Balancing Privacy and Fidelity in Packet Traces for Security Evaluation

Sonia Fahmy and Christine Tan
Department of Computer Science, Purdue University
250 N. University St., West Lafayette, IN 47907–2066, USA
Phone: +1-765-494-6183, Fax: +1-765-494-0739, E-mail: fahmy@cs.purdue.edu,
christinetan1@yahoo.com

Abstract

Security mechanisms, such as firewalls and intrusion detection systems, protect networks by generating security alarms and possibly filtering attack traffic, according to a specified security policy. Evaluation of such security mechanisms remains a challenge. In this work, we examine the problem of compiling a set of high fidelity traffic traces, that include both attacks and background traffic, to make them available for trace-based evaluation of Internet firewalls and intrusion detection systems. For these traces to be representative of real-world Internet traffic traces at the time they area used, synthesizing or generating a trace is inadequate. Hence, developing an anonymization tool for captured traffic traces is necessary. This will ensure that traces reflecting current traffic characteristics can continuously be made available to the network security research community. The primary challenge in this process is not to compromise user privacy. We identify private information that can be inferred from an Internet traffic trace, including both packet headers and payloads. We then design a set of tools that read a configuration file and a traffic trace, and anonymize the the trace accordingly. We add "noise" to the traces to enhance privacy, while preserving *the long term* characteristics of the traces. Finally, we design simple mechanisms for reordering (mixing) packet streams (packets in a flow) to make it difficult to infer private user information.

Keywords

firewalls, intrusion detection systems, denial of service, trace sanitization, trace anonymization, trace transformation, privacy

I. Introduction

FIREWALLS , intrusion detection systems (IDSs), and other security mechanisms detect and protect networks from attacks by analyzing, filtering, and managing Internet traffic. Despite their critical role, such security mechanisms have traditionally been tested without well-defined and effective methodologies. The ultimate goal of our work is to enable objective trace-based comparisons of the relative merits of security mechanisms, in order to allow end users to choose the appropriate mechanism for a certain network. Previous attempts at developing security benchmarks include the 1998 and 1999 evaluation of intrusion detection systems (IDSs) conducted by the Lincoln Laboratory of MIT [1]. Most such attempts suffered from a number of flaws [2] that call into question the usefulness of the results obtained. Developing a security testing methodology is a significant undertaking, and hence we will only focus on a single aspect: generating realistic (high fidelity) traffic traces. Having such traces is crucial for security evaluation, e.g., it is necessary for computing a meaningful false positive rate for an IDS.

In the Lincoln Lab effort, background traffic was generated artificially with users simulated by software automata. The reason for not utilizing collected traces was that examining every single packet to remove private data was deemed infeasible, and it was deemed that it would be too complex to prevent the introduction of artifacts when attacks are inserted. The Lincoln Lab approach, however, cannot faithfully reproduce certain properties of actual traffic on the original network. Such properties must be preserved if the measured false positive rate of an IDS is to be accurate. Typical network traffic is not well-formed, as it may exhibit unusual or unexpected behaviors due to variations in protocol implementations, unpredictable user behavior in the real world, or attacks. In addition, no measurements were taken to validate the Lincoln Lab claim that their background traffic was indeed similar to the traffic on the real network they were attempting to simulate. Our goal is to overcome these pitfalls by employing trace-based generation of background traffic.

Fidelity when generating traces must be balanced by the goal of protecting private or sensitive information present in the original network traces we use. These two goals often conflict, as we will discuss in section II. We will examine this tradeoff at the level of a single packet in sections III and IV, then at the level of a stream (connection) in section V. We propose two trace transformations (based on mixing and random noise) with respect to our goals in section VI. We include a discussion of possible attacks against our anonymization and transformation schemes that would allow an attacker to extract private or sensitive information through traffic analysis. We discuss related work in section VII. Finally, section VIII summarizes our conclusions and discusses future work. We give example trace transformation results from our preliminary experiments in the appendix.

II. Background and rationale

Our ultimate goal is to develop a tool for transforming traces that are suitable for testing security mechanisms such as IDSs. Our tool will have a simple interface instead of having a user write complex policy scripts. However, we need to make our tool as flexible as possible since different networks may need different levels of privacy. We will do this by allowing the user to specify for

– This work has been sponsored by the CRA-W DMP program in Summer 2003.

all IP, TCP, UDP, and application header fields whether they are to be removed, mapped, or left in the trace. Application payloads are present since IDSs may check for certain strings in payloads as part of an attack signature.

In [3], a tool for sanitizing application payloads was introduced, which reassembles packets to obtain application protocol data, applies a "filter-in" principle to application payloads in which all text that is not explicitly instructed to be left in is replaced, and it finally converts the parsed data back into an ASCII trace. Only correct application data transformation is performed, since their aim is to sanitize traces for general purposes, which limits the changes they can make to the original trace. While one of the goals of that work was for testing intrusion detection systems, certain aspects of its implementation may not be sufficient to ensure that properties of the trace that are *not* well-formed are preserved. Specifically, the trace composer that converts the data back into a trace aims to generate "correct" traces with well-formed connections such that certain fields are recomputed and overwritten with correct values. If the original packet had incorrect values, either due to incorrect protocol implementations, or because the packet is part of an attack, such properties that may be identified by an IDS as suspicious are lost. In addition, in their implementation, fragments are not preserved, but since many Denial of Service (DoS) attacks make use of malformed fragments, removing this in a trace that is to be used as background traffic for detecting such attacks can affect the false positive rate. Preserving such properties, however, involves greater complexity, which can become an issue if traces are very large. Since anonymization is done off-line, the amount of time taken to process the trace is not crucial as long as we ensure that optimizations are made where possible, such that the time taken to process the trace is within a reasonable limit.

