CERIAS Tech Report 2000-24

Packet Tracker Technical Report 2

Florian Buchholz, Thomas E. Daniels, Clay Shields
Center for Education and Research in
Information Assurance and Security
&
Department of Computer Science, Purdue University
West Lafayette, IN 47907

1 Introduction

In today's Internet environment, there is a growing need for methodologies that help identify the origin of an attacker. Recent denial of service attacks have further emphasized the need for such technology. In the past few years, some solutions have been proposed in the research community. In this report, we will introduce and analyze two of those proposals, which seem promising: "Caller Identification System in the Internet Environment" by Jung et. al. [JKS+93] and "Distributed Tracing of Intruders" by Stuart Staniford-Chen [SC95]. We will also discuss experiences with the implementation of the two systems, and give a detailed evaluation of both.

The two approaches were chosen because they each represent one type of monitoring class that is feasible when dealing with traceback. Jung et. al. propose a host-based solution, whereas Staniford-Chen has developed a network-based approach. Each approach makes sense and both call for thorough investigation. There are pros and cons with both of them as will become clear in the following sections.

2 Host-based Tracing of Intruders

In this section we discuss our efforts to reimplement the Caller Identification System in the Internet Environment (CISIE)[JKS+93] as introduced by Jung, et.al. We begin by describing CISIE's components and functionality. Next, we present our plan for reimplementation of CISIE and discuss the plan's advantages and disadvantages. As a result of our work to reimplement CISIE, we next present several surprising results that make us doubt that CISIE was ever implemented or tested. We conclude the section by describing new operating system features needed for CISIE to work properly and also detail some cryptographic mechanisms that could make a system based on CISIE more resistant to attack.

2.1 Summary of CISIE

CISIE was developed to trace network attackers across multiple possibly compromised hosts. Such a situation where an attacker remotely logs into a host and from there logs into yet another is called an extended connection. The difficulty in tracing extended connections manually is that hosts do not normally store information that can easily be used to correlate an outgoing connection with its corresponding incoming one. CISIE was designed to alert hosts in an extended connection about all previous "hops" in the connection.

The hosts could then use this information to augment their audit trails or even for disallowing logins.

2.1.1 Components of CISIE

CISIE is made up of two components, each of which is installed on every host: the Caller Identification Server (CIS) and an Extended TCP Wrapper (ETCPW). The CIS keeps a trace for each user who has remotely logged into the host. A trace is a list of previous hosts (and user identifiers oh those hosts) in the user's extended connection. The ETCPW is an extension to Wietse Venema's well-known TCP Wrapper package[Ven]. When user requests a login service from the host, the ETCPW tells the local CIS to ask the CIS on the host that made the request for a full trace of the user.

2.1.2 The CISIE Protocol

The initial phase of the CISIE protocol is similar to conventional TCP Wrapper configuration that uses the ident protocol[Joh93]. The server's ETCPW signals the local CIS to do a request for information about the incoming service request. The local CIS then makes a request to the remote CIS which returns a trace for the user. This request consists of the TCP ports that along with the host addresses identify the TCP connection. If this were an ident request, the remote host would reply with the user identifier that owns that end of the connection. Instead, the remote CIS responds with the trace information associated with the user. Note that if this is the first hop of an extended connection, the information is basically the same. The local CIS then stores trace for later use.

The second phase of CISIE consists of authenticating the trace received in the initial phase. To do this, the CIS contacts all hosts in the trace except for itself and its predecessor. It asks the CIS on each of the hosts if the user is logged in. If all reply in the affirmative, the trace is assumed to be valid and the user may continue logging into the system. Otherwise, login is denied and an alarm may be raised.

2.2 Reimplementing CISIE

Initially, we had hoped to obtain the source code of CISIE from the authors of the original paper. However, our attempts to contact them failed possibly due to language issues or because the primary author of the paper no longer appears to be at the Seoul National University. Since we were not able to get the source code for CISIE, we have had to reimplement the system

solely based on the paper. Here we describe the details of our effort to reimplement CISIE. First we describe our development platform and then we'll give a description of our design. We finish the section with the current status of the implementation.

