Holding Intruders Accountable on the Internet

Stuart Staniford-Chen
L. Todd Heberlein

Department of Computer Science
University of California at Davis
Davis, CA 95616

Abstract

This paper addresses the problem of tracing intruders who obscure their identity by logging through a chain of multiple machines. After discussing previous approaches to this problem, we introduce thumbprints which are short summaries of the content of a connection. These can be compared to determine whether two connections contain the same text and are therefore likely to be part of the same connection chain. We enumerate the properties a thumbprint needs to have to work in practice, and then define a class of local *thumbprints which have the desired properties. A methodology from multivariate statistics called principal component analysis is used to infer the best choice of thumbprinting parameters from data. Currently our thumbprints require 24 bytes per minute per connection. We develop an algorithm to compare these thumbprints which allows for the possibility that data may leak from one time-interval to the next. We present experimental data showing that our scheme works on a local area network.*

1 Motivation

Networked computer systems are under attack, and the number of attacks is growing exponentially. In 1990, 252 incidents were reported to the Computer Emergency Response Team (CERT). In just the first six months of 1991, that number had grown to 1172. In addition to the growth in the number of reported incidents, the number of systems involved per incident is growing - one recent incident involved 65,000 systems. [1]

Furthermore, it seems probable that most incidents are not reported. For example, ASSIST, the Department of Defense incident response team, recently evaluated the security level of one of their sites by launching automated attacks against it continuously for two months. Only one person reported suspicious activity. In a second example, a particularly security conscious DoD site detected 69 attacks in 1992. After installation of an intrusion detection tool, they detected 4100 attacks in just the first quarter of 1993. [2]

Why are so many attacks occurring? Studies reveal computer attacks have similarities with many other crimes: perpetrators are motivated by many things, including greed, revenge, and peer pressure. [3, 4] As the Internet continues to grow, and as more and more commercial activity takes place over it, it would seem likely that the problem will continue to worsen.

Studies also suggest that many intruders are deterred by the perceived risks involved. One of the intruder's greatest fears is losing his or her anonymity. [4]

Unfortunately, attackers can take advantage of the architecture of the Internet to hide their point of origin, thus preserving their anonymity. Since many hosts are insecure, intruders assemble a collection of accounts on hosts around the world that they have broken into. When conducting an attack, they log-in through a series of such hosts before assaulting the target. Since the machines in question are in different administrative domains, with personnel who may not know or trust one another in advance, and perhaps do not even have the same legal system, this makes it extraordinarily difficult to trace back the chain of activity to its source. Clifford Stoll's experience is a good example. [5]

Because of these problems, most incident response teams such as CERT make little or no effort to find the intruder. The result: an intruder still has all the potential rewards with almost no risks.

The goal of our research in this area is to develop means by which intruders can be traced efficiently. This paper details our work so far. An important restriction which we impose on our approaches is that they can be retrofitted to the existing Internet. Thus, we do not consider methods which would require changes to the low-level network protocols, or require a trusted computing base.

This means that the methods we produce will certainly be imperfect. It is impossible to produce a foolproof method to give correct security information when the scheme must be implemented on hosts and networks which have insecure operating systems and insecure protocols. Nonetheless, we feel that some tracing ability, however imperfect, is better than none at all.

This paper provides an introduction to the problem, a discussion of what possible solutions could look like, a description of previous efforts, and a discussion of *thumbprinting*. This is a technology we have been developing to compare connections and thereby trace.

1081-6011/95 $04.00 © 1995 IEEE

A thumbprint is similar to a checksum. One is stored for each interval of each connection, and later used to verify whether or not two connections had the same content. We describe our algorithms and present some early experimental results.

2 Approaches.

We first define terminology. When a person (or a program) logs into one computer, from there logs into another, and another, via network connections or modems, we refer to that as an *extended connection*, or a *connection chain*. The task of a *tracing* mechanism is, given some part of the chain, to identify the beginning of it.

In general we believe tracing mechanisms fall into two classes. In the first class are methods which attempt to keep track of all individuals on the network and account all activity to network wide user-ids. The DIDS system developed at UC Davis is an example of a system which did this for a single local area network. [6] We believe that, while this approach may be a good idea for a wide area network which is under central administration, it is not feasible on the Internet with its diversity of administrations, operating systems, and security policies.

The second class contains *reactive* tracing mechanisms. In this case, no global accounting of users is attempted until a problem arises. Then the activity is traced back to its source.

One class of reactive methods is host based solutions. These involve one tracing system per network host. Each such system is capable of establishing where a chain that crosses *it* goes next, and tracing is accomplished by the hosts communicating in some way to establish the whole extended connection.

A system along these lines is CIS (Caller Identification System)[7]. This system is applied in an attempt to authenticate users about to log into a machine at the end of an extended connection. Each machine along the chain keeps a record of what the chain looks like as far as it. When the user attempts to log into the nth machine from the $n-1$th machine, the nth machine asks it's predecessor for information about the chain so far. The nth machine then queries machines 1 through $n-2$ to check that their impression of the connection chain agrees with that of machine $n-1$. Only if all machines along the chain agree (and machine 1 is acceptable to machine n) does the login proceed.

This recursive checking of the chain eliminates some, but not all, of the obvious attacks on this kind of scheme. It seems that it would be likely to induce excessive delays in some cases however.

A different approach was actually used by the United States Air Force to track an intruder and arrest him. [8] This technique, confusingly called Caller ID also, is controversial and required special permission from the Department of Justice, so it is probably not a technique for general use.

Caller ID is based on the belief that if an intruder hops through intermediate systems prior to making an attack, there is a high probability that these systems have known vulnerabilities which the intruder used to access them. For example, if the intruder hops from $H_0 \rightarrow H_1 \rightarrow \ldots \rightarrow H_n$, where H_n is the target, H_1 through H_{n-1} contain at least one vulnerability allowing access by an outsider. The Air Force, having knowledge of the same attack methods that their intruder did, simply reversed the attack chain - breaking into H_{n-1}, examining the system tables to see from where the intruder was coming, breaking into H_{n-2}, and so on. Eventually they identified the original point of entry of the perpetrator.

The drawbacks of this tracing technique include the possibility that one cannot break into one of the intermediate system, one must perform the tracing while the intruder is active, and one runs the risk of accidentally damaging intermediate systems. For many practitioners, the legal situation is likely to cause severe problems also.