III. Information leakage if only one header field is known

We will examine how each field of a packet can be used to infer private information and the importance of each field in attacks. We must balance these two factors in our decision of whether to preserve, remove, or map the value to a new value. For the fields in the IP header, all fields could be part of an attack signature but not crucial to the attacks execution. We will only take note of those fields which are crucial for the attack to work or unique to that attack.

A. IP header

In the IP header, the version number can reveal if a certain part of the network topology supports IPv6 instead of the more common IPv4. An IDS needs the version number in order to interpret the packet correctly.

In the IPv4 header, the application can be inferred from the header length only if it is a unique to that application. If the application only runs on certain operating systems, the operating system can thus be inferred. For example, an IDS may log IP header lengths that are less than 5 to detect scanning, as discussed in [4].

The 8-bit type of service (TOS) field consists of a 3-bit "Precedence" field, a 4-bit "Type-of-Service" field, and a 1-bit "MBZ" (must be zero) field. The IP precedence field in ICMP error messages can reveal the operating system due to bad implementations [5], [6], [7], [8], [9]. Knowing the value of the "Type-of-Service" field can narrow down the possible applications being used, or the Internet Service Providers (ISPs) from which the trace was taken. If the value of the TOS field is unique to an application or an ISP policy, the application or ISP can be deduced.

The total length field can indicate the transport protocol or the type of data sent, which can narrow down the list of possible applications. Some operating systems miscalculate the IP total length field [10], which can identify the operating system on the host. The IP header length subtracted from the total length gives the size of the IP payload, which can indicate the type of data sent, and hence the application. Attacks may be identified by the size of their payload, such as the Tiny Fragment attack, in which fragments are made so small that not all the TCP header information fits into the first fragment, in order to get past firewalls which apply filtering rules to the first fragment only [11], [12].

The IP identification field may not be echoed correctly by some operating systems such that the bit order of the field is changed [10]. The identification can be part of an attack signature of a flooding attack, where a flood of identical packets is sent [13].

The 3-bit flags and offset fields may not be echoed correctly by some operating systems such that the bit order is changed [10]. The application could be deduced if the flags are unique. Invalid IP flags can be detected by an IDS since this can indicate that someone is sending malformed packets to determine a host's operating system, or to get past firewalls [14].

The time to live (TTL) field can give an attacker an estimate of how many hops the packet has traveled since it left the original source. Since the (anonymized) source and destination IP addresses are known, the TTL field only reveals information about routing at that point in time.

The protocol number identifies the next level protocol, so the transport protocol is known. This can narrow down the type of application being used.

The header checksum may not be computed correctly by certain operating systems. Attack packets could have a checksum of zero, which should be discarded by a router but may cause it to crash.

B. TCP header

The source and destination ports can identify the application being used. If the application only runs on a certain operating system, the operating system could be revealed. An attacker may be able to pass through a firewall by using a particular port

number. If this port number also appears in legitimate packets in the trace, this could affect the measured false positive rate. Port numbers are commonly used in firewall and IDS rules.

By studying the predictability of initial and subsequent sequence numbers, the operating system can be easily fingerprinted. A possible attack that makes use of sequence numbers is exploiting a flaw in how sequence numbers are interpreted [15].

The data offset gives the size of a TCP header, which can identify an application if it uses unusual options. An attack could be based on having TCP headers of an unusual size.

The reserved field is not normally used. If an application or operating system fills this field with a unique value, they could be identified. An attacker could possibly use this field as a covert communication channel.

TCP flags can indicate the application used if the combination of flags is unique. Flags can also indicate whether a connection is being set up and who initiated the connection. Illegal combinations of TCP flags can be used in Denial of Service attacks or to bypass checks in firewalls or IDSs.

The TCP window size gives congestion information at the time the trace was taken, which does not seem to be sensitive.

Some operating systems do not correctly compute the checksum, making it possible to fingerprint these operating systems.

The TCP urgent pointer can narrow down the possible applications if only certain applications use this field.

TCP options can give routing information if the Loose Source Routing or Strict Source Routing option was used. An attacker may use the routing option to control the path of a packet as part of an attack. If the padding is given an unusual value, this could fingerprint an application or operating system.

C. UDP header

The length can be used to deduce the type of information being sent or the application-level protocol, which can identify the application. If all fields in the header except IP addresses are known, the checksum can narrow down the possible list of IP addresses.

D. Application headers and payloads

In this section, we discuss some sample popular application protocols.

D.1 HTTP 1.0 or 1.1

The method entry indicates whether the message is a request or a reply. This could be used to distinguish Web servers from clients.

The HTTP version number is not sensitive. In contrast, the URL is sensitive. A query string may be unique to a certain web site. The presence of a query string can indicate that a database is possibly being queried. Parts of a bad URL can be used in an attack for remote execution of code.

Optional header fields can reveal additional information that may be sensitive, including date, server, last-modified, content-length, content-type, browser, if-modified-since, accept, referrer, user-agent, host, expires, MIME-Version, and connection.

