 32 bit number in its place. Second, the three-way handshake requires the third (acknowledgment) packet, and the server will wait several seconds before timing out a connection. With each requested connection, the server allocates a space in its table and waits for the final acknowledgment (or for the connection to time out). The attacker can easily fill the table and keep it filled by sending spoofed SYN packets to the server.

Thus, the attacker sends many SYN packets to the server, spoofed to appear to come from a large number of different hosts. The server responds with SYN/ACK packets to these hosts, and puts the connection in its table to await the final ACK, or a time-out (usually several seconds). Since the ACK packets are not forthcoming, the table quickly fills up, and stays full for as long as the attacker continues to send packets.

There are clever ways to mitigate this problem, which can keep the table from filling up. One, the "SYN-cookie" involves encoding the sequence number of the SYN/ACK in a way that allows the server to recognize legitimate ACK packets without needing to save a spot in the table for the connection. However, even these can be defeated through a sufficiently high volume attack.

These unsolicited SYN/ACK packets can be observed by any network sensor, and thus provide a method for estimating the number and severity of such attacks throughout the Internet. These unsolicited packets are referred to as *backscatter*. They may take other forms than SYN/ACK packets, depending on the type of packet sent in the attack. See [24, 19, 17] for more information.

Typically, the attacker first compromises a large number of computers, using special distributed attack software, and it is these computers that launch the attack. This makes it very difficult to block the attack, and essentially impossible to track down the attacker, at least through information available to the victim.

Backscatter packets provide several opportunities for statistical analysis. They allow the estimation of the number of attacks on the Internet in real time. One may be able to estimate the severity of the attacks and number of attackers. Finally, it may be possible to characterize different types of attacks or different attack tools and identify them from the pattern of the packets. Some initial work describing some of these ideas is found in [12].

A network sensor is a computer that captures packets (usually just the packet headers) as they traverse the network. These are usually placed either just before or just after a firewall to collect all the packets coming into a net-

work. Through such a system, one can observe all the unsolicited SYN/ACK packets addressed to one of the IP addresses owned by the network.

Note that this means that only a fraction of the backscatter packets resulting from the attack are seen by any sensor. If we assume that the sensor is monitoring a class B network (an address space of 65,536 IP addresses), then we observe a random sample of 1/65,536 of the packets, assuming the attack selects randomly from all 2^{32} possible IP addresses. This points to several areas of interest to statisticians: we observe a subset of the packets sent to a subset of the victims, and wish to estimate the number of victims, the number of packets sent to any given victim, and the number of attackers for any given victim.

1.4 Streaming Data

Network packets are streaming data. Standard statistical and data mining methods deal with a fixed data set. There is a concept of the size of the data set (usually denoted n) and algorithms are chosen based in part on their performance as a function of n. In streaming data there is no n: the data are continually captured and must be processed as they arrive. While one may collect a set of data to use to develop algorithms, the nonstationarity of the data requires methods that can handle the streaming data directly, and update their models on the fly.

Consider the problem of estimating the average amount of data transfered in a session for a web server. This is not stationary: there are diurnal effects; there may be seasonal effects (for example at a university); there may be changes in the content at the server. We'd like a number calculated on a window of time that allows us to track (and account for) the normal trends and detect changes from this normal activity.

This requires some type of windowing or recursive technique. The recursive version of the sample mean is well known:

$$\bar{X}_n = \frac{n-1}{n}\bar{X}_{n-1} + \frac{1}{n}X_n.$$

Replacing n on the right hand side with a fixed constant N implements an exponential window on the mean. This was exploited in the NIDES intrusion detection system ([2]). Similar techniques can be used to compute other moments. An alternative formulation is:

$$\hat{X}_n = (1-\theta)X_{n+1} + \theta\hat{x}_{n-1},$$

for $0 < \theta < 1$. θ may be fixed or may itself change based on some statistic of the data.

In fact, the kernel density estimator has a simple recursive version, that allows the recursive estimate of the kernel density estimator at a fixed grid of points. [39, 34] give two versions of this:

$$\hat{f}_n(x) = \frac{n-1}{n}\hat{f}_{n-1}(x) + \frac{1}{nh_n}K\left(\frac{x-X_n}{h_n}\right)$$

$$\check{f}_n(x) = \frac{n-1}{n}\left(\frac{h_{n-1}}{h_n}\right)^{\frac{1}{2}}\check{f}_{n-1}(x) + \frac{1}{nh_n}K\left(\frac{x-X_n}{h_n}\right).$$

In either case, fixing n at a constant and h_n either at a constant or a recursively estimated value implements an exponentially windowed version of the kernel estimator. (Similarly, one can phrase this in terms of θ as was done with the mean; see [36]). These can in turn be used to estimate the "normal" activity of various measurements on the network, and provide a mechanism for detecting changes from normal, which in turn may indicate attacks. More information on these issues can be found in [36].

Similar approaches can be implemented for other density estimation techniques. In particular, the adaptive mixtures approach of [27] has a simple recursive formulation that can be adapted to streaming data.

There are several applications of density estimation to intrusion detection that one might consider. It is obvious that unusually large downloads (or uploads) may be suspicious in some environments. While it is not clear that density estimation is needed for this application, there might be some value in detecting changes in upload/download behavior. This can be detected through the tracking of the number of bytes transfered per session.

Perhaps a more compelling application is the detection of trojan programs. A trojan is a program that appears to be a legitimate program (such as a telnet server) but acts maliciously, for example to allow access to the computer by unauthorized users. Obviously the detection of trojans is an important aspect of computer security.

Most applications (web, email, ftp, et cetera) have assigned ports on which they operate. Other applications may choose to use fixed ports, or may choose any available port. Detecting new activity on a given port is a simple way to detect a trojan program. More sophisticated trojans will replace a legitimate application, such as a web server. It is thus desirable to determine when a legitimate application is acting in a manner that is unusual.