The difficulty with all such host-based tracing systems is that, when an extended connection crosses a host which is not running the system, accountability is altogether lost at that point. This severely limits their usefulness as a general purpose tracing mechanism on the Internet. Since many hosts on the Internet are insecure, the integrity of the tracing system on those hosts cannot be relied upon. Some intruders almost reflexively install Trojan horse versions of important system binaries on hosts they penetrate. Even if most hosts could be secured, the intruder community could easily maintain a set of machines to launder connections, just as they maintain anonymous remailers.

Such schemes may nonetheless be useful in single administrative domains where measures are taken to guarantee the integrity of the system.

The approach we discuss in the remainder of the paper is *thumbprinting*. This relies on the fact that the content of an extended connection is invariant at all points of the chain (once protocol details are abstracted out). Thus if the network tracing system can compute summaries (thumbprints) of the content of each connection, these summaries can later be compared to establish whether two connections have the same content. The technical feasibility of this idea will be discussed later in the paper. The main drawback to this approach is that it is still vulnerable to countermeasures. Firstly, the system must still be protected from trojaning, though this is perhaps easier to do since there are fewer stations involved and they can be special purpose. The second weakness is that disguising the content of the extended connection (such as encrypting it differently on each link of the chain) can circumvent the technology.

By far the most compelling advantage of the thumbprinting approach is that it could be useful even when only parts of the Internet use it. We discuss this point at greater length in a later section. We note that since the thumbprints are very small, it is usually impossible to deduce details of the connection content from them. This limits their impact on privacy to traffic analysis.

3 Thumbprinting

3.1 Idea

The idea of a thumbprint, which was originally proposed in [9], is a small quantity which effectively summarizes a certain section of a connection. The ideal is a function of the connection which uniquely distinguishes a given connection from all other unrelated connections, but has the same value over two connections which are related by being links in the same connection chain. This would allow a tracing system to later compare various connections to find all pieces of a chain. If all components of the system routinely store thumbprints, then in the event of an intrusion being detected, it is possible to trace the connection back by comparing thumbprints from different hosts or networks.

We now turn to discussing how thumbprinting could be implemented. Initially, we have concentrated our attention on TCP connections, and specifically interactive connections over the telnet or rlogin protocols. We believe, however, that many of our results will extend in principle to machine driven data transfers and (in some cases) to connectionless protocols such as UDP. (Though the performance requirements for the thumbprinting system increase correspondingly). Also, we currently take the view that lengthy connections should be broken up into time intervals, and each interval separately thumbprinted. Interval size is typically a minute.

3.2 Design Constraints

A good thumbprint should have the following properties.

It should require as little space as possible to minimize storage needs for logs of thumbprints.

It should be sensitive; the probability that two unrelated pieces of connection will be close together in thumbprint space should be as small as possible. Of course, if two unrelated pieces of connection happen to have the same content then no thumbprint will distinguish them. The most common case of this is idle connections.

It should be robust, *i.e.* it should change as little as possible when the connection gets distorted by the kinds of errors that are likely in practice.

Ideally, thumbprints should be additive. This means that successive ones can be combined into a thumbprint for a longer interval. This allows that where successive thumbprints do not provide a clear comparison, they can be combined to produce a better signal. It also allows thumbprints of intervals of different but congruent lengths to be compared.

Finally, it is essential that creating the thumbprints not place an excessive load on the network components which do the work. It is useful but less important if they are cheap to compare.

3.3 Sources of Error

We have identified the following sources of error. Any scheme must cope with these.

1) Clock skew - thumbprints on different hosts may not always start at quite the same time, and may not end at quite the same time either. This causes errors in comparing them since characters that in one place may be in the nth minute of a connection may be in the $n+1$th minute elsewhere. It is essential that synchronization errors be much smaller than the thumbprinting interval.

2) Propagation delays - thumbprints may contain slightly different data in different places because the connections they are measuring are delayed by propagation times. This has a very similar effect to clock skew in moving some characters from one thumbprinting interval to the next. In our experience the worst problems are created by overloaded hosts, rather than by the network itself. Badly overloaded hosts may pause for seconds or tens of seconds before transmitting data they have received.

3) Loss of characters. Since thumbprinting is based on passive monitoring of connections rather than being a party to them, the system cannot have access to the error and flow control features of the transport protocol (TCP). Thus it might lose some characters (due to a buffer overflowing say) and not be able to recover them. We have found that this is a problem in practice.

4) Packetization variation. Thumbprinting at a low level in the protocol stack is made difficult by the fact that packetization, timing of packet transmission, *etc.* are not invariant at different points in the connection chain.

For these reasons, we decided to rely solely on the content of the connection after reconstruction up through at least the transport layer. In the future we hope to study whether this is really necessary, or whether it is better to rely on the error tolerance of the thumbprinting method introduced below to cope with retransmissions etc.

An obvious contender for thumbprints is a checksum such as the CRC. These are very small, they are very sensitive, they are cheap to compute. The big problem is that they are not robust at all - any error in the data used to make the checksum is likely to completely change the value of it. They are also not additive. Message digest algorithms have the same drawbacks.

Other possibilities we considered and ruled out due to space considerations were compression techniques, and signature retrieval techniques (as used in the search of large free-text databases). [10]

3.4 Applicability

In the short term, we see several applications in which this kind of technology could be deployed almost immediately.

1) In the context of distributed intrusion detection systems such as DIDS[6], thumbprinting could allow the system to relate activity which went outside the domain but then re-entered. This might be important when an inside attacker was seeking to disguise himself as an outsider. Indeed we understand that Trident Data Systems in conjunction with the Air Force Office of Information Warfare is presently incorporating these ideas into DIDS.

2) Thumbprinting systems could be placed at the places where a network for some site touched other networks. This would allow the administrators of that

site to determine whenever their systems were being used as a pass-through site. (Bob Pallasek of Lawrence Livermore Laboratories suggested this application to us).

3) Sites which were logically a single site, but physically several networks, could use this means to correlate activity between the different sites.

4) Law enforcement in pursuit of particular intruders could use this technology at a variety of places which were under suspicion as the likely source of an intruder.

In the longer term, this technology could be a useful component in a general internet tracing system (akin to the trap-and-trace facility provided by the phone networks). No such facility is presently planned. However, as computer networks become increasingly used for commerce, it may become necessary.