D.2 SMTP

The sender, receiver, server, and client are sensitive, as are most other header fields, including Received, Return-Path, cc, Subject, Reply-To, In-Reply-To, Message-Id, Delivered-To, Errors-To, Bounces-To, X-Mailman-Version, X-BeenThere, List-Id, List-Unsubscribe, etc. Some of the less sensitive fields include Content-Type, MIME-Version, Content-Transfer-Encoding, Content-Disposition, User-Agent (assuming it is not an unusual mail program), and Precedence. The message body is extremely sensitive, but an attack could be embedded in the message.

D.3 FTP

FTP commands include sensitive information such as user name, password, and account information, path names, and host port. FTP server replies may include sensitive information such as user name, password, IP address and port number of local machine, path names, file names, number of bytes sent, server operating system, client machine name and operating system. The text varies according to implementation.

D.4 Telnet

A telnet connection is a TCP connection used to transmit data interspersed with telnet control information. Each end of a telnet connection is assumed to start and end at a Network Virtual Terminal (NVT). Once a telnet connection is established, option requests are sent back and forth between both parties to determine NVT options. User names and passwords are sent to a remote machine in clear text. Data sent over a telnet connection may include file names and file contents, as well as every command executed by the user on the remote machine.

D.5 IRC

IRC messages are lines of characters terminated with a carriage return/line feed pair [16]. Sensitive information included are password, user name, hop count, host name, server name, real name, channel, topic, quit message, nickname, key (password) to join a channel, comment for the KICK command, server version, server statistics, all servers known by the server answering the LINKS query, server's local time, port number, target server for the server to connect to due to CONNECT message, route to a server in response to TRACE command.

D.6 ICQ (Version 5)

ICQ client to server packets can include sensitive information such as password, client build date, number of contacts, user ID, time, port, message text, URL, and "general info" which includes name, phone, address, etc [17]. The sizes of each of these strings is also included in packets. The format of packets varies according to the type of command being sent. ICQ server to client packets can include IP address, port, user ID, user info (nickname, name, email, authorization status), URLs, list of contacts by nickname and user Id's.

D.7 AOL instant messenger (OSCAR protocol)

The AIM/OSCAR protocol has been reverse engineered, and the information here is taken from [18]. The AIM protocol consists of AIM commands. The FLAP protocol is the underlying protocol used by AIM channels. One or more FLAP packets containing an AIM command are embedded in a TCP packet. The FLAP payload is usually a SNAC packet. The FLAP header does not contain sensitive information. Sensitive information contained in SNAC packets consists of Screen name lengths, message lengths, warning level of users, user's "class," member since date, "on since," idle time, and messages.

IV. Information leakage from combinations of fields

In this section, we study the extent of information leakage if combinations of fields are known.

A. Identifying a user

Knowing any combination of multiple fields in a single packet is not enough to pinpoint a user with a significant amount of certainty.

B. Identifying an operating system

The values of IP and transport-layer header fields, such as incorrect checksums, TCP flags, can narrow down the possibilities for which operating system is running on a host, due to incorrectly implemented protocols or lack of explicit specifications in the protocols. Any combination of more than one of these fields has a higher probability of being incorrect/unique than just one field by itself.

C. Identifying an application

The IP protocol number reveals the transport protocol being used. The IP header length subtracted from the IP total length gives the payload size. Individually, each of these pieces of information can only narrow down the list of possible applications. If more than one of these properties of the packet is known, it becomes easier to identify the application with more certainty. The IP port number is associated with a specific application protocol, if the application is well-known.

V. Information leakage from a packet stream

In this section, we evaluate the extent of information leakage if a packet stream (packets in a connection) is known.

A. Identifying a user

A user may have behavior patterns that are unique, such as using more than one or the same mix of applications using the Internet simultaneously. The applications that an individual uses are not random, and generally change very slowly over time. The differences in frequency of applications being used can unique to a user. The frequency of sending or receiving packets, amount of data sent per connection, time of day messages are sent, aggregate amount of data sent and received, either overall or for specific sources/destinations, measured over a long enough period of time, can be accurate enough to identify a user.

B. Identifying an operating system

The pattern in initial sequence numbers chosen for new connections as well as how sequence numbers for subsequent packets are incremented can be used to fingerprint an OS. Certain applications can only be run on particular families of operating systems. Certain versions of applications may only run properly on certain versions of an operating system family. The typical mix of types of applications used on different operating systems could be unique. The predictability of sequence numbers clearly distinguishes Windows from Unix-based operating systems. If active OS fingerprinting was taking place when the trace was taken (active

inference is when the attacker has inserted packets into the network while the trace was taken to gain knowledge of how the original trace was mapped to the anonymized trace), this can reveal the operating system from its response to unusual packets.

C. Identifying an application

An application may have a signature based on the frequency and length of packets sent in both directions. Whether the unknown application uses a client-server model or a peer to peer model can narrow down which application is being run. If it is known that the destination is a web server, we can infer that the packet is probably an HTTP packet.

By comparing the number of packets going back and forth between pairs of nodes, Web servers can be distinguished from clients. Popular web servers could be uniquely identified.

VI. Anonymizing packet streams

We need to go beyond transforming header fields and data transformations of filtering text in payloads by performing transformations on streams. These include (i) introducing random noise and (ii) mixers for reordering packet streams. The transformations increase privacy by preventing passive traffic analysis attacks and active known-text attacks (passive and active attacks are further discussed in Section VI-B). They are, however, limited by the need to preserve properties of the trace which must be present for a security mechanism to function properly. The properties of the trace we choose to maintain depend on how a particular security mechanism (e.g., IDS sensor) works and on the properties of attacks.