Consider Figure 1.2. We have collected data for two applications (web and secure shell) over a period of 1 hour, and estimated the densities of the packet length and inter arrival times. As can be seen, the two applications have very different patterns for these two measures. This is because they have different purposes: secure shell is a terminal service which essentially sends a packet for every character typed (there is also a data transfer mode to secure shell, but this mode was not present in these data); web has a data transfer component with a terminal-like user interaction.

By monitoring these and other parameters, it is possible to distinguish between many of the common applications. This can then be used to detect when an application is acting in an unusual manner, such as when a web server is being used to provide telnet services. See [7] for a more extensive discussion of this.

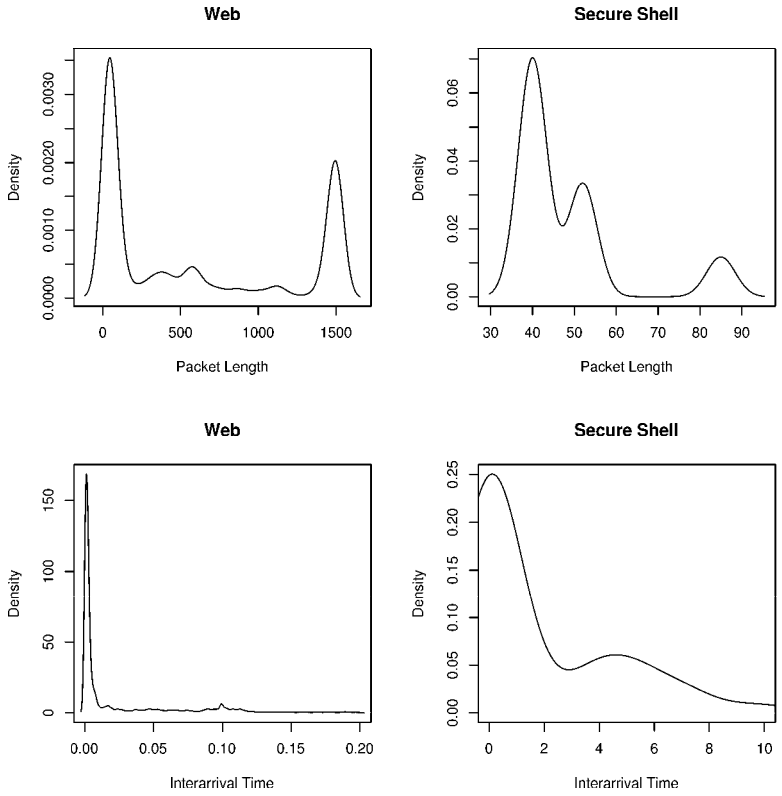

Fig. 1.2. Packet length in bytes (top) and packet inter arrival times in seconds (bottom) for web (left) and secure shell (right) sessions. Kernel estimators were used to estimate the densities. The inter arrival times were truncated to show the bulk of the data.

Note that web traffic has two main peaks at either end of the extremes in packet size. These are the requests, which are typically small, and the responses, which are pages or images and are broken up into the largest packets possible. The mass between the peaks mostly represent the last packets of transfers which are not a multiple of the maximum packet size, and small transfers that fit within a single packet.

The inter packet arrival times for secure shell also have two peaks. The short times correspond to responses (such as the response to a directory list command) and to characters typed quickly. The later bump probably corresponds to the pauses between commands, as the user processes the response. These arrival times are very heavy tailed because of the nature of secure shell. Sessions can be left open indefinitely, and if no activity occurs for a sufficiently long time, "keep alive" packets are sent to ensure that the session is still valid.

In [7] it is shown, in fact, that differences in the counts for the TCP flags can be used to differentiate applications. These, combined with mean inter packet arrival times and packet lengths (all computed on a window of n packets for various values of n), do a very creditable job of distinguishing applications. This is clearly an area in which recursive methods like those mentioned above would be of value. It also is reasonable to hypothesize that estimating densities, rather then only computing the mean, would improve the performance.

By detecting changes in the densities of applications it may be possible to detect when they have been compromised (or replaced) by a trojan program. It may also be possible to detect programs that are not performing as advertised (web servers acting like telnet servers, for example).

1.5 Visualization

Visualization of complex data is important but difficult. This is especially true of streaming data. While many complex techniques for visualization have been developed, simple scatter plots can be used effectively, and should not be shunned.

Figure 1.3 shows a scatter plot of source port against time for an 8 hour period of time. These are all the SYN packets coming in to a class B network (an address space of $65,536$ possible IP addresses). This graphic, while simple, provides quite a few interesting insights.

Note that there are a number of curves in the plot. These are a result of the fact that each time a client initiates a session with a server, it chooses a new source port, and this corresponds to the previous source port used by the client incremented by one. Contiguous curves correspond to connections by a single source IP. Vertical gaps in the curves indicate that the IP visited other servers between visits to the network. It is also easy to see the start of the work day in this plot, indicated by the heavy over plotting on the right hand side.

The source ports range from 1024 to $65,536$. Different applications and operating systems select ports from different ranges, so one can learn quite a bit from investigating plots like this.

The plot of Figure 1.3 is static. Figure 1.4 is meant to illustrate a dynamic plot. This is analogous to the waterfall plots used in signal processing. It displays a snapshot in time that is continuously updated. As new observations are obtained they are plotted on the right, with the rest of the data shifting left, dropping the left most column. Plots like this are required for streaming data.