4 Local thumbprints

The thumbprinting technology we settled on we refer to as local thumbprints. Suppose the sequence of characters to thumbprint is $a_1, a_2, \ldots, a_n$, and further suppose that we have a function ϕ which takes a character as argument and returns a short vector of real numbers; $\phi : \mathcal{A} \rightarrow \Re^K$. Then one possible kind of thumbprint would be

$$T = \frac{1}{n} \sum_{i=1}^{n} \phi(a_i) \qquad (1)$$

T is a vector of short fixed length K. This is a local thumbprint in that it only depends locally on the character stream. The advantages of this kind of scheme are as follows. Robustness is good, since if we lose a few characters, only those terms in the sum are affected. Additivity is obviously satisfied in that the thumbprint for a combination of two character sequences is the sum of the thumbprints for the individual sequences. The thumbprint is small since it's just a few real numbers (in practice, some quantization of them). It's cheap to compute since the function ϕ can be stored in a lookup table. The remaining question is one of sensitivity - can such quantities effectively distinguish different connections? We will address this question empirically later in this paper.

In essence, equation (1) mandates studying linear combinations of the frequencies with which each character occurs in the particular interval of the particular connection being thumbprinted.

A number of variations on this scheme are possible. For example, give a function $\psi(a, b)$, we could define a digram thumbprint at some separation k by

$$T_\psi = \frac{1}{n-k} \sum_{i=1}^{n-k} \psi(a_i, a_{i+k}) \qquad (2)$$

More complex schemes based on trigrams or higher-order combinations are also possible. It might appear that such schemes would be more sensitive than the single character scheme because they capture some information about the order of the characters in (a_i). Ordering information is lost in the single character scheme. We conducted some preliminary experiments which suggested that this makes little difference in practice and so we have focussed on single character schemes. We hope to return to this point for more careful experimentation in the near future.

4.1 Overview of Experiments

In order to test our ideas about thumbprinting in a realistic setting, we developed C++ code to thumbprint actual network traffic. This code presently runs on a Sun 4/280 computer on one of our departmental ethernet LANs. The code uses the network interface in a promiscuous mode (through the */dev/nit* device provided in SunOS). The software analyzes each packet and associates it with the particular pair of machines and ports it is traveling between. It reconstructs the data flowing on each such connection up through the transport layer (TCP). It divides that data up into consecutive minutes, and saves the frequencies with which each character occurs in that minute for that connection. At present we restrict our attention to rlogin and telnet connections (as determined by the internet port number used). We also consolidate the data flowing in both directions into a single set of character frequencies. It would interesting to pursue whether the technique could be made more sensitive by separating out the data according to which direction it flowed in.

We find that, although the LAN segment is fairly busy because it houses one of our department's main mail and file servers, the thumbprinting program typically uses no more than a few percent of the processor time on the machine it is running on.

Two points should be mentioned here. Firstly, we mask all characters down to 7 bits since we have found eight bit characters to be comparatively rare in the interactive connections on our network and it is convenient in the rest of the analysis to work with only 128 characters rather than 256. Secondly, we have found that ASCII character 24 plays a peculiar role in our data. This character is used by the telnet protocol as part of its negotiation over which terminal type is in use[11]. Normally, this character is very infrequent. However, in a very few of our connections, massive numbers of these appear (tens or hundreds of thousands per minute). We do not presently understand the cause of this, though we suspect an implementation bug in some version of telnet. The resulting variability in the frequency of this character means that it receives significant weight in our analysis which we think is undesirable. Therefore we set the frequency of this character to zero, regardless of its actual value.

A typical experimental protocol is as follows. While the thumbprinting software is running, we execute a script which sets up an extended connection across several machines and then causes data to flow back and forth across the connection in a way that is designed to simulate a human rapidly issuing system commands and receiving responses. The data is drawn randomly from a previously saved file of activity. The script types a line of characters at a rate similar to human typing, then issues a burst of numerous lines

1	2	3	4	5	6
38.2	6741.1	6975.7	2587.2	3446.5	2451.2
11.8	13505.5	92.8	2569.7	3388.7	2446.0

Table 1: Thumbprints in concept experiment

of data all at once. It then waits a random amount of time (typically a number of seconds) before repeating the cycle.

We arrange for the extended connection to cross the LAN segment we are monitoring several times. This allows us to compare the thumbprints at those points in the chain. The only difference between this set up and a more realistic one where two geographically separated pieces of the extended connection were being compared is that we do not have to arrange for synchronization between the monitors.

The reason we injected our own simulated connections into the network traffic was to make it easy to find them again when we came to analyze the data, and to allow us to control variables such as how many machines the extended connection crossed before returning to our monitored LAN.

In total we took about a week's worth of data. Some of this represented our injected connections, but much of it was unrelated activity by other users.

4.2 Concept Experiments

To give the reader some feel for the kind of data we analyze, we present the following table (Table 1). This was one of our early proof of concept experiments, and it differs from our current setup in that it is based on total counts of characters, not frequencies, and it used a single (*i.e.* $K = 1$) randomly chosen thumbprint function. However, it illustrates several important points.

The top row labels time in minutes. The other two rows are the thumbprints obtained in two different places on an extended connection chain during each of those minutes. Notice that in minutes 4 through 6, the thumbprints agree quite well, but not exactly. Errors of one or two percent like this are quite common due to missed packets or synchronization errors. The thumbprints in minute 1 do not agree well. This too is very common at the beginning of a connection chain. As each successive link in the chain is set up, its thumbprint is initially based on no data, while thumbprints of earlier links of the chain are based on some text (*e.g.* the command to log into the next machine). The most interesting point is that minutes 2 and 3 also match very poorly. However, if these are added together, then the combined thumbprint for the top row is 13716.8, while that in the bottom row is 13598.3. This represents quite good agreement. We inspected the datastreams here and determined that the cause was several large packets of characters which in the lower row fell in minute 1, but were delayed due to an overloaded host so that by the time they were recorded in the upper row of the table, they fell into minute 2. We believe this kind of scenario is not uncommon, and it illustrates the importance of having thumbprints which are additive and tolerant of noise.

4.3 Which thumbprint function?

Given that we are using single character local thumbprints, the question still arises as to which such thumbprint is best. Given that the vector of character frequencies for a particular period of some connection is $\vec{f} = (f_1, f_2, \ldots, f_L)$, the thumbprint can be written as a linear combination

$$T_j = \sum_{a=1}^{L} \phi_j(a) f_a \tag{3}$$

Thus we condense the vector of L character counts into a vector of K thumbprint components (indexed by j). The question that must be addressed is which linear combinations of the f_i should be used?