At the level of a single stream of many packets, consisting of packets with the same IP source and destination addresses and port numbers, we can introduce random noise into certain properties by a percentage of the original amount. In our tool, a default of 5% is given. Random noise can be introduced into the spacing between packets, as long as we keep the average spacing of the stream constant. The packet size can also have random noise added to it, keeping the distribution of packet sizes constant. The duration of the connection can be changed by a random amount, such that the average connection length in the trace is constant. If the noise is within a sufficiently small percentage of the original value, the distributions of values should remain approximately the same.

A. Stream reordering

We define a stream as a sequence of packets in one flow (e.g., TCP connection). At the level of many streams, we can reorder streams by swapping streams that are close to each other. We determine which streams can be swapped with each other by extracting the streams and the order in which they appear, and dividing these streams into groups of a user-defined number. First, the earliest start time of all streams in the group are recorded. If any stream starts before the previous group ended, that stream is shifted down until it no longer overlaps. Next the latest end time of all streams in the group is recorded. The streams in the group are then randomly paired for swapping, but if it is found that swapping will cause one of the streams to exceed the group's time bounds, the swap is not carried out. Swapping here means that each stream is shifted such that its new start time is equal to the old start time of the other stream. This ensures that in the small picture, patterns are changed, but in the big picture the trace exhibits the same characteristics as the original.

Since it is possible for a stream to be present for most of the duration of the trace, we will not process very long streams in our swapping algorithm. If we see numerous TCP streams of less than three packets close together, this indicates that scanning is taking place. Our reordering will not disrupt this pattern of many incomplete connections close together as long as groups are sufficiently small since it is likely that all these streams will be in the same group, and will only be swapped with each other. Our swapping will reorder these streams such that an active attacker who inserted a scan when the trace was taken cannot know the correct IP anonymization mappings for the addresses they scanned.

Another property of interactions between many streams is which streams are interleaved together, the distribution of how many streams are interleaved together, and the amount of overlap if the streams do not start and end at the same time. Changing the interleaved properties of the trace is more complex than reordering streams, since many streams can be interleaved together, and doing this would also change the sequence of streams. This effect is not desirable because it can disrupt the groupings described above. Groupings must be preserved to ensure that the trace properties remain constant at the macroscopic level, and also ensure that the timestamps for each packet in the group will always be smaller than those for the next group, so that each group can be sorted and printed out before the next group is processed, reducing the amount of memory used at any one time. The stream reordering process can change which streams are interleaved together to a certain extent, which depends on the size of the groups.

If some TCP packets appear in incorrect order in the trace, they are put in order and other packets, either in a different stream or the same stream, can be put out of order.

Traffic analysis attacks can also be prevented by merging two or more traces into a single trace, while keeping the properties of an equivalent output trace taken from one trace to a sufficient extent. This is more complex, since there are issues with merging traces that need to be taken into account. There are several ways in which this could be implemented. A composite trace can be created from traces taken from the same network on different days at different times. However, we must be careful to ensure that the number of connections for a server does not exceed its maximum capacity. Another method would be to merge a certain percentage of traffic from a trace with the remaining percentage of traffic taken from another trace from a different network or the same network, making sure mix of applications, bandwidth remains the same. Servers and hosts from each trace should not be

combined in the final trace to avoid seeing in the output trace a server that is operating normally but handling more requests than it is actually capable of handling in reality without Denial of Service.

B. Inference attacks on complete traces

Inferring private or sensitive information from the anonymized trace can be done through either passive attacks, where the attacker's only source of information is the anonymized trace, and active attacks, where the attacker has inserted packets into the network while the trace was taken to gain knowledge of how the original trace was mapped to the anonymized trace.

In an active attack, since IP addresses are anonymized, the attacker would have to know some other distinguishing characteristic about the traffic they inserted in order to identify the inserted packets in the transformed trace, such as duration of the connection, strings in payloads, unusual values for header fields, number of packets, total amount of data sent, packet sizes, rate of sending packets, whether this stream overlaps with another stream at a certain time, number of connections between the same pair of IP addresses. In our trace transformations, it is possible to change these properties for all streams, but only within a certain range to ensure that the trace remains of high fidelity. However, characteristics that enable us to identify an attack should not be removed. Hence if an attack is inserted into the trace, it will be visible in the transformed trace. For example, if an attacker inserts scanning into the trace, they could find out the mappings between anonymized and real IP addresses for all the addresses that they scanned. To prevent this, we can randomly reorder packets in such an attack so that the attacker would only get incorrect information about the mappings.

C. Trace characteristics

Metrics to quantify the fidelity of the transformed trace with respect to attack properties are needed to quantify how well we have succeeded in our goal of preserving the properties of network traffic. We need to preserve both behavior that appears similar to attacks which is needed by misuse intrusion detection, as well as any other unusual behavior which is needed by anomaly intrusion detection. Malformed headers and incomplete TCP connections must be preserved. Packet sizes must not be changed by too large a percentage of the original packet sizes. We also need to preserve characteristics of the network which could affect the performance or operation of an IDS, which includes the mix of applications used over the network, the distribution of packet sizes for each stream, average connection length of streams, total number of hosts, total number of connections, average bandwidth usage. These trace properties that need to be preserved are used in our trace transformations to ensure that they are preserved.