2.2.1 Development Platform

We chose to implement CISIE for the Linux operating system with kernel version 2.2.12. Using Linux for this type of development simplifies several aspects of the project. First, since Linux is open source, we can examine and even modify kernel source code if necessary. Second, in Linux the interface used by the ident service is provided via the proc file system. This is a virtual file system that a user process can read to query data from the kernel. In other systems, the data must be manually read from kernel memory.

We chose the Java programming language for implementing CISIE. Java gives us a mostly cross-platform code base and easy network programming primitives. It also makes it easier to write readable code for others to examine in the future.

2.2.2 Our Design

Our design is very similar to that proposed in the paper[JKS+93]. We chose to modify TCP Wrappers as little as possible, and to use features of the wrappers to implement much of what was done in the ETCPW. Our CIS component aims to have the nearly the same functionality as that described in the paper.

In examining TCP Wrappers, we found that by configuring it appropriately we could have the wrapper spawn a program of our choice upon a service request. Furthermore, the wrapper could pass the IP addresses of both ends of the connection to the program. Unfortunately, the wrapper did not allow for passing the other information needed for the trace, the port numbers. This means that the only necessary modifications to TCP Wrappers was to add the ability to pass port numbers to other applications. This is a relatively minor change, and we plan to submit a patch to the package's maintainer as it may be useful in other situations.

When a remote login request is received, the wrapper issues a line of command shell code that launches a program of our design called *Wrap* with the address and port information specified on the command line. Wrap then sends this information to the local CIS using a localhost-limited TCP socket thereby requesting a trace. When the trace request is complete, CIS passes

the results back to Wrap which then outputs the trace to standard output so that a calling shell script can allow or disallow the connection.

The CIS portion of our design is nearly the same as that in the paper[JKS+93]. The CIS, as is Wrap, is written in Java and uses multiple threads of execution to handle simultaneous requests. Our CIS maintains two listening server sockets. One of them is restricted to the local host and implements the communication between Wrap and CIS. The other handles requests from the CIS's on other hosts.

2.2.3 Status of Implementation

For technical reasons described below, our implementation of CISIE is nearly complete, but it does not function correctly. Wrap is complete, and CIS currently attempts to implement phase 1 of the protocol as described above. Phase 2 of the protocol in which the trace is authenticated is nearly implemented. It has been postponed due to problems replicating phase 1. The incompleteness of the work actually yields our primary result which will be described below.

2.3 Results of The Implementation

In our attempts to implement CISIE, we have discovered a major technical difficulty that is not mentioned in the original work. The problem lies in determining how to link incoming and an outgoing TCP connections so that the trace can be assembled. Initially, it appeared that the same mechanism used in ident could be used for this purpose, but it seems that this is not the case. An ident daemon can determine the client user's identifier for a TCP connection because the process that creates the connection usually runs with the user's privileges. This implies that the local CIS can determine the user identifier on the remote host. The problem of determining the local user identifier associated with an incoming connection to the local host is different. Anything run by TCP Wrapper can not know this immediately because the user has not logged on yet. The local end of the connection was also originally connected by inetd, which runs as root, therefore the user identifier associated with the local end of the connection is root.

The result of this is that there is no easy known way to use the available data to correlate the incoming and outgoing trace. This sheds doubt on whether or not Jung, et. al., had an actual working implementation of CISIE. The matching of incoming and outgoing connections in CISIE appears to be a difficult problem and hence would surely have been an issue

covered in their paper. In this light, other inconsistencies show up in the paper. One is regarding the method used to authenticate the streams in phase 2. In Section 2.2 of the paper, item 7 suggests that the CIS's authenticate a substantial part of the trace whereas in Section 2.3.3, we see that the CIS's are just verifying that a user identifier "has a process." Finally, the issue of whether CISIE has been implemented is never directly addressed. We might expect implementation details such as operating system and version, programming language, and possibly real-world experiences with the system, but there are none.

2.4 Modifications to CISIE

We had proposed to experiment with modifications to CISIE, but since we have had to reimplement a system that has substantial undocumented or unknown issues, we have not made any such modifications. Below, we roughly describe some possible additions or techniques to first make CISIE work and secondly to secure and enhance CISIE.