Happily, statisticians have developed a machinery for answering this kind of question known as principal component analysis. Since this technique is well discussed in textbooks [12, 13], and we do not have the space for a full treatment, we will only describe it briefly.

The aim of principal component analysis is to take a series of vectors and find a set of linear combinations of the components which explains the maximal proportion of the variance of the vectors. This is done by computing the covariance matrix of the vectors, and then finding its eigenvalues and eigenvectors. The eigenvalues are sorted by size. Then the eigenvector associated with the largest eigenvalue represents the linear combination of the data which has the most variance, and the eigenvalue is that variance. The eigenvector associated with the second largest eigenvalue is the linear combination which explains the maximal amount of variance after that associated with the first eigenvalue has been removed. This continues in the same pattern – there are K principal components, each successive one accounting for less and less of the variance, but accounting for all of it between them.

This exactly answers our need. We wish to use the linear combination with the greatest variance, since character frequencies, or combinations of frequencies, which vary very little are unlikely to be useful in distinguishing amongst different connections, while very variable frequencies are the most likely to be different in unrelated connections.

We obtained from our data sets a total of 28677 distinct connection minutes (excluding ones we had injected for experimental purposes). For each of these we formed the frequency vector (which has 128 components). We then applied principal component analysis to these vectors. The largest eigenvalues are shown in Figure 1.

Ideally, we could look at this picture and there would be some obvious place to stop - the first N principal components would explain almost all the variance, and we could ignore the ones after that. This is not the case; the graph becomes very flat after the first few components. Rather arbitrarily, we decided to use the first $K = 6$ components in this study. We hope

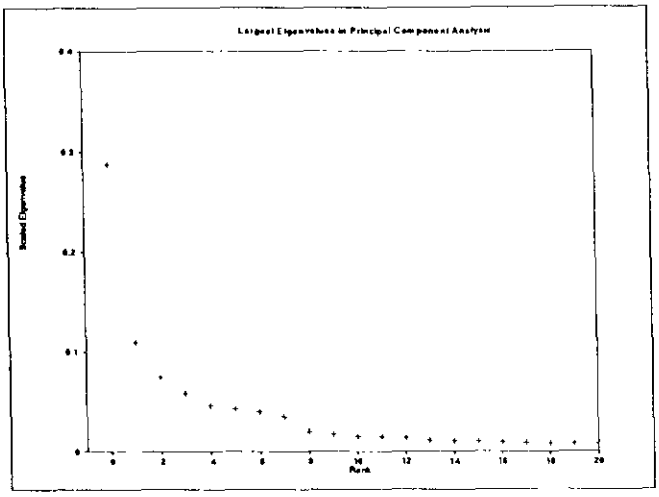


Figure 1: The largest 20 eigenvalues of the covariance matrix of our samples of character frequency in network connections.

to study more carefully the impact of this decision in future work.

The corresponding coefficients are shown as a function of ASCII character in Figure 2 for the first three components.

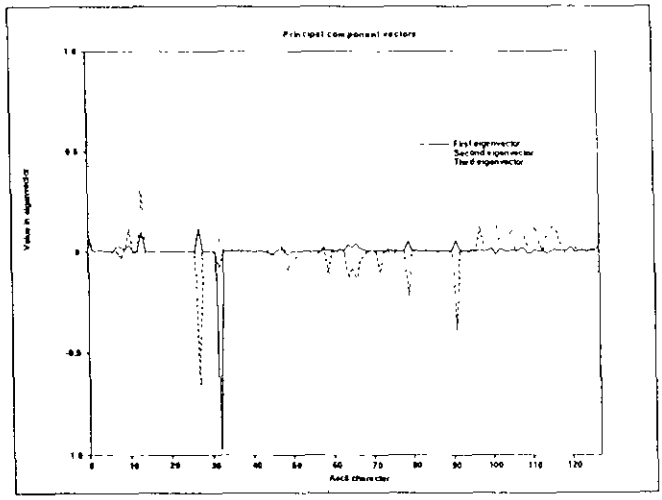


Figure 2: The first three principal components. The value is graphed for each ASCII character.

The first vector (which explains 28% of the variance in our character frequencies) is clearly measuring how many spaces (ASCII 32) there are in the traffic versus other characters. Succeeding vectors make little use of the space frequency. It is striking that the statistical procedure picks very different things to emphasize than humans might expect. Our expectation was that most of the meaningful information was in the relative frequency of letters of the alphabet. However, it seems in fact to be more useful to work with punctuation characters, terminal codes, and white space. Letters of the alphabet are mainly treated as a block (the lower case letters occur from ASCII 97 to ASCII 122).

We used these vectors as our choice of $\phi(a)$ in the remainder of our work.

4.4 Comparison Algorithm

Having created thumbprints of all the connection intervals we need a procedure to compare them. This has to distinguish when two connections are the same, and when they are different. It is complicated firstly by the need to cope with displacements of some characters across interval boundaries, and secondly by the existence of noise in the data due to dropped packets. We developed a procedure which seems to handle both of these difficulties well.

Since the noise distribution is very difficult to characterize (because missed packets, by definition, are missed), we work with the known distribution of unrelated thumbprints and attempted to establish that related ones are atypical if they are considered to be drawn from a distribution of unrelated ones.

Specifically, we start with the K thumbprint components $T_k(C,t)$ for a particular connection C and time interval t. To compare this with some other set of thumbprints $T_k(C',t)$, we form the quantity

$$\delta_t(C,C') = \log\left(\prod_{k=1}^{K} |T_k(C',t) - T_k(C,t)|\right) \qquad (4)$$

The idea is that the product of differences between the $T_k(C',t)$ and the $T_k(C,t)$ will be much smaller if C and C' are related than if they are not. This will make $\delta_t(C,C')$ larger in magnitude than expected.

However, if successive thumbprints match over time, that further increases our confidence that the connections are the same. We wish to incorporate this fact into our procedure.

We can consider the $T_k(C,t)$ to be drawn from some probability distribution $P_k(T)$ of thumbprints of all intervals of all rlogin and telnet connections on the Internet. This in turn induces a probability distribution for the δ_t, viz:

$$\delta_t(C,C') \sim P(\delta) \qquad (5)$$

under the assumption that C and C' are independent.