D. Design methodology

We are developing a tool to perform the packet header anonymizations and trace transformations described above. Currently our tool supports user-defined removal of packet fields for IP, TCP, and UDP header fields. We use CryptoPAN [19] for prefix-preserving IP address anonymization. Mapping of fields to new values, reassembling application data, and applying the "filter-in" anonymization policy to the application payload [3] will be added.

An important decision we made is that we must save fragment properties, in addition to parsing application-level data, which involves reassembling fragments. We can resolve this by saving both fragment headers and application data for each stream. After the application data has been anonymized, which may involve changing the size of the data, it can then be broken up and put back into the corresponding fragment. Since the size may have changed, fragment headers that are not needed can be discarded, or new fragments can be created. Creating new fragments would not be too difficult since fragments from the same stream usually share the same properties. This would have some effect on the statistics of certain properties of the trace, but as long as the change in data sizes is not too large the effect should not be too great.

Streams are extracted according to the order in which they first appear. The streams are divided into "groups" of a certain number of streams (corresponds to the variable "swaparea"). First, we check if any stream in the group starts before the previous group ended. If so, that stream is shifted down until it no longer overlaps. Next we record the start time of the 1st stream. This is the group's start time. Next the latest end time of all streams in the group is recorded. The streams in the group are randomly paired for swapping, but if the swapping will cause one of the streams to exceed the group's end time, the swap is not done. Swapping means that 2 streams exchange start times.

Observe that very long streams should not be processed in the swapping algorithm above because if a stream is as long as the entire trace, it can never be reordered. Also, this would mean all packets not in the group have to be shifted until after the long stream ends, making the time for the trace much longer than before.

Also observe that many TCP streams of less than 3 packets should be in the same group, because they need to be kept close together but also reordered among themselves so that an attacker cannot obtain mappings between real and anonymized IP addresses.

The trace transformation of adding random noise to spacing between packets has been implemented. Spacing between packets is recalculated to be the average spacing for that stream plus or minus random noise within a user-specified percentage of the average spacing.

Other simple transformations we will incorporate include transforming:

1. Packet size (keeping distribution of packet sizes the same)
2. Duration of connection (keeping distribution of connection lengths the same)
3. Packet order (if TCP packets in incorrect order appear in the trace, put them in order and make other packets out of order)

4. Stream interleaving (which streams/how many streams are interleaved together)
5. Trace synthesis (creating a composite trace from traces taken from the same network on different days at different times, ensuring that the number of connections for a server does not exceed its maximum)
6. Merging traffic from another trace from a different network (making sure mix of applications, bandwidth remains the same) Servers, hosts from each trace should not be combined in the final trace.

We give example trace transformation results from our preliminary experiments in the appendix.

VII. Related work

Sections I and II discussed work on traces for security evaluation. In this section, we discuss general work on firewall and IDS testing.

Firewall/IDS testing has different goals, including determining if the firewall is a correct implementation of the firewall security policy, how well the firewall resists particular types of attacks, if leakage occurs in the security perimeter created by a firewall, if the logging capability is adequate, if the firewall has the ability to send alarms, and if the firewall can hide information and addresses from the internal network it protects [20], [21]. In this section, we survey prior work that has examined firewalls and firewall testing. The basic design of a firewall and sample firewall algorithms are discussed in [22], [23], [24], [25], [26], [27]. Schuba [28] formalizes firewalls using hierarchical colored Petri nets. Bellovin [29] recently proposed a distributed approach to Internet firewalls. In this approach, personal firewalls are installed at the hosts themselves. The advantages of this distributed approach include detection of attacks within the internal network, detection of more application level attacks, and speeding up firewall functions.

In [30], Haeni describes a methodology to perform firewall penetration testing. The testing steps include indirect information collection, direct information collection, attacks from outside, and attacks from inside. Attack approaches are based on the type of firewall. For packet filtering firewalls, the attacks include blind IP spoofing, non-blind IP spoofing, source porting and source routing. For application level firewalls, the attacks are on bad security policy, poorly implemented policies, SOCKs incorrectly configured, brute force attacks, and enabled services/ports.

Another firewall testing methodology is presented in [31]. The motivation behind this work is that field testing is currently performed using simple checklists of vulnerabilities without taking into account the particular topology and configuration of the firewall target operational environment. A firewall testing methodology is proposed, based on a formal model of networks that allows the test engineer to model the network environment of the firewall system; to prove formally that the topology of the network verifies the sufficient conditions for protection against attacks; and to build test cases to verify that protections are actually in place.

In [32], firewall security and performance relationships are explored. Experiments are conducted to classify firewall security into seven different levels and to quantify their performance effects. These firewall security levels are formulated, designed, implemented and tested under an experimental environment in which all tests are evaluated and compared. Based on the test results, the effects of the various firewall security levels on system performance with respect to transaction time and latency are measured and analyzed. It is interesting to note that the intuitive belief that more security would result in degraded performance does not always hold.

In addition, the Internet engineering task force (IETF) has examined Internet firewalls. RFC 2647 [33] extends the terminology used for benchmarking routers and switches with definitions specific to firewalls. Forwarding rate and connection-oriented measurements are the primary metrics used in the RFC. RFC 2979 [34] defines behavioral characteristics and interoperability requirements for Internet firewalls. The RFC observes that firewall behavior is often either unspecified or under-specified, and this lack of specificity often causes problems in practice. Requirements specification makes the behavior of firewalls more consistent across implementations and in line with accepted protocol practices. The Lumeta firewall analyzer [35] is an easy-to-use tool that checks a firewall security policy by simulating incoming and outgoing packets.