2.4.1 System Features Needed

As described above, the operating system (Linux in this case, but Unix-like operating systems in general) do not provide any easy way to match the outgoing connections with the incoming ones. It may be possible to do so with a variety of techniques with varying degrees of accuracy. We foresee future work in determining the best mechanism to match these connections. Possibilities for this include searching through kernel data structures to attempt to make the match, or modification of the kernel such that it denotes all processes belonging to a given user session with a unique identifier that can be used to match the incoming and outgoing connections. Further work will be required to determine the difficulty of this system modification and its possible impact on the system's performance.

2.4.2 Securing CISIE with Cryptography

The CISIE protocol as initially presented in the paper ignores the security of the CISIE protocol itself. Because of this, a savvy attacker could pretend to be a previous CIS or possibly query the CIS's in the network to track authorized users to either subvert the system or compromise the users' privacy. The end of the paper suggests a public key method to prevent this, but it is flawed. Here we present a number of suggested modifications to the CISIE protocol to help secure it.

The paper's brief section on securing CISIE suggests that public key cryptography be used to verify that the sender of a Caller Path Request is a CIS. Their suggestion is that a replying CIS would encrypt the reply with the recipient's public key. This does make it so that only the appropriate recipient could read the reply, but it ignores replay attacks. It also ignores the overhead involved with a public key infrastructure (PKI).

Our suggestion is that if the expense of a PKI is warranted, we can do a much better job than this. Since most interactions in CISIE are of the form client request and server answer, we could simply have the client encrypt requests with the server's public key and include a random nonce in the message. The server can then respond by including the nonce sent by the client, encrypting with the client's public key, and signing the message with the server's private key. This simple set of modifications should eliminate most forms of replay attack. Further modifications might have the client sign each request as well so that only valid CIS's can request a trace be done If the CISIE is made to work, we still may experiment with these modifications to the protocol to determine the overhead that cryptography adds to the protocol. One area of possible study is to consider modifications that do not require a public key infrastructure thereby allowing for lighter weight protocols.

2.5 Conclusions about CISIE

We have made a substantial effort to reproduce results reported by Jung, et. al. Unfortunately, we have been unsuccessful and even doubt that an implementation existed prior to the writing of their paper. We have also discussed modifications to the Linux (or other) operating system to make CISIE work. Finally, cryptographic methods have been considered to help secure the CISIE protocol, but implementation seems unwarranted until the protocol has been shown to work.

3 Network-based Tracing of Intruders

In order to determine the identity of a network attacker, it may be necessary to be able to trace back a chain of connections between network hosts to its origin. One might consider performing this task at the host level of each machine in the chain. However, there are a number of problems with this. A user might have multiple open connections, making it difficult to associate specific ones to those on other hosts. Header information can change, also complicating the successful matching. A user might even change

services within the chain of connections on a particular host, e.g., from *telnet* to *rlogin*. Covert channels on a host, even though unlikely, can render a host-based solution virtually worthless. Last but not least, a host can be compromised by an attacker, leaving the usefulness of any audit data questionable. Thus it seems reasonable to also examine and pursue network-based approaches.

One approach, taken by Stuart Staniford-Chen [SC95], is to analyze properties of the TCP stream itself between pairs of hosts. The goal of the approach is to find certain characteristics within the stream that uniquely sets it apart from other streams. Ideally, those characteristics will persist throughout the chain of connections. Staniford-Chen introduces the concept of *thumbprints* to achieve this goal. A thumbprint is a short piece of data that summarizes the content of a connection. It should have the following properties: small size, sensitivity to the content, and robustness. Furthermore, the thumbprints should be additive, easy to compute, and be easily comparable. Under the current model, only *telnet* and *rlogin* sessions are considered. This is due to the fact that content within the stream is used, and these services do not change the content at each hop as some encrypted services do.

In this model, a number of network sniffers are placed at appropriate points in the network, and at each location information about each TCP stream is collected. Each stream is divided into intervals, typically of one minute length. This allows for comparing portions of the stream. Over the time of one interval, a vector, represented as an array of integers, is used to keep track of the number of occurrences for each of the possible 128 ASCII characters in the TCP packets of the stream. This vector by itself might suffice as a thumbprint as it identifies a connection or a stream. However, the vectors are quite large, and effectively comparing two vectors might become difficult.