Of course, we cannot know this distribution $P(\delta)$, but we approximate it by the following procedure. We take our list of connection-minutes observed in our data (excluding injections) and randomly draw two of them. Then we compute δ from them using the above procedure as if they had actually been taken at the same time. Doing this many times gives us a histogram $P'(\delta)$. Ours is based on a Monte Carlo sampling of 10^7 differences. We take this as an approximation to the true P. Now given this, we define the statistic $p_t(\delta_t)$ by

$$p_t(\delta_t) = \int_0^{\delta_t} P'(x)dx \qquad (6)$$

Intuitively, p_t is (an approximation to) the probability of observing a δ as small as δ_t or smaller by comparing independent connection intervals. We refer to this as the significance of the comparison at time t. A very small p_t implies a significant result.

Now, to agglomerate a comparison over time the most naive procedure is to take the product of the p_t for all the minutes in which we can compare C and C'.

$$p_{\text{naive}}(C, C') = \prod_{t=1}^{s} p_t(C, C') \tag{7}$$

where we assume that t runs from 1 to s. It is natural to think of $p_{\text{naive}}(C, C')$ as the probability that all the thumbprints would be as close as they are if C and C' were unrelated connections. This is not correct for several reasons.

Firstly, in taking the product of the probabilities for successive minutes, we are assuming independence of successive thumbprint comparisons over time, which is unlikely to be exactly the case even for unrelated connections. It is not feasible presently for us to quantitatively assess the lack of independence, and so our approach is to make the approximation that successive minutes of unrelated connections are independent, and then study how badly this fails when we apply the whole comparison analysis to control data. We find that although the assumption is not perfect, nonetheless we are well able to distinguish control data from connections which really should match.

More importantly, under the null hypothesis that C and C' are independently and randomly chosen connections, the p_t are random variables drawn from a uniform distribution on $[0, 1]$. Thus when we take their product, the result is drawn from the distribution of the product of s $U(0, 1)$ distributions. This distribution can be calculated analytically (see Appendix A), and the result is

$$U^s(x) = \frac{(-\log x)^{s-1}}{(s-1)!} \tag{8}$$

Thus we define

$$p_{\text{basic}} = \int_0^{p_{\text{naive}}} U^s(x)dx \tag{9}$$

So p_{basic} is the probability of p_{naive} being as small as the observed value or smaller, under the hypotheses of unrelated connections and independence over time.

This statistic still takes no account of the need in some cases to add together successive thumbprints because of leakage of characters from one interval into a neighboring one. Our algorithm is as follows. We compute p_t for each t. If $p_t < \tau$, where the tolerance τ is some small value (10^{-3} in this study), then we immediately count this value of t as a good match. After we have done this for all t, we go back through the data and look for situations in which consecutive values of t do not constitute good matches. We then combine those thumbprints in pairs, and produce a combined value of δ as

$$\delta_t^{(2)}(C, C') = \log\Big(\prod_{k=1}^{K} |(T_k(C', t) \oplus T_k(C', t+1)) - (T_k(C, t) \oplus T_k(C, t+1))|\Big) \tag{10}$$

where the $\oplus$ operation just represents the combination of the thumbprints weighted by the number of characters in each minute.

$$T(C, t) \oplus T(C, t+1) = \frac{n_t T(C, t) + n_{t+1} T(C, t+1)}{n_t + n_{t+1}} \tag{11}$$

Now, the $\delta_t^{(2)}$s are not drawn from the same distribution as the δ_ts. However, we can again produce an estimate of this distribution by Monte Carlo sampling of summed differences of independent thumbprints drawn from our data. This allows us, in a similar way to before, to compute $p_t^{(2)}$ as the percentile point of $\delta_t^{(2)}$ in its distribution. Thus $p_t^{(2)}$ is the significance of the comparison of C and C' for the combined intervals t and $t+1$.

The question that then arises is; is the comparison of the combined intervals t and $t+1$ more significant than the comparison of the two intervals taken separately? To answer this, the natural thing to do is to compare $p_t^{(2)}$, with

$$p_t^{(1,2)} = \int_0^{p_t p_{t+1}} U^2(x)dx \tag{12}$$

which is our measure of how significant the comparison is in the two intervals taken separately.

We then adopt either $p_t^{(2)}$ or $p_t^{(1,2)}$, whichever is the smallest. It is important to note that both of these numbers are drawn from $U(0, 1)$ under the hypotheses of unrelated connections and independence in time. The fact that they have the same distribution is the justification for comparing them. We do this wherever it is advantageous and the individual p_ts failed to meet the tolerance. Suppose we perform this comparison r times. We then have $s - 2r$ numbers p. Some of these may be p_ts, some $p_t^{(2)}$s, and some $p_t^{(1,2)}$s. We take the product of all of these and compute

$$p_{\text{pair}} = \int_0^{\prod p} U^{s-2r}(x)dx \tag{13}$$

This we then take as the significance of the full comparison of C and C' over the s time intervals. For convenience we look at

$$l_{\text{pair}} = -\log(p_{\text{pair}}) \tag{14}$$

This number is always positive, large when the comparison is very significant, and small when it is not.

We note again that since several approximations are made in this development, this can only be considered the logarithm of a probability in a rather approximate sense.

We note that it would be straightforward to extend these ideas to adding more than two consecutive thumbprints together. We have not yet carried out this analysis.

We make a last point; in practice when comparing thumbprints of related connections, there is a significant chance that the thumbprints will be *exactly* the same. This causes the analysis above to produce an infinite answer. Alternatively, it is possible for δ_t to be so small that it was smaller than any of the values used in constructing the Monte Carlo approximate histogram δ_t. This again gives an infinite answer. In both of these cases, we refer to this as a dead hit at time t. Thus the analysis produces two values: the number of dead hits, and the significance l_{pair} of the observations which were not dead hits.

Generally, any dead hits are very strong grounds for suspecting that the two connections have identical content. l_{pair} comes into play when the data are too noisy to allow of this.