We have described a data-flow-based methodology for analyzing vulnerabilities in Internet firewalls in [36], [37]. We have examined firewall internals, and cross referenced each firewall operation with causes and effects of weaknesses in that operation. We have analyzed twenty reported problems with available firewalls. The result of our analysis is a set of matrices that illustrate the distribution of firewall vulnerability causes and effects over firewall operations. These matrices are useful in avoiding and detecting unforeseen problems during both firewall implementation and firewall testing. Two case studies of Firewall-1 and Raptor illustrate our methodology.

VIII. Conclusions and Future Work

This work is a preliminary investigation into the problem of obtaining anonymized traces that can be made available to the security research community. Such traces are crucial for security experiments, e.g., those that can be conducted on the DETER/EMIST testbed project [38]. A number of efforts are now underway at the U.S. Department of Homeland Security and the National Science Foundation to facilitate such efforts.

We are currently extending this work in two directions. First, we are incorporating more sophisticated randomization techniques to enhance privacy, and examining the assignment of default configuration values. Second, we are investigating ways of merging normal background traces with attacks when we have new attack tools.

REFERENCES

[1] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman, "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation," in *Proceedings of the DARPA Information survivability conference and exposition (DISCEX)*, January 2000, Web site at: http://ideval.ll.mit.edu/intro-html-dir/.

[2] J. McHugh, "Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Transactions on Information and System Security*, vol. 3, no. 4, pp. 262–294, November 2000.

[3] Ruoming Pang and Vern Paxson, "A high-level programming environment for packet trace anonymization and transformation," in *Proc. of ACM SIGCOMM*, August 2003, http://www.cs.princeton.edu/∼rpang/bro-anonymizer-sigcomm03.pdf.

[4] I. Dubrawsky, "Portsentry for attack detection," http://www.securityfocus.com/infocus/1580, May 2002.

[5] O. Arkin, "Nmap hackers: TOSing OSs out of the window / fingerprinting windows 2000 with ICMP," http://lists.insecure.org/lists/nmap-hackers/2000/Jul-Sep/0037.html, August 2000.

[6] O. Arkin, "Nmap hackers: Precedence field value in ICMP error messages with linux," http://lists.insecure.org/lists/nmap-hackers/2000/Oct-Dec/0009.html, October 2000.

[7] "RING: An opensource OS fingerprinting tool," http://www.securiteam.com/tools/5YP0L006UC.html, April 2002.

[8] "Bugtraq:pof - passive OS fingerprinting tool," http://lists.insecure.org/lists/bugtraq/2000/Jun/0141.html, June 2000.

[9] "Remote OS detection via TCP/IP fingerprinting," http://www.insecure.org/nmap/nmap-fingerprinting-article.html, June 2002.

[10] O. Arkin and F. Yarochkin, "Phrack volume 11, issue 57," http://www.phrack.org/show.php?p=57&a=7, August 2001.

[11] "Tiny fragment attack," http://www.zvon.org/tmRFC/RFC1858/Output/chapter3.html, 2003.

[12] "Security considerations for IP fragment filtering," RFC 1858, October 1995.

[13] M. Olsson, "BugTraq archive: Analysis of jolt2.c," http://www.securityfocus.com/archive/1/62011/2002-11-29/2002-12-05/0.

[14] "Symantec attack signatures," http://securityresponse.symantec.com/avcenter/nis_ids/.

[15] "FreeBSD TCP RST denial of service vulnerability," http://ciac.llnl.gov/ciac/bulletins/j-008.shtml, October 1998.

[16] J. Oikarinen and D. Reed, "Internet relay chat protocol," RFC 1459, www.ietf.org.

[17] H. Isaksson, "Version 5 of the icq protocol," http://www.algonet.se/∼henisak/icq/icqv5.html, April 2001.

[18] A. Fritzler, "Unofficial AIM/OSCAR protocol specification," http://aimdoc.sourceforge.net/OSCARdoc/, April 2000.

[19] Jun Xu, Jinliang Fan, Mostafa Ammar, and Sue B. Moon, "Prefix-preserving ip address anonymization," in *Proc. of International Conference on Network Protocols (ICNP)*, 2002, pp. 280–289, citeseer.nj.nec.com/462352.html.

[20] E. Schultz, "How to perform effective firewall testing," *Computer Security Journal*, vol. 12, no. 1, pp. 47–54, 1996.

[21] E. Schultz, "When firewalls fail: lessons learned from firewall testing," *Network Security*, pp. 8–11, Feb 1997.

[22] William R. Cheswick and Steven M. Bellovin, *Firewalls and Internet Security: repelling the wily hacker*, Addison-Wesley, Reading, Massachusetts, 1994.

[23] Bill Cheswick, "The design of a secure Internet gateway," in *USENIX*, Anaheim, California, June 1990, pp. 233–237.

[24] Kathryn M. Walker and Linda Croswhite Cavanaugh, *Computer Security Policies and SunScreen Firewalls*, Prentice Hall, Upper Saddle River, New Jersey, 1998.

[25] Linda McCarthy, *Intranet Security*, Prentice Hall, Upper Saddle River, New Jersey, 1998.

[26] Rita C. Summers, *Secure Computing: Threats and Safeguards*, McGraw-Hill, 1997.

[27] Gene Spafford and Simson Garfinkel, *Practical Unix and Internet Security*, O'Reilly & Associates, Inc, second edition, 1996.