To compress the size of the thumbprint and to provide better means of comparing them, Staniford-Chen proposes principal component analysis to provide a function that emphasizes more unusual characters over those that are frequently used in all or most telnet or rlogin traffic.

3.1 Local Thumbprints

The author proposes what he calls *local thumbprints*, where "[...] the thumbprint is a sum of terms, each of which depends only locally on the character stream to be thumbprinted" [SC95]. Thus, for a simple thumbprint vector T_j with j components, he suggests $T_j = \frac{1}{n} \sum_i \phi_j(a_i)$, where n denotes the number of

characters in the alphabet and the a_i stand for those individual characters. In the experiments described in the paper, this definition of a thumbprint is used, and therefore will be the only one discussed here.

The function ϕ_j is obtained through principal component analysis described below. Essentially, for each character a_i of the alphabet and each component j of the thumbprint, ϕ_j is simply a factor that helps emphasizing characters that are more atypical for a telnet session. A big advantage of having a fairly easy thumbprint function is that it is very easy to compute. Most of the work is spent obtaining the function, which can then be stored in a simple look-up table.

3.2 Comparing Thumbprints

For comparing thumbprints effectively, Staniford-Chen develops a scheme that magnifies differences between components and considers the proposition that, when successive thumbprints match over a longer time period, they are likely to be related. For this purpose, a probability approximation $P'(\delta)$ is created by randomly choosing pairs of thumbprints out of a sample pool and calculating logarithm of the product of the differences of the individual components. The resulting distribution can then be used to approximate the probability of a given δ. The smaller the probability, the more likely it is that the streams are related. For successive intervals of time, the individual probabilities can be multiplied. The author refines this method to a great extend, using statistical models, which are beyond the scope of this report.

Again, as for the ϕ-function discussed above, most of the work is performed in advance on sample data. Once the approximate distribution is found, comparing pairs of thumbprints is a matter of a few multiplications and additions. Note, however, that one cannot know in advance which thumbprints should be compared with each other. The author doesn't address this issue directly.

3.3 Principal Component Analysis

Principal component analysis is a methodology adapted from statistics, and Staniford-Chen gives two references [Krz88, CC80]. The goal is to find directions in a set of vectors that respresent more of the variation within those vectors than any other direction. For this purpose, a covariance matrix is calculated over the elements of the vectors. Then, the eigenvalues and eigenvectors of the matrix are computed. The eigenvectors that belong to the largest eigenvalues are then taken, and its individual components

describe the function ϕ.

Assume that the thumbprints have c components over an alphabet of n characters. For a sample of N vectors, the following steps need to be taken to calculate ϕ:

1. Calculate the covariance matrix $C_{jk} = \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \overline{x_j})(x_{ik} - \overline{x_k})$, where $\overline{x_j}$ and $\overline{x_k}$ are the averages in each direction, respectively.

2. Take the c largest eigenvalues of C_jk and the eigenvectors that belong to those, $\vec{e_1}, \cdots, \vec{e_c}$.

3. ϕ_1 now corresponds to $\vec{e_1}$, ϕ_2 to $\vec{e_2}$, and so on. I.e., $\phi_j(a_1)$ is the first component of $\vec{e_j}$, $\phi_j(a_2)$ is the second component of $\vec{e_j}$ etc.

3.4 Implementation

We asked Mr. Staniford-Chen as well as faculty at the University of California Davis for source code. This request being denied, the thumbprinting approach is currently being implemented using the machines provided for the project. The software platform for the project is Linux, Kernel version 2.2.12. The following assumptions had to be made, as the paper does not discuss them:

- All TCP packets without explicit data content such as SYNs, ACKs, and FINs are not considered for the thumbprint calculation.

- It is assumed, that the eigenvectors obtained as described above indeed constitute the factors of the function *phi*. The author never explicitly mentions this, but it is the only reasonable approach.

- It is assumed that taking the first c largest eigenvalue/eigenvector pairs also results in the thumbprint consisting of c components.

- TCP connections will all start with a SYN packet and will all end with a FIN packet.