4.5 Tests of Thumbprinting

We begin by describing our control data-set. We scanned through all the connections we had recorded thumbprints for. We excluded any which were deliberately created by us as experiments, and any which had less than five minutes worth of data in. We then paired the connections randomly. Any pairs which involved the same set of machines, or which were closer than an hour together in time were excluded in an attempt to reduce the chance of accidentally comparing connections which had the same content. We used a total of 40000 pairings in the control. For each of these, we applied our comparison methodology to four minutes. (We excluded the first minute of the connections). We observed exactly one dead hit in one minute of these comparisons. We checked and found that the character totals were identical, and some detective work with these suggests that this was the last minute of two unrelated connections which happened in both cases to contain little more than a prompt, and the word 'logout'. This kind of thing is bound to happen occasionally.

The histogram of the obtained values of l_{pair} is shown in Figure 3 as the dotted line. The solid line is the curve that would apply if successive values of p_t were independent so that p_{pair} was distributed uniformly on $[0, 1]$. Clearly (as expected) this assumption is violated and thus comparisons between unrelated connections tend to be more significant than this assumption would allow. However, it is not so grossly wrong as to make us abandon the natural comparison suggested in and immediately after equation (12). We also speculate that the extreme right tail of the control histogram contains comparisons between connections which chance to have some related data (a risk when all data is taken on the same network).

We applied the same comparison procedure to four

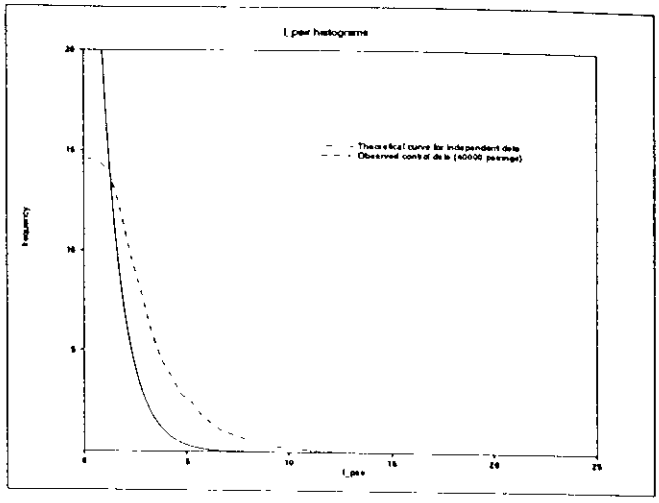


Figure 3: Histogram of l_{pair} for control data, together with the theoretical distribution assuming independence in time.

Run	N	4 hits	3 hits	2 hits	1 hit	0 hits
I & II	302	60	17	14	7	2
III	54	20	26	30	19	6
IV	28	4	32	29	32	4

Table 2: Number of trials and percentage of each number of hits for the experimental runs described in the text.

sets of injected data. In Run I, our extended connection began on *toadflax*, went to *k2*, and then went back to *toadflax* where both *toadflax* and *k2* are within our department. In Run II, the extended connection went from *k2* to *toadflax* to *k2* and back to *toadflax*. This gave three legs of the extended connection that could be compared. Thus there are two independent sets of thumbprint differences for each injected connection. As for the control data, we looked at four minutes worth of data in each case, after dropping the first minute (which usually gives an unreliable comparison).

These two runs gave similar results, so we combined them. There were a total of 302 comparisons. The percentage of trials with various numbers of hits is given in Table 2. In all, 98.3% of the comparisons gave at least one dead hit. Five comparisons were sufficiently disturbed by noise as to give no dead hits. The values for l_{pair} in these cases were 36.49, 37.46, 37.76, 39.70, and 42.34. Comparison of these values with the control histogram in Figure 3 makes it clear that they are very large, indicating that the method clearly can identify these connections despite the noise.

In our next experiments, we tested the method on extended connections over long-haul networks. These are harsh conditions, (but ones that are perhaps typi-

cal of intrusions). The delays between typing a character and seeing the echo were typically several seconds over these chains. In Run III, the connection chain went

toadflax → k2 → helvellyn
→ alps.cc.gatech.edu → k2 → toadflax

Here, *alps.cc.gatech.edu* is in Georgia while the rest of the hosts are in Davis. We compared the chain as it left and re-entered *toadflax*. All but three of the 54 comparisons gave some dead hits. On those that did not, the values of l_{pair} were 22.74, 41.29, and 44.31. Again, these numbers are very far out into the tail of the control histogram, although the smallest of these does cross with the most significant of the control comparisons.

The schema in Run IV was

toadflax → k2 → helvellyn → po.csc.liv.ac.uk
→ alps.cc.gatech.edu → k2 → toadflax

po.csc.liv.ac.uk is in Liverpool, England. Only one of the 28 experimental connections gave 0 dead hits, and it had an l_{pair} value of 28.09. Thus, even in this long chain, we can successfully match up the endpoints of the connection in all cases.

We also studied whether we could reliably pick our injected connections out from our control connections. For each pair of samples from an injected connection, we chose one of the pair and compared it to all the connections in our control set. We then assessed whether it was more similar to it's actual partner than to any of the unrelated data.

To compare the value of two matches, it is convenient to have a method to combine the number of dead-hits with the significance level where there is not a dead-hit. Thus, we must give a significance level to a dead-hit. To do this, we looked at the significance level of all of our comparisons on an individual, minute-by-minute basis. We found that the highest significance level achieved for a minute of comparison which was not a dead-hit to be 13.82. We therefore set the significance of a dead hit at 14. We then combined all significances into a single number which incorporated the dead-hits.

For each of our injected connections, we then computed its total significance in this manner, and the total significance of comparing it with all the unrelated control connections. We formed the ratio $\mathcal{R}$ between the total significance of the correctly matched comparison, and the best of the unrelated comparisons. Thus we get one value of $\mathcal{R}$ for each injected connection. If things are working correctly $\mathcal{R}$ should be more than one. Preferably quite a bit more than 1.

Table 3 tells the story. For each group of runs, we present the median value of $\mathcal{R}$ and the worst case value of $\mathcal{R}$. The essential point is that in every case, the comparison involving the two samples from the same connection had a significance level at least twice as great as the best comparison of an injected connection to a control connection. The reader is reminded that the significance here is on a logarithmic scale.

Run	N	med $\mathcal{R}$	worst $\mathcal{R}$
I & II	302	7.51	3.89
III	54	6.59	2.44
IV	28	7.23	3.49

Table 3: Cross control: ratio of significance of true comparison with best of control comparisons.

4.6 Applicability Beyond Ethernet

While the thumbprint mechanism we describe in this paper has many applications, we are focusing specifically on the assigning of signatures to interactive login sessions. So although the total amount of traffic crossing a large internetwork may be enormous, the portion of the traffic in which we are interested is quite small.