[28] Christoph L. Schuba, *On the Modeling, Design and Implementation of Firewall Technology*, Ph.D. thesis, Department of Computer Sciences, Purdue University, December 1997, https://www.cerias.purdue.edu/techreports-ssl/public/97-07.pdf.

[29] S. Ioannidis, A. Keromytis, S. Bellovin, and J. Smith, "Implementing a distributed firewall," in *Proceedings of the ACM CCS*, November 2000.

[30] Reto E. Haeni, "Firewall penetration testing," http://www.seas.gwu.edu/∼reto/papers/firewall.pdf, 1997.

[31] Giovanni Vigna, "A formal model for firewall testing," http://www2.elet.polimi.it/pub/data/ Giovanni.Vigna/www_docs/pub/fwtest.ps.gz.

[32] M. R. Lyu and L. K. Y. Lau, "Firewall security: policies, testing and performance evaluation," in *Proceedings of the COMSAC*. IEEE Computer Society, 2000, pp. 116–21.

[33] D. Newman, "Benchmarking terminology for firewall performance," Request for Comments 2647, ftp://ftp.isi.edu/in-notes/rfc2647.txt, Aug. 1999.

[34] N. Freed, "Behavior of and requirements for internet firewalls," Request for Comments 2979, http://search.ietf.org/rfc/rfc2979.txt, Oct. 2000.

[35] Avishai Wool, "Architecting the lumeta firewall analyzer," in *Proceedings of the 10th USENIX Security Symposium*, 2001, pp. 85–97, http://www.usenix.org/events/sec01/wool.html.

[36] M. Frantzen, F. Kerschbaum, E. Schultz, and S. Fahmy, "A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals," *Computers and Security*, vol. 20, no. 3, pp. 263–270, May 2001.

[37] Seny Kamara, Sonia Fahmy, Eugene Schultz, Florian Kerschbaum, and Michael Frantzen, "Analysis of vulnerabilities in internet firewalls," *Computers and Security*, vol. 22, no. 3, pp. 214–232, April 2003, http://www.cs.purdue.edu/homes/fahmy/.

[38] R. Bajcsy et. al, "Cyber defense technology networking and evaluation," *Communications of the ACM*, vol. 47, no. 3, pp. 58–61, March 2004.

APPENDIX

I. SAMPLE RESULTS

A. Anonymization

The following is an example where fields in an extended tcpdump trace are removed (actually replaced by "=") based upon a user-specified configuration file.

```
Original:

1057765967.444373 131.179.187.107 > 131.179.192.136: ip 4 5 10 116
          59009 000 0 64 6 d097 tcp 22 37651
          1383826984:1383827048(64) - 8 00 P 8576 8044 -
          <nop,nop,timestamp 75080402 41636870>

1057765967.444552 131.179.192.136 > 131.179.187.107: ip 4 5 10 52
          40927 000 0 63 6 187a tcp 37651 22 1364384983:1364384983(0)
          - 8 00 . 8880 38ad - <nop,nop,timestamp 41636871 75080402>

1057765967.474775 802.1d config 80bb.00:06:52:23:b5:00.2154 root
          8000.00:d0:03:72:b8:ba pathcost 8 age 2 max 20 hello 2
          fdelay 15
```

```
1057765969.254058 131.179.187.1 > 255.255.255.255: ip 4 5 c0 532 0 000
          0 2 17 7765 udp 520 520 504 a448

1057765969.254060 131.179.187.1 > 255.255.255.255: ip 4 5 c0 112 0 000
          0 2 17 7909 udp 520 520 84 f192

1057765969.474621 802.1d config 80bb.00:06:52:23:b5:00.2154 root
          8000.00:d0:03:72:b8:ba pathcost 8 age 2 max 20 hello 2
          fdelay 15

1057765971.161852 131.179.187.107 > 164.67.62.194: ip 4 5 00 76 0 000
          0 64 17 187d udp 123 123 48 4a3c

1057765971.162547 164.67.62.194 > 131.179.187.107: ip 4 5 10 76 2 000
          0 59 17 1d6b udp 123 123 48 be05

1057765971.477579 802.1d config 80bb.00:06:52:23:b5:00.2154 root
          8000.00:d0:03:72:b8:ba pathcost 8 age 2 max 20 hello 2
          fdelay 15

1057765973.478741 802.1d config 80bb.00:06:52:23:b5:00.2154 root
          8000.00:d0:03:72:b8:ba pathcost 8 age 2 max 20 hello 2
          fdelay 15

1057765975.478741 802.1d config 80bb.00:06:52:23:b5:00.2154 root
          8000.00:d0:03:72:b8:ba pathcost 8 age 2 max 20 hello 2
          fdelay 15

1057765977.483530 802.1d config 80bb.00:06:52:23:b5:00.2154 root
          8000.00:d0:03:72:b8:ba pathcost 8 age 2 max 20 hello 2
          fdelay 15

Anonymized:

1057765952.000000 132.82.35.51 132.82.64.137 ip 4 = = = = 000 0 64 6 =
          tcp = = = - 8 00 P = = = =

0.444552 132.82.64.137 132.82.35.51 ip 4 = = = = 000 0 63 6 = tcp = =
  = - 8 00 . = = = =

0.474775 802.1d config 80bb.00:06:52:23:b5:00.2154 root 0.474775
  802.1d config 80bb.00:06:52:23:b5:00.2154 root
  8000.00:d0:03:72:b8:ba pathcost 8 age 2 max 20 hello 2 fdelay 15

2.254058 132.82.35.126 206.120.97.255 ip 4 = = = = 000 0 2 17 = udp =
  = 504 =

2.254060 132.82.35.126 206.120.97.255 ip 4 = = = = 000 0 2 17 = udp =
  = 84 =

2.474621 802.1d config 80bb.00:06:52:23:b5:00.2154 root 2.474621
  802.1d config 80bb.00:06:52:23:b5:00.2154 root
  8000.00:d0:03:72:b8:ba pathcost 8 age 2 max 20 hello 2 fdelay 15

4.161852 132.82.35.51 163.66.192.241 ip 4 = = = = 000 0 64 17 = udp =
  = 48 =

4.162547 163.66.192.241 132.82.35.51 ip 4 = = = = 000 0 59 17 = udp =
  = 48 =

4.477579 802.1d config 80bb.00:06:52:23:b5:00.2154 root 4.477579
  802.1d config 80bb.00:06:52:23:b5:00.2154 root
  8000.00:d0:03:72:b8:ba pathcost 8 age 2 max 20 hello 2 fdelay 15

6.478741 802.1d config 80bb.00:06:52:23:b5:00.2154 root 6.478741
  802.1d config 80bb.00:06:52:23:b5:00.2154 root
  8000.00:d0:03:72:b8:ba pathcost 8 age 2 max 20 hello 2 fdelay 15

8.478741 802.1d config 80bb.00:06:52:23:b5:00.2154 root 8.478741
  802.1d config 80bb.00:06:52:23:b5:00.2154 root
  8000.00:d0:03:72:b8:ba pathcost 8 age 2 max 20 hello 2 fdelay 15

10.483530 802.1d config 80bb.00:06:52:23:b5:00.2154 root 10.483530
   802.1d config 80bb.00:06:52:23:b5:00.2154 root
   8000.00:d0:03:72:b8:ba pathcost 8 age 2 max 20 hello 2 fdelay 15
```

```
Configuration file:
y       IP Header Length
y       IP Type of Service
y       IP Total Length
y       IP Identification
n       IP Flags
n       IP Fragment Offset
n       IP Time to Live
n       IP Protocol No.
y       IP Header Checksum
y       Source Port
y       Destination Port
y       TCP Sequence No.
n       TCP Ack No.
n       TCP Data Offset
n       TCP Reserved Field
n       TCP Flags
y       TCP Window Size
y       TCP Checksum
y       TCP Urgent Pointer
y       TCP Options
n       UDP Length
y       UDP Checksum
```