Simple plots can also be used to investigate various types of attacks. In Figure 1.5 is plotted spoofed IP address against time for a denial of service attack against a single server. Each point corresponds to a single unsolicited

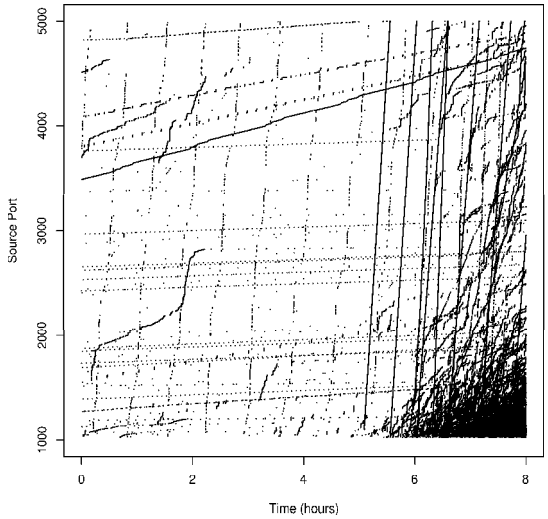

Fig. 1.3. Source port versus time for all the incoming SYN packets for an 8 hour period.

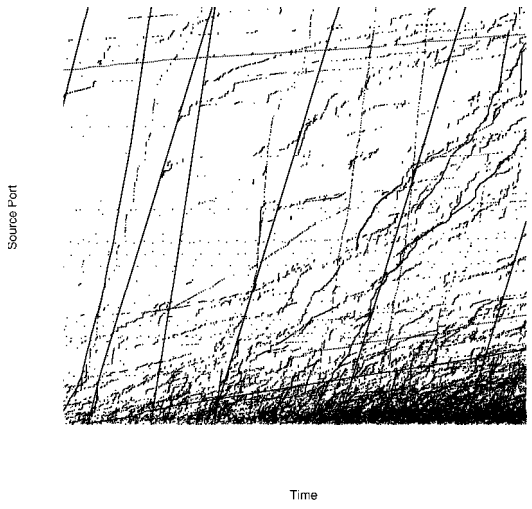

Fig. 1.4. Source port versus time for a short time period, the last two hours from Figure 1.3. As time progresses, the plot shifts from right to left, dropping the left most column and adding a new column on the right.

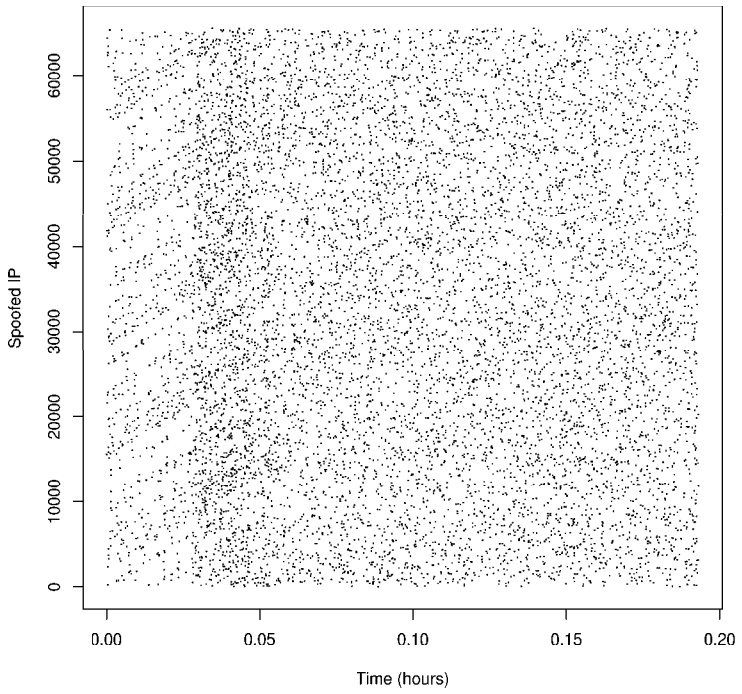

Fig. 1.5. Plot of spoofed IP address against time for backscatter packets from a denial of service attack against a single server. The IP addresses have been converted to 16-bit numbers, since in this case they correspond to the final two octets of the IP address.

SYN/ACK packet received at the sensor from a single source. This plot provides evidence that there where actually two distinct attacks against this server. The left side of the plot shows a distinctive stripped pattern, indicating that the spoofed IP addresses have been selected in a systematic manner. On the right, the pattern appears to be gone, and we observe what looks like a random pattern, giving evidence that the spoofed addresses are selected at random (a common practice for distributed denial of service tools). Between about 0.03 and 0.06 there is evidence of overlap of the attacks, indicating that this server was under attack from at least two distinct programs simultaneously.

Another use of scatter plots for analysis of network data is depicted in Figure 1.6. These data were collected on completed sessions. The number of packets is plotted against the number of bytes. Clearly there should be a (linear) relationship between these. The interesting observation is that there

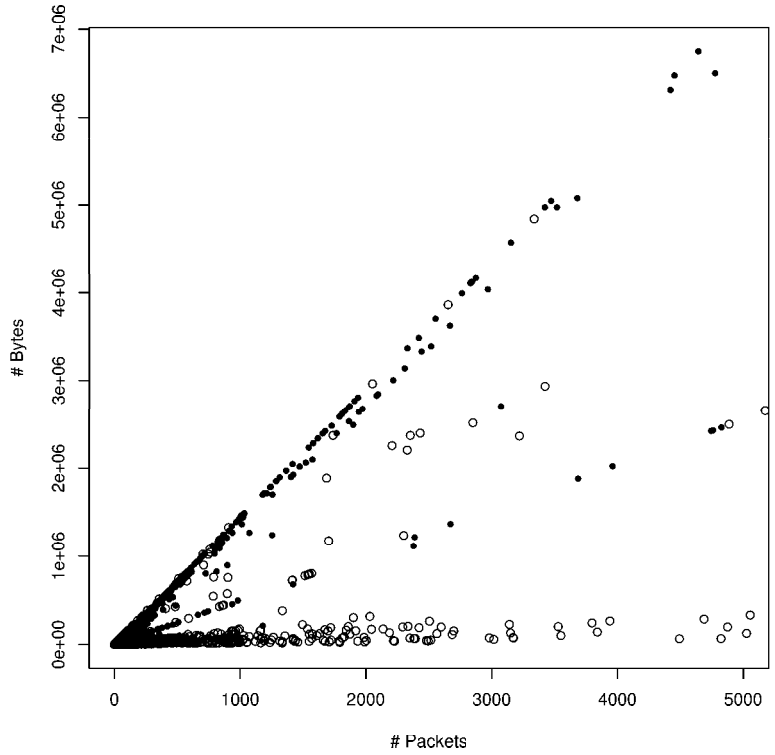

Fig. 1.6. Number of bytes transfered within a completed session plotted against the number of packets within the session. Solid dots correspond to email sessions, circles correspond to all other applications.