For example, we looked at the traffic statistics for the NSFNET internetwork[14]. For November 1994, the combined rlogin and telnet traffic of 1.024×10^{12} bytes, accounted for only 4.56% of the total traffic. Distributed evenly over the month, we find the data rate to be 3.95×10^5 bytes per second. Furthermore, if we use a machine with an available 50 million instructions per second, this would allow us to perform 126 instructions for each byte.

While the assumption that the traffic is distributed evenly is unrealistic, the fact that a single, moderately powered workstation could, in the steady state, apply 126 instructions to every byte of telnet and rlogin data crossing the NSFNET is remarkable. Furthermore, while the amount of traffic across the NSFNET doubled between November 1993 and November 1994, the traffic for telnet and rlogin increased at only about half that rate.

Similarly, a T1 data line can carry 1.9×10^5 bytes per second in total, while a T3 line carries 5.6×10^6 bytes per second. If we make the assumption that only 5% of these bytes are rlogin and telnet (as on the NSFNET) then our 50 MIP machine dedicated to this task has about 5200 instruction per byte on the T1 line, and 178 instructions per byte on the T3 line.

These calculations are of course simplistic - they neglect the fact that some work must be done examining headers of other protocols to determine that they must be ignored. We are also not in a position to assess the capabilities of suitable network interfaces. Nonetheless, the fact that upwards of a hundred instructions are available per byte on average in several contemporary network settings is very encouraging as to the applicability of this method, given an implementation on a machine dedicated to the purpose.

5 Conclusions and Future Work

Our main result is that it is easily possible, on an ethernet, to save summaries of interactive connections which can be stored in only a few tens of bytes per minute per connection. In the case where these connections are a few minutes long and have moderate data flows, it is then possible to compare these

summaries later and identify whether two connections have the same content or not with very low probability of error. This is true even when one of the sets of data being compared has passed through a tortuous route to Europe and back on the internet.

We are actively working to extend this result in various ways. Firstly, we wish to establish whether the methodology here is adequate when the connections have very low rates of dataflow, and very high rates. Secondly, we are still doing research to fine tune the statistical algorithms to give the best performance possible. We are also studying what is the best length of time to thumbprint over. Experiments to date have been done with one minute divisions, but we have found that many connections are very short and so we wish to make the thumbprinting interval short also.

We are also studying ways to break up the connection into pieces that do not depend on time, but rather on content based triggers. Success at this would obviate the need to synchronize geographically separated thumbprint stations.

Once this is done, it is our intent to build a prototype system to implement these ideas and make it available to the Internet community. We anticipate that this system, where implemented, will be capable of reliably tracking intruders who do not take adequate precautions to avoid it.

The main vulnerabilities of such a system will be, firstly, parts of the system being replaced by Trojan horses, and secondly, intruders encrypting their connections differently in each link of the extended connection chain. While both of these are within the capability of the more talented members of the intruder community, we believe that a tracing system such as this could raise the entry price paid to become an intruder, and, where deployed, would increase the risks and inconvenience of penetrating computers for all intruders. Such a system would not be a panacea, but might be a deterrent.

Acknowledgments

We thank Karl Levitt, Biswanath Mukherjee, Matt Bishop and the rest of our colleagues in the Security Group at UC Davis for helpful discussions on this paper. We also had useful input from Kevin Zeise and Scott Wadell of the Air Force Office of Information Warfare, and Bob Palasek of Lawrence Livermore Laboratories. We would like to thank Geof Staniford and the University of Liverpool, and Amarnath Mukherjee and Georgia Institute of Technology for the use of facilities at those institutions in our long range tests. Finally, we particularly wish to thank ARPA for their support of this research.

A Appendix

We calculate the probability density function (pdf) of

$$z = x_1 x_2 \ldots x_n \tag{15}$$

assuming that the x_i are distributed $U(0,1)$. Throughout, we take $f(x)$ to be the pdf of x, and $F(x)$ to be the cumulative frequency distribution (cfd) of x. To begin, we define

$$Y = \log(z); y_i = \log(x_i) \tag{16}$$

so that

$$Y = \sum_{i=1}^{n} y_i \tag{17}$$

Both Y and the y_i have range $(-\infty, 0)$. We can easily calculate $F(y_i)$ for each i, since

$$F(y_i) = \text{Prob}(\log x_i \leq y_i) = \text{Prob}(x_i \leq e^{y_i}) = e^{y_i} \tag{18}$$

Then

$$F(Y) = \int_{\sum y_i \leq Y} \exp\left(\sum y_i\right) \prod dy_i \tag{19}$$

This integral can be effected by making the change of variables

$$P = \sum_{i=1}^{n} y_i \tag{20}$$

$$q_i = y_i - y_1 \quad \forall i \neq 1 \tag{21}$$

$$\tag{22}$$

If we denote the linear transformation defined in these equations by M, then (19) can be rewritten as

$$F(Y) = \int_{-\infty}^{Y} e^P \det(M^{-1}) A \, dP \tag{23}$$

where $\det(M^{-1})$ is the Jacobean of the variable transformation, and A is a factor coming from integrating over the $n-1$ variables q_i (on which the integrand did not depend). It is possible, though a little tricky, to directly evaluate A and $\det(M^{-1})$. It is easier to sidestep this work by noting that, since M is a linear transformation, $\det(M^{-1})$ must be constant. A, on dimensional grounds, must be proportional to P^{n-1}. Thus

$$F(Y) = C \int_{-\infty}^{Y} P^{n-1} e^P dP \tag{24}$$

where C is an unknown constant. This integral is a standard form [15], and the result is

$$F(Y) = Ce^Y \left[\sum_{i=0}^{n-1} (-1)^{n-1-i} \frac{(n-1)! Y^i}{i!}\right] \tag{25}$$

The requirement that $F(Y) = 1$ at $Y = 0$ then fixes the unknown constant C at

$$C = \frac{(-1)^{n-1}}{(n-1)!} \tag{26}$$

Since $Y = \log z$ we can deduce the cfd for z as

$$F(z) = z \sum_{i=0}^{n-1} (-1)^i \frac{(\log z)^i}{i!} \tag{27}$$

Finally, differentiating this wrt z gives the pdf for z, which we earlier called $U^n(z)$

$$U^n(z) = \frac{(-\log z)^{n-1}}{(n-1)!} \tag{28}$$

From an implementor's perspective, it is easiest to use this in the form of $\log F(Y)$ which is close to a linear function in the region of interest and so can be efficiently approximated as a lookup table with linear interpolation.