Our implementation differs from that of Staniford-Chen in that we differentiate between the two ends of a TCP connection. By doing so, we hope to achieve an even better characterization of the stream. If desired, one can always add the two thumbprints to achieve one that characterizes both directions of the stream.

For the purpose of examining the TCP streams, the *libpcap* library [Lab] was utilized. Much of the programming code was adapted from Stevens [Ste98] from examples using that library.

The parameters were kept the way as described in the paper [SC95]. Thus the time interval has a length of one minute, only the first 128 ASCII characters are considered, and the thumbprints will consist of 6 components.

The program establishes a counting array for each individual connection, that is, for each 4-tuple <*source IP-address, destination IP-address, source port, destination port*>, an array with 128 counters - one for each possible character - is created. As data is being sent, the counters are increased appropriately. An alarm will interrupt every 60 seconds, and the arrays can then be treated as the vector that describes the stream. The thumbprint function can then be applied and the thumbprint be stored. After that, the counters in the array are reset to 0.

Obtaining good quality sample data is a very important aspect of the implementation. Indeed, the overall success will almost solely depend on it. After a collection of artificially generated sample TCP traffic data that was provided to us early in the project proved to be insufficient for our purpose, actual user data is currently being gathered within the Computer Science Department of Purdue University.

For this reason, the developed program yet lacks a proper thumbprint function, and the comparison routine is not yet existent.

By the time this report is finished, the first data set should be available, and we will be able to proceed to evaluate the ϕ-function as well as the probability distribution $P'(\delta)$. A second data set will be generated and compared against the first. After that, the ϕ-function can be incorporated into the program, and the comparison routine can be developed.

3.5 Shortcomings

There are several flaws and shortcomings with the thumbprint approach.

1. The whole computation of the thumbprints is solely based on content. If encryption is used with different key pairs between different hosts, the thumbprints will differ greatly, indicating (falsely) different TCP streams.

2. It is unclear why Staniford-Chen limits his work to an analysis of content. Clearly, similar vectors to the content-based ones can be created for packet frequency and idle times. Those could be processed in a similar fashion and evaluated. Thumbprints obtained that way could

then be compared within their own classes (content, timing, idle-time), or one could think of a way to combine them, taking all the characteristics of the stream into account. This way, the thumbprinting scheme would not solely depend on content alone, and maybe could still suffice in analyzing encrypted traffic. However, there are also ways to perturb the other two characteristics of the stream.

3. The choice of Principal Component Analysis to determine the optimal thumbprint function is disputable. Even though "[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 [...]" [SC95], the author never argues why this approach is the best choice. Other methods for obtaining ϕ, such as a genetic algorithm or machine learning scheme also seem feasible.

4. Several other choices the author made seem rather arbitrary and are not explained satisfactorily: why sample the stream in intervals of one minute length? He only differentiates against a 10 second long interval. Why use only the first 6 principal components?

5. The author doesn't provide any ideas as to how to choose thumbprints that should be compared with each other. Initially, when comparing a particular thumbprint of a stream that was used for an attack on host A to thumbprints of host B, one might limit the range of prints to compare to those that lie in a certain time interval prior to the attack. However, if no thumbprints match, it does not guarantee that host B was not part of the chain of connections, as an attacker could certainly have caused a delay in execution of commands. Thus, in the worst case, all thumbprints of host B have to be compared, resulting in a very negative performance impact.

3.6 Conclusions about Staniford-Chen

Stuart Staniford-Chen's work is a start. It allows for an efficient and fast computation of local thumbprints for *telnet* and *rlogin* sessions. The underlying theory has a well-established foundation in statistics. A major advantage of the approach lies in the fact that a vast part of the computation can be performed in advance, reducing the actual calculation of a thumbprint to a few mathematical operations. Local thumbprints seem to

work extraordinarily well within the given environment, even though this claim has yet to be verified.

However, there are many unanswered questions. The most obvious flaw is, of course, the solitary focus on data content alone. Staniford-Chen makes it rather easy for himself this way, leaving out other important information about the data stream. Besides content, there is also meta information about the stream. One can look at timing, idle times, and at header information that remains constant. Ideally, all those factors should be incorporated into a thumbprint. In order to prevent a thumbprint from becoming useless if one or more of the session characteristics are circumvented, the components should be stored separately. A new comparison function needs to be created that combines the components, but also allows for individual inclusion or exclusion of features.