are several linear relationships. This is similar to the observations made about Figure 1.2, in which it was noted that different applications use different packet lengths.

Figure 1.7 shows the number of bytes transfered within a session plotted against the start time of the session. There is a lot of horizontal banding in this plot, corresponding mostly to email traffic. It is unknown whether the distinctive repetitive patterns are a result of spam (many email messages all the same size) or whether there are other explanations for this. Since these data are constructed from packet headers only, we do not have access to the payload and cannot check this hypothesis for these data. Figure 1.8 shows a zoom of the data. The band just below 400 bytes correspond to telnet sessions. These are most likely failed login attempts. This is the kind of thing that one would like to detect. The ability to drill down the plots, zooming and selecting observations to examine the original data, is critical to intrusion detection.

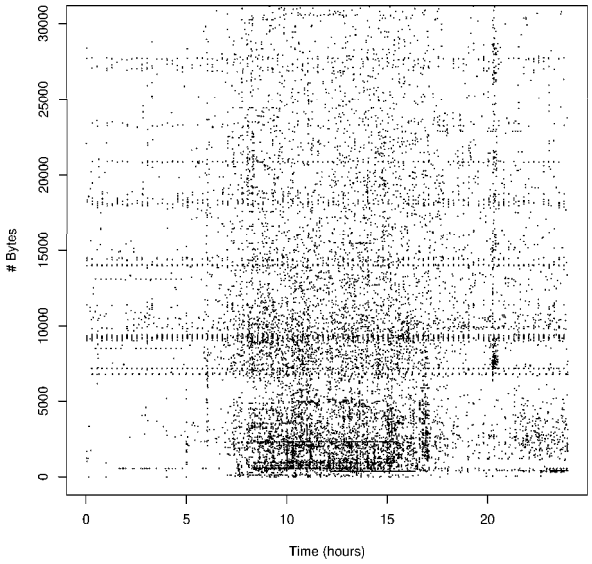

Fig. 1.7. Number of bytes transfered for each session plotted against the starting time of the session for a single day.

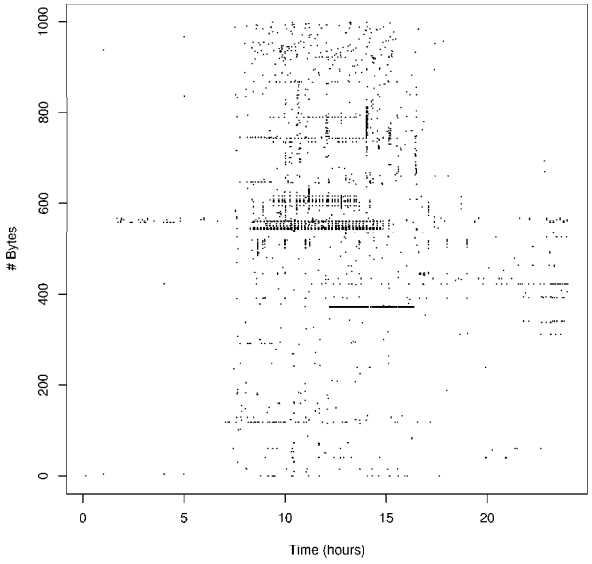

Fig. 1.8. The portion of the sessions in Figure 1.7 which were less than 1000 bytes.