References

[1] B. Fraser (CERT). Private Communication. 1994.

[2] K. Van Wyck (ASSIST). Private Communication. 1994.

[3] R. Bace. A New Look at Perpetrators of Computer Crime. In *Proc. 16th Department of Energy Computer Security Group Conference*, 1994.

[4] P. Neumann and D. Parker A Summary of Computer Misuse Techniques. In *Proc. 12th National Computer Security Conference*, pages 396-407, 1989

[5] C. Stoll *The Cuckoo's Egg.* Doubleday, 1987

[6] S. Snapp *et al.* DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and An Early Prototype. In *Proc. 14th National Computer Security Conference*, 1991.

[7] H.T. Jung *et al.* Caller Identification System in the Internet Environment. In *Proc. 4th Usenix Security Symposium*, 1993.

[8] S. Wadell. Private Communication. 1994.

[9] L.T. Heberlein, K. Levitt and B. Mukherjee. Internetwork Security Monitor: An Intrusion-Detection System for Large-Scale Networks. In *Proc. 15th National Computer Security Conference* pages 262-271, Oct. 1992.

[10] C. Stanfill and B. Kale. Parallel Free-Text Search on the Connection Machine System. *Communications of the ACM*, 29:1229, 1986.

[11] M. Solomon and E. Wimmers. Telnet Terminal Type Option. Request for Comments RFC 884, 1983.

[12] W. Krzanowski. *Principles of Multivariate Analysis.* Clarendon Press, Oxford, 1988.

[13] C. Chatfield and A. Collins. *Introduction to Multivariate Analysis.* Chapman and Hall, London, 1980.

[14] Statistics available by ftp from *ftp.merit.edu.* 1994.

[15] M. Abramowitz and I. Stegun, Editors. *Handbook of Mathematical Functions.* Dover, New York, page 71, 1965.

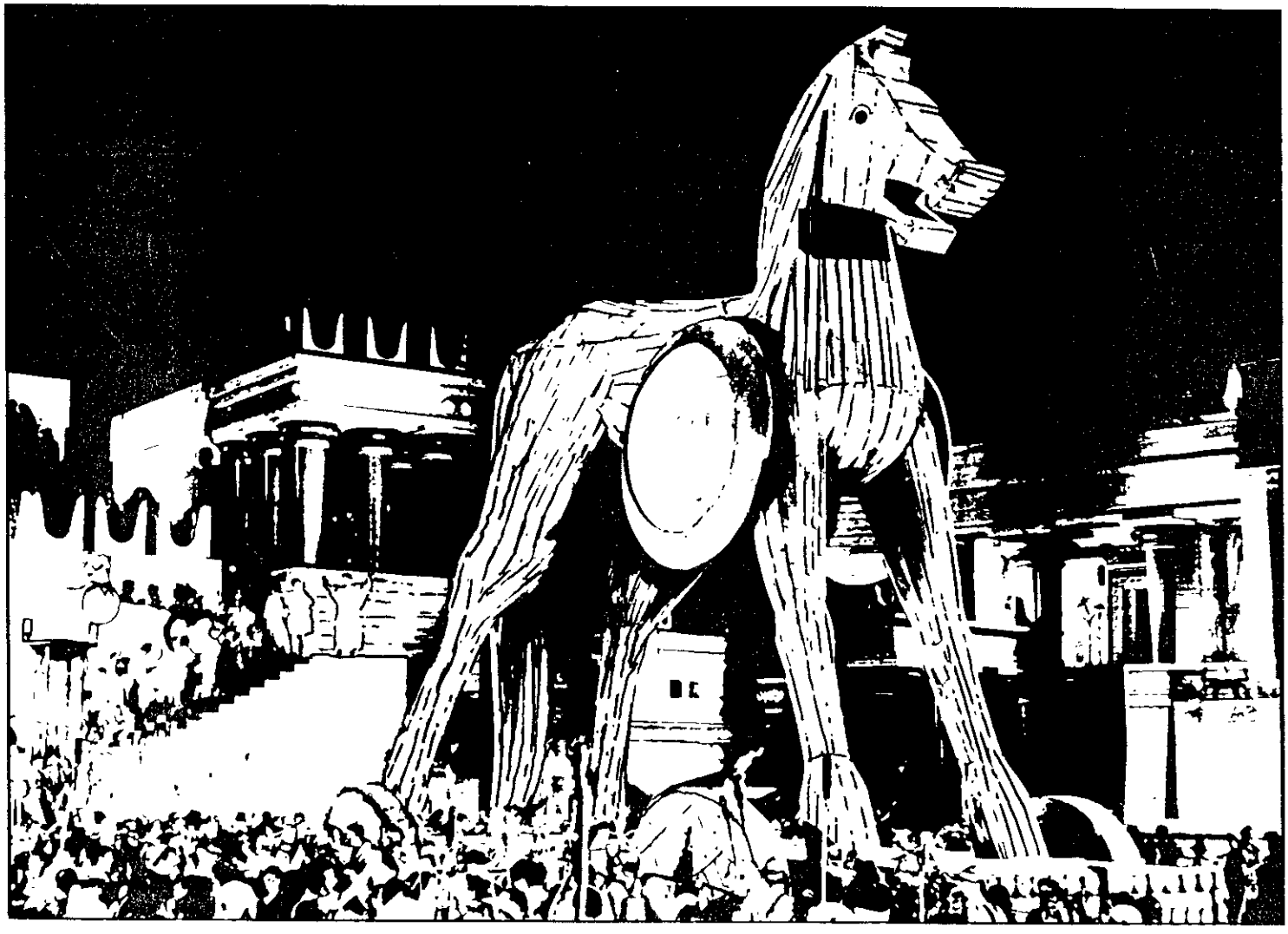
Proceedings

1995 IEEE Symposium on

Security and Privacy

May 8–10, 1995 **Oakland, California**

Sponsored by the
IEEE Computer Society
Technical Committee on Security and Privacy
in cooperation with
The International Association for Cryptologic Research (IACR)



IEEE Computer Society Press

The Institute of Electrical and Electronics Engineers, Inc.