Even though Staniford-Chen gives a thorough discussion about the underlying theory of his approach, it becomes obvious that the work he has performed covers only a small part of what needs to be done. In many cases, he arbitrarily picks the value of a parameter, sometimes after only a little pre-evaluation, and then sticks with it for the rest of his experimental results. There are many parameters to this problem, generating a huge search space for the optimal combination. There are many open questions concerning these parameters. What is the best number of thumbprint components? How long should the timing interval be? Are local thumbprints as good, better, or worse than the higher-order thumbprints he discusses? What constitutes the best thumbprint function? All these questions will have to be answered in the future.

Another concern is that of the thumbprint ϕ-function. The way it is constructed leads directly to the following questions: What kind of sample data needs one to take to best capture "ordinary" *telnet* and *rlogin* features? To what extent do individual or sequences of characters stand out to characterize individual streams? Is there a single "ideal" thumbprint function? Right now, the ϕ-function is static. It is pre-computed, and will never change from that point on. It might be desirable, to be able to adjust the function, as features of TCP streams might change over time. As a matter of fact, this might be a major disadvantage of principal component analysis. In order to change the function, a new covariance matrix needs to be computed, and its eigenvectors and eigenvalues calculated. This amounts to the same extensive work as the initial computations. A machine learning approach could lead to more efficient results. Once the training phase with the sample data set is complete, the functions can be updated by incorporating new samples into the process. This can be done without having to recompute any previous

calculations. Thus, a machine learning approach and its effectiveness might be well worth pursuing. This way, it might also be possible to find other important distinctions between data streams than the greatest variation.

Comparing TCP streams on a network level is very important. Staniford-Chen introduces a new concept for doing so, and he explores a small fraction of the overall framework. There is yet plenty of research to be done in the area. If the obstacles mentioned in this report can be overcome, this technique could evolve into a very useful methodology. Note, however, that any attempt of matching streams is subject to countermeasures by a determined attacker. In addition to encryption, tools can be created that randomly obscure timing and idle time information on a per session basis. Streams can be split up, sent through chains of different hosts, and later be combined again. Those are problems that are yet to be solved.

4 Conclusions

In this paper, we have discussed our work towards implementation and evaluation of two known approaches to tracing network intruders. CISIE appears to have been designed but possibly not implemeted by its authors. Our attempts to recreate their system uncovered the crucial problem of correlating incoming TCP connections with the outgoing ones. This is difficult although probably doable on the host by either using intimate knowledge of the operating system to make the correlation or by making minor modifications to the kernel of the operating system itself. We have also outlined some possible approaches to securing CISIE against tampering. We have also discussed Staniford-Chen's thumprinting techniques. We conclude that while his work is a good start in the area, there remains a great many unanswered questions about the work. There are many parameters of his approach that seem set arbitrarily, and it is unclear that the scheme will work in the real world, in which attackers use encryption to hide the contents of their data streams.

References

[CC80] C. Chatfield and A. Collins. *Introduction to Multivariate Analysis.* Chapman and Hall, London, 1980.

[JKS+93] Hyun Tae Jung, Hae Lyong Kim, Yang Min Seo, Ghun Choe, Sang Lyul Min, Chong Sang Kim, and Kern Koh. Caller iden-

tification system in the internet environment. In *UNIX Security Symposium IV Proceedings*, pages 69–78, 1993.

[Joh93] M. St. Johns. Identification protocol. Request for Comments 1413, February 1993.

[Krz88] W. Krzanowski. *Principles of Multivariate Analysis*. Clarendon Press, Oxford, 1988.

[Lab] Lawrence Berkeley National Laboratory. ftp://ftp.ee.lbl.gov.

[SC95] Stuart G. Staniford-Chen. Distributed tracing of intruders. Master's thesis, University of California Davis, 1995.

[Ste98] W. Richard Stevens. *UNIX Network Programming*, volume 1. Prentice Hall, Upper Saddle River, second edition, 1998.

[Ven] Wietse Venema. TCP wrappers. available at `ftp://ftp.win.tue.nl/pub/security/tcp_wrappers_7.6.tar.gz`.