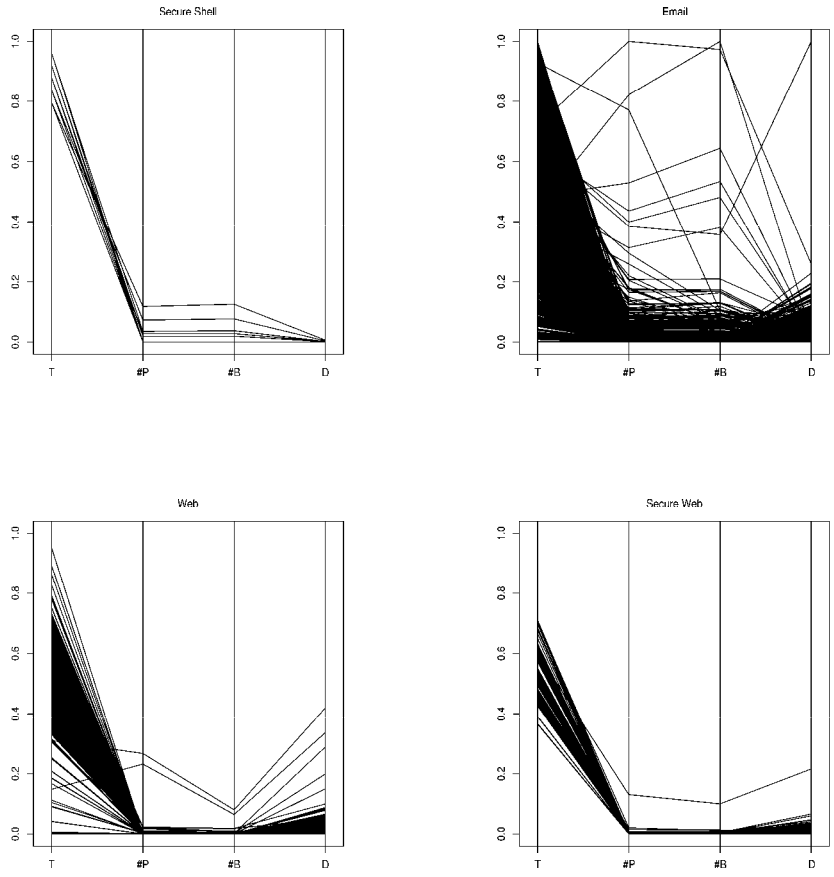

Fig. 1.9. Parallel coordinates plots of session statistics for four different applications. From left to right, top to bottom they are: secure shell, email, web and secure web. The coordinates are the time of the initiating SYN packet, the total number of packets, the total number of bytes sent and the duration of the session. The axes are all scaled the same among the plots.

High dimensional visualization techniques are clearly needed. Parallel coordinates is one solution to this. In Figure 1.9 we see session statistics for four different applications plotted using parallel coordinates.

One problem with plots like this is that of over plotting. Wegman solves this via the use of color saturation (see [35, 38]). Without techniques such as this it is extremely difficult to display large amounts of data. Figure 1.9 illustrates this problem in two ways. First, consider the secure shell data in

the upper left corner. It would be reasonable to conclude from this plot that secure shell sessions are of short duration, as compared with other sessions. This is an artifact of the data. For these data there are only 10 secure shell sessions, and they all happen to be of short duration. Thus, we really need to look at a lot of data to see the true distribution for this applications. Next, look at the email plot in the upper right. Most of the plot is black, showing extensive over plotting. Beyond the observation that these email sessions have heavy tails in the size and duration of the sessions, little can be gleaned from this plot.

A further point should be made about the web sessions. Some of the sessions which are relatively small in terms of number of packets and bytes transfered have relatively long durations. This is a result of the fact that often web sessions will not be closed off at the end of a transfer. They are only closed when the browser goes to another web server, or a time-out occurs. This is an interesting fact about the web application which is easy to see in these plots.

1.6 Profiling and Anomaly Detection

We will now briefly consider host based intrusion detection. While the data considered is not network data, the statistical techniques used are applicable to network problems, as will be discussed.

One of the important problems of computer security is user authentication. This is usually performed by requiring the user to type a password at the initial login. Once a user is logged in, there are generally no checks to ensure that the person using the terminal is still the authorized person. User profiling seeks to address this by extracting "person specific" information as the user interacts with the computer. By comparing the user's activity with a profile of the user, it is hoped that masqueraders can be detected and locked out before they are able to do any damage.

We will discuss the usual host-based user profiling problem first, and then discuss a network based profiling application that has a similar flavor. The mathematics and statistics used for the two problems are very similar, only the data are different.

Several attempts have been made on this problem. Early work focused on utilizing keystroke timings. It was hoped that people had characteristic patterns of typing that could be discovered through measurement of the time between keystrokes for words or phrases. See for example [5], [26], [16] and [29].

This type of approach has been applied at the network level to crack passwords. [31] describes using simple statistical techniques applied to packet arrival timings to determine the length of passwords in secure shell, and even to allow for the cracking of passwords. Secure shell is an application that allows remote login via an encrypted pathway. It sends a packet for each character typed, to minimize the delay for the user. Thus, by timing the packets, one

can get an idea of what key combinations are being sent (it takes longer to type two characters with the same finger than it does if the characters are typed by fingers on different hands, for example). By utilizing statistics such as these, the authors were able to show that they could dramatically reduce the search space needed to crack the passwords.

Other work focuses on tracking user commands. The idea is that the command streams that users type (ignoring the arguments to the commands) could be used to authenticate the user in much the same way that keystroke timings could. A good discussion of this for statisticians can be found in [30]. See also [22, 23] for some critiques of this work and extensions. The former paper considers arguments to the commands as well.

For Microsoft Windows operating systems, user command sequences are generally not applicable. Instead, window titles may be used. These correspond roughly to the same information that is contained in the Unix command lines. They typically contain the application name and the arguments to the applications such as the file open, the email subject, the web page visited, et cetera.

To illustrate this, we consider a set of data taken from six users on seven Windows NT machines over a period of several months. All window titles generated from the login to the logout were retained for each user/host pair (only one of the users was observed on a second host). Each time a window became active it was recorded. These data are a subset of a larger set. More information on these data, with some analysis of the data and performance of various classifiers can be found in [6].

Table 1.1 shows some statistics on these data. Three sessions are shown for each user/host pair. The length of the login session (in seconds), the name of the first and last applications used within the session, and the number of distinct applications, windows and window titles are shown. The task is to extract statistics from a completed login session that allow one to determine whether the user was the authorized user indicated by the userid. This is an easier problem than masquerader detection, in which one tries to detect the masquerader (or authenticate the user) as the session progresses, and it is not assumed that the entire session corresponds to a single user (or masquerader).

The table indicates that there is some variability among the sessions of individual users, and this is born out by further analysis. Table 1.2 shows the most common window titles. The number of times the title occurs in the data set, the number of login sessions in which the title occurs, and the title itself are shown. Some words in the titles have been obfuscated by replacement with numbers in double brackets, to protect the privacy of the users. All common application and operating system words were left alone. The obfuscation is consistent across all sessions: there is a bijection between numbers and words that holds throughout the data.

Figure 1.10 shows part of a single login session. The rows and columns correspond to the list of words (as they appear in the titles) and a dot is placed where the word appears in both the row and column. The blocks of

Table 1.1. Session statistics for three login sessions for each user/host pair.