B. Stream Reordering

In this section, we show an example of reordering streams and introducing noise in packet spacing. In the “report” section below, reordering can be seen in the third and fourth columns, and noise is seen in the fi fth column.

```
Original:

0       0.444373        132.82.35.51    132.82.64.137   ip      4       5
        10      116     =       000     0       =       6       =       tcp
        22      37651   1383826984:1383827048(64)       -       =       =
        AP      =       =       =       <nop,nop,timestamp75080402,41636870,>

1       0.444552        132.82.64.137   132.82.35.51    ip      4       5
        10      52      =       000     0       =       6       =       tcp
        37651   22      1364384983:1364384983(0)        -       =       =
        A       =       =       =       <nop,nop,timestamp41636871,75080402,>

2       2.254058        132.82.35.126   206.120.97.255  ip      4       5
        c0      532     =       000     0       =       17      =       udp
        =       520     504     a448
3       2.254060        132.82.35.126   206.120.97.255  ip      4       5
        c0      112     =       000     0       =       17      =       udp
        =       520     84      f192
4       4.161852        132.82.35.51    163.66.192.241  ip      4       5
        00      76      =       000     0       =       17      =       udp
        =       123     48      4a3c
5       4.162547        163.66.192.241  132.82.35.51    ip      4       5
        10      76      =       000     0       =       17      =       udp
        =       123     48      be05

Transformed:


0.444552        0.444373        132.82.35.51    132.82.64.137   ip      4
        5       10      116     =       000     0       =       6       =
        tcp     22      37651   1383826984:1383827048(64)       -       =
        =       AP      =       =       =       <nop,nop,timestamp75080402,4163
6870,>

0.444373        0.444552        132.82.64.137   132.82.35.51    ip      4
        5       10      52      =       000     0       =       6       =
        tcp     37651   22      1364384983:1364384983(0)        -       =
        =       A       =       =       =       <nop,nop,timestamp41636871,7508
0402,>

2.254060        2.254058        132.82.35.126   206.120.97.255  ip      4
        5       c0      532     =       000     0       =       17      =
        udp     =       520     504     a448
```

```
2.254058        2.254060        132.82.35.126   206.120.97.255  ip      4
        5       c0      112     =       000     0       =       17      =
        udp     =       520     84      f192

4.162547        4.161852        132.82.35.51    163.66.192.241  ip      4
        5       00      76      =       000     0       =       17      =
        udp     =       123     48      4a3c

4.161852        4.162547        163.66.192.241  132.82.35.51    ip      4
        5       10      76      =       000     0       =       17      =
        udp     =       123     48      be05


Report:

0.444373        0.444373        0       1       0.000179        0.000000

0.444552        0.444552        1       0       -0.000179       0.000000

2.254058        2.254058        2       3       0.000002        0.000000

2.254060        2.254060        3       2       -0.000002       0.000000

4.161852        4.161852        4       5       0.000695        0.000000

4.162547        4.162547        5       4       -0.000695       0.000000
```