User	Session	Login Length	1st App	Last App	#Apps	#Wins	#Titles
user1-host19	3	30794	msoffice	msoffice	6	13	134
user1-host19	5	28788	msoffice	msoffice	8	15	194
user1-host19	6	19902	msoffice	msoffice	10	25	267
user1-host5	1	3472.47	explorer	explorer	3	6	34
user1-host5	2	142.98	explorer	explorer	2	3	6
user1-host5	40	21912.79	explorer	explorer	7	25	187
user19-host10	5	31432.5	msoffice	msoffice	7	8	133
user19-host10	6	16886.3	msoffice	msoffice	6	7	75
user19-host10	11	2615.55	msoffice	acrord32	6	8	45
user25-host4	2	28362.82	explorer	explorer	4	19	382
user25-host4	3	45578.82	explorer	explorer	5	16	316
user25-host4	12	6788.44	explorer	explorer	4	11	102
user4-host17	10	19445.96	wscript	explorer	8	21	452
user4-host17	30	6310.72	explorer	explorer	3	5	60
user4-host17	44	17326.21	explorer	winword	8	10	138
user7-host20	10	23163.6	outlook	outlook	5	7	51
user7-host20	11	44004.11	wscript	mapisp32	5	5	72
user7-host20	12	33125.27	wscript	outlook	5	7	166
user8-host6	1	31395.08	wscript	explorer	7	14	116
user8-host6	4	1207.84	outlook	explorer	4	4	14
user8-host6	21	134.01	cmd	explorer	3	4	13

Table 1.2. Window title usage.

#	#Sessions	Window Title
7002	425	Inbox - Microsoft Outlook
2525	411	Program Manager
2188	215	Microsoft Word
792	126	Netscape
704	156	Print
672	213	Microsoft Outlook
639	156	<<12761>> <<9227>>
592	170	<<16193>> - Message (<<16184>> <<5748>>)
555	174	<<6893>> <<13916>>
414	297	Microsoft(<<3142>>) Outlook(<<3142>>) <<7469>>
413	36	<<13683>> <<3653>> - Microsoft Internet Explorer
403	33	<<13683>> <<10676>> - Microsoft Internet Explorer
402	309	- Microsoft Outlook
401	61	Microsoft PowerPoint
198	84	http://<<1718>>.<<7267>>.<<4601>>/<<16345>>

diagonal lines are characteristic of a single window in session. The "plus" in the lower left corner shows a case of the user switching windows, then switching back. This type of behavior is seen throughout the data.

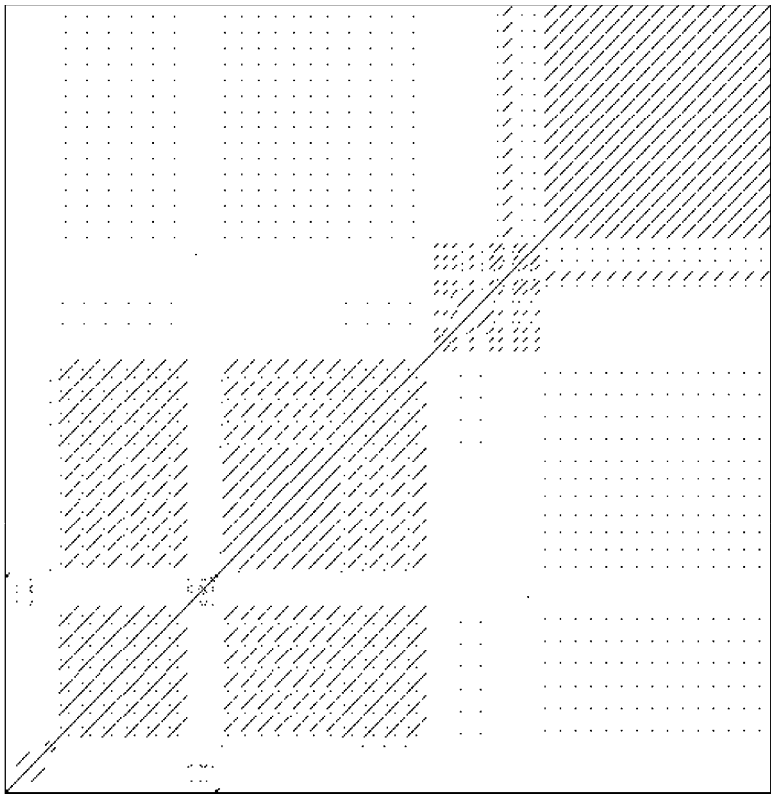

Fig. 1.10. First 500 words from a single session. The rows and columns correspond to words in the order in which they appear (with duplicates). A dot is plotted in (i, j) if the same word is in row i and column j.

Many features were extracted from the data, and several feature selection and dimensionality reduction techniques were tried. The results for these approaches were not impressive. See [6] for more discussion.

The classifiers that worked best with these data were simple intersection classifiers. For each session, the total set of window titles used (without regard to order) was collected. Then to classify a new session, the intersection of its title set with those from user sessions was computed, and the user with the largest intersection was deemed to be the user of the session. Various variations on this theme were tried, all of which performed in the mid to high 90 percent range for correct classification.

Much more needs to be done to produce a usable system. Most importantly, the approach must move from the session level to within-session calculations. Further, it is not important to classify the user as one of a list of users, but to simply state whether the user's activity matches that of the userid. It may

be straight forward to modify the intersection classifier (for example, set a threshold and if the intersection is below the threshold, raise an alarm) but it is not clear how well this will work.

We can state a few generalities about user profiling systems. Users are quite variable, and such systems tend to have an unacceptably high false alarm rate. Keystroke timings tend to be much more useful when used with a password or pass phrase than in free typing. No single technique exists which can be used reliably to authenticate users as they work.

The intersection classifier leads to interesting statistics. We can construct graphs using these intersections, each node of the graph corresponding to a session, with an edge between two nodes if their sets intersect nontrivially (or have an intersection of size at least T).

In another context (profiling the web server usage of users) [20] discusses various analyses that can be done on these graphs. This uses network data, extracting the source and destination IP addresses from the sessions. In these data there is a one-to-one correspondence between source IP address and user, since all the machines considered were single user machines.

In this case the nodes correspond to users and the sets consist of the web servers visited by the user within a period of a week. A random graph model, first described in [13] is used as the null hypothesis corresponding to random selection of servers. The model assumes a set $\mathcal{S}$ of servers from which the users draw. To define the set of servers for a given user, each server is drawn with probability p. Thus, given the observations of the sets S_i drawn by the users, we must estimate the two parameters of the model: $m = |\mathcal{S}|$ and p. These can be estimated using maximum likelihood (see also [21] for discussion of this and other types of intersection graphs). With the notation

$$\begin{aligned} k_i &= |S_i| \\ M_i &= |\bigcup_{j=1}^{i} S_j| \\ u_i &= M_i - M_{i-1}, \end{aligned}$$

the likelihood is easily shown to be

$$L = \prod_{j=1}^{n} \binom{M_{j-1}}{k_j - u_j} \binom{m - M_{j-1}}{u_j} p^{k_j} (1-p)^{m-k_j} .$$

Using data collected for several months, [20] computed the probability of any given edge, under the null hypothesis, and retained those that had a significantly large intersection (after correcting for the multiple hypotheses tested). The most common of these were retained, and the resulting graph is shown in Figure 1.11.

There are two triangles in Figure 1.11, and it turns out that the users in these correspond to physicists working on fluid dynamics problems. Users A,

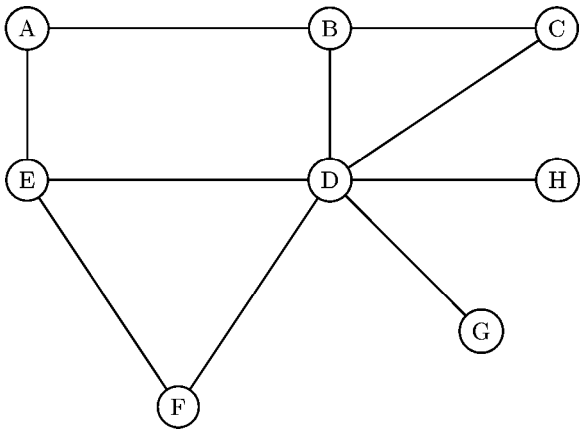

Fig. 1.11. A graph of the users with significantly large intersections. The edges for which the intersection size was statistically significant for 95% of the weeks are shown.

D and E are system administrators. Thus, there is some reason to believe that the relationships we have discovered are interesting.

The model is simplistic, perhaps overly so. It is reasonable to assume that users have different values of p, and some preliminary investigation (described in [20]) bears this out. This is an easy modification to make. Further, intuition tells us that perhaps all web servers should not have the same probabilities either. This is more problematic, since we cannot have a separate probability for each server and hope to be able to estimate them all. A reasonable compromise might be to group servers into common/rare groups or something similar.

The above discussion illustrates one of the methodologies used for anomaly detection. For determining when a service, server, or user is acting in an unusual manner, one first groups the entities using some model, then raises an alert when an entity appears to leave the group. Alternatively, one can have a single entity, for example "the network" or a given server, and build a model of the behavior of that entity under normal conditions. When the behavior deviates from these conditions by a significant amount, an alert is raised.

Other researchers have investigated the profiling of program execution, for the purpose of detecting attacks such as buffer overflows which can cause the program to act in an unusual way. See for example [11, 10, 9, 33]. Programs execute sequences of system calls, and the patterns of system calls that occur under normal conditions are used to detect abnormal execution.

1.7 Discussion

There are many areas in which computational statistics can play a part in network intrusion detection and other security arenas. We have seen a few in this chapter, including modeling denial of service attacks, visualization, the analysis of streaming data applied to network data and profiling and anomaly detection.

The biggest problems for intrusion detection systems are the false alarm rates and the detection of novel attacks. The enormous amount of data that must be processed requires that false alarm rates must be extremely low. Typical network data consists of millions of packets an hour, and system administrators generally do not have time to track down more than a few false alarms a day. Signature based systems have the advantage that they rarely false alarm (assuming the signature is properly defined), but they tend to have poor performance on novel attacks. Thus it is essential that techniques be found that detect novelty that is "bad" without alarming on novelty that is benign.

One area we have not discussed is modeling attack propagation. Early work on this can be found in [14, 15]. See also [37] for a related model. For a discussion of the slammer worm, see

`http://www.cs.berkeley.edu/~nweaver/sapphire/`

The slammer worm was interesting because the spread was self-limiting: the worm spread so fast that the available bandwidth was reduced to the point that the worm as unable to continue to spread at its initial rate. Models for these types of worms is an interesting area of study.

References

1. Edward Amoroso. *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response.* Intrusion.net Books, Sparta, New Jersey, 1999.
2. Debra Anderson, Teresa F. Lunt, Harold Javitz, Ann Tamaru, and Alfonso Valdes. Detecting unusual program behavior using the statistical component of the next-generation intrusion detection expert system (nides). Technical Report SRI-CSL-95-06, SRI International, May 1995.
3. anonymous. *Maximum Security.* Sams.net Publishing, Indianapolis, IN, 1997.
4. Rebecca Gurley Bace. *Intrusion Detection.* MacMillan Technical Publishing, Indianapolis, IN, 2000.
5. Saleh Bleha, Charles Slivinsky, and Bassam Hussien. Computer-access security systems using keystroke dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 12(12):1217–1222, 1990.
6. Kristen DeVault, Nick Tucey, and David Marchette. Analyzing process table and window title data for user identification in a windows environment. Technical Report NSWCDD/TR-03/122, Naval Surface Warfare Center, 2003.

7. J. P. Early and C. E. Brodley. Behavioral authentication of server flows. In *The 19th Annual Computer Security Applications Conference*, 2003. to appear.
8. Terry Escamilla. *Intrusion Detection: Network Security Beyond the Firewall.* John Wiley & Sons, Inc., New York, 1998.
9. Stephanie Forrest and Steven A. Hofmeyr. Immunology as information processing. In L. A. Segel and I. Cohen, editors, *Design Prinicples for the Immune System and Other Distributed Autonomous Systems*, Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, Oxford, UK, In press. Also available at www.cs.unm.edu/~forrest/ism_papers.htm.
10. Stephanie Forrest, Steven A. Hofmeyr, and Anil Somayaji. Computer immunology. *Communications of the ACM*, 40:88–96, 1997.
11. Stephanie Forrest, Alan S. Perelson, Lawrence Allen, and Rajesh Cherukuri. Self-nonself discrimination in a computer. In *1994 IEEE Symposium on Research in Security and Privacy*, 1994. Also available at www.cs.unm.edu/~forrest/isa_papers.htm.
12. Kendall Giles, David J. Marchette, and Carey E. Priebe. A backscatter characterization of denial-of-service attacks. In *Proceedings of the Joint Statistical Meetings.* 2003. to appear.
13. M. Karonski, K. Singer, and E. Scheinerman. Random intersection graphs: the subgraph problem. *Combinatorics, Probability and Computing*, 8:131–159, 1999.
14. Jeffrey O. Kephart and Steve R. White. Directed-graph epidemiological models of computer viruses. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 343–359, 1991.
15. Jeffrey O. Kephart and Steve R. White. Measuring and modeling computer virus prevalence. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 2–15, 1993.
16. Daw-Tung Lin. Computer-access authentication with neural network based keystroke identity verification. In *International Conference on Neural Networks*, pages 174–178, 1997.
17. David J. Marchette. Passive detection of denial of service attacks on the internet. In W. Chen, editor, *Statistical Methods in Computer Security.* Marcel Dekker. to appear.
18. David J. Marchette. *Computer Intrusion Detection and Network Monitoring: A Statistical Viewpoint.* Springer, New York, 2001.
19. David J. Marchette. A study of denial of service attacks on the internet. In *Proceedings of the Army Conference on Applied Statistics*, 2002. to appear.
20. David J. Marchette. Profiling users by their network activity. In *Proceedings of the Joint Statistical Meetings.* 2003. to appear.
21. David J. Marchette. *Random Graphs for Statistical Pattern Recognition.* John Wiley & Sons, New York, 2004.
22. Roy A. Maxion. Masquerade detection using enriched command lines. In *International conference on dependable systems and networks(DNS-03).* IEEE Computer Society Press, 2003.
23. Roy A. Maxion and Tahlia N. Townsend. Masquerade detection using truncated command lines. In *International conference on dependable systems and networks(DNS-02).* IEEE Computer Society Press, 2002.
24. David Moore, Geoffrey M. Voelker, and Stefan Savage. Inferring Internet denial-of-service activity. Available on the web at www.usenix.org/publications/library/proceedings/sec01/moore.html, 2001. USENIX Security '01.

25. Stephen Northcutt, Judy Novak, and Donald McLaclan. *Network Intrusion Detection. An Analyst's Handbook.* New Riders, Indianapolis, IN, 2001.
26. M. S. Obaidat and Balqies Sadoun. Verification of computer users using keystroke dynamics. *IEEE Transactions on Systems, Man, and Cybernetics*, 27(2):261–269, 1997.
27. Carey E. Priebe. Adaptive mixture density estimation. *Journal of the American Statistical Association*, 89:796–806, 1994.
28. Paul E. Proctor. *The Practical Intrusion Detection Handbook.* Prentice-Hall, Englewood Cliffs, NJ, 2001.
29. John A. Robinson, Vicky M. Liang, J. A. Michael Chambers, and Christine L. MacKenzie. Computer user verification using login string keystroke dynamics. *IEEE Transactions on Systems, Man, and Cybernetics*, 28(2):236–241, 1998.
30. Matthias Schonlau, William DuMouchel, Wen-Hua Ju, Alan F. Karr, Martin Theus, and Yehuda Vardi. Computer intrusion: Detecting masquerades. *Statistical Science*, 16:58–74, 2001.
31. Dawn X. Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proceedings of the 10th USENIX Security Symposium*, 2001. http://www.usenix.org/publications/library/proceedings/sec01/song.html.
32. W. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols.* Addison-Wesley, Reading, MA, 1994.
33. Kymie M. C. Tan and Roy A. Maxion. "why 6?" defining the operational limits of stide, an anomaly-based intrusion detector. In *IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, 2002.
34. E. J. Wegman and H. I. Davies. Remarks on some recursive estimators of a probability density. *Annals of Statistics*, 7:316–327, 1979.
35. E. J. Wegman and A. Dorfman. Visualizing cereal world. Technical Report TR 178, George Mason University, Center for Computational Statistics, October 2001.
36. E. J. Wegman and D. J. Marchette. On some techniques for streaming data: a case study of Internet packet headers. *JCGS*, 2004. to appear.
37. J. C. Wierman and D. J. Marchette. Modeling computer virus prevalence with a susceptible-infected-susceptible model with reintroduction. *Computational Statistics and Data Analysis*, 2004. to appear.
38. A. F. X. Wilhelm, E. J. Wegman, and J. Symanzik. Visual clustering and classification: The oronsay particle size data set revisited. *Computational Statistics*, pages 109–146, 1999.
39. H. Yamato. Sequential estimation of a continuous probability density function and the mode. *Bulletin of Mathematical Statistics*, 14:1–12, 1971.