Analysis of Session Flow Information: First Experiment

Todd Heberlein

Our Environment Aware security project explores how detailed knowledge of an organization's environment (e.g., network resources, file configurations, common traffic patterns, and trust and dependency relationships between objects) and the threats the organization faces can be used to optimize system administrators' efforts to secure their site. One source of data we can use to build a model of an organization's network and threat environment is network audit trails, records of all network activity that a number of organizations currently collect. We have developed a series of experiments to determine how we can use this existing data in our Environment Aware effort, and this report documents this first of those experiments.

Table of Contents

List of Figures

List of Tables

1 Introduction

This report describes the first in a series of experiments involving network-based session records. Traditional network security sensors (e.g., intrusion detection systems such as RealSecure and Snort) primarily look for known attack patterns and only report attacks. However, a number of organizations such as the Air Force and the Department of Energy use sensors that create session records for all network activity observed on part of their networks, typically on the border between the site and the rest of the Internet.

Our Environment Aware security project explores how additional knowledge of an organization's environment (e.g., network resources, file configurations, common traffic patterns, and trust and dependency relationships between objects) and the threats the organization faces can be used to optimize system administrators' efforts to secure their site. With a model of the environment available to us we can apply algorithms to search the model and identify optimal changes to network control surfaces (e.g., firewall rules, file permissions, application of patches, placement of sensors) that provide the greatest security advantage with respect to a known or hypothesized threat.

In an ideal world, all the relevant data describing the full details of the environment would magically be available for analysis, but in practice we must make due, at least initially, with data that is readily available. One of the richest sources of data available to build our environment models is the session records collected by some DOD and DOE organizations, and we have developed a series of experiments to explore how we can use and improve upon this data. This report describes our first experiment to explore how session records can be applied to our Environment Aware security project.

1.1 The Case for Session Records

The UC Davis Network Security Monitor (NSM), originally developed in 1991, created a log file containing session records for all TCP/IP and UDP/IP network sessions. This data essentially served as an audit trail for network activity and was analyzed by other programs. The UC Davis NSM served as the foundation for the Air Force's ASIM sensor, the Defense Information Systems Agency's JIDS sensor, and the Department of Energy's NID sensor. But in the mid to late 1990s organizations started developing and deploying network-based intrusion detection sensors that eschewed the network audit trail approach in favor of just report specific attack reports.

Network session records (network audit trails) can still provide additional information and, if acted on, additional capability that simple attack-reporting IDS sensors cannot provide. Some of the additional information and capability we believe we can exploit in our Environment Aware effort the following:

- **Identify active servers in the network.**
- **Identify typical traffic patterns.**
- **Develop firewall filter rules.** The known traffic patterns provided by the session vectors can be used to develop firewall rules that support all existing traffic patterns but can block unexpected traffic patterns. These rules can aid the system administrators in the day-to-day tuning of their firewalls, or they can be applied during an emergency such as when a new worm is circulating.
- **More quickly and accurately identify attacks and probes against the network.** With information about known servers and typical traffic patterns, each connection can be

weighted with an anomaly score (does it fit with normal traffic patterns) and a legitimacy score (when was there an active server running on the target of a connection), and this weighting approach should produce better results than the more traditional approach of simply counting sessions.

- **Detect variants of a known attack.** Traditional network based sensors can detect known attacks, but they often are unable to distinguish one variant of an attack from another. Combining traditional IDS alerts with the network session records that record statistical behavior of a session, we should be able to distinguish variants in known attacks.
- **Detect unknown attacks.** If a new attack is tried against more than one server and the attack is not highly polymorphic, statistical clustering algorithms should be able to identify similar content going to multiple unrelated servers.
- **Determine whether an attack was successful.** For known attacks an organization can use traditional vulnerability scanners (e.g., ISS and Nessus) to know which attacks are likely to have succeeded. For relatively new attacks for which there is no vulnerability information available, this approach does not work. However, session records may be able to identify clusters in server responses (e.g., for an attack against Microsoft's IIS web server, an Apache web server may respond one way, a patched IIS server respond another way, and a vulnerable IIS web server respond a third way) providing the system administrator with a triage capability.

1.2 Approach and Goals for Our Experiment

Since some organizations already have their own session record (network audit trail) capability, we are not trying to develop a system (or set of systems) to replace their existing infrastructure. Instead our goals are to identify and quantify how session records can add value to securing an organization and how small changes to the session record formats can enhance the value of the information and increase capability.

Our approach is to carry out a rolling series of experiments and document those experiments. Each experiment will test certain basic hypotheses and assumptions and identify areas for future improvement or new hypotheses that need to be tested in future experiments. Our goals for this first round of experiments include:

- **Validate our session record software.** This is the experiment's *primary* goal since future experiments will build on this software. Since we do not have the session record software used by organizations such as the Air Force and the Department of Energy, we needed to develop our own. While we starting with some existing software we had developed earlier, we (1) are running on a new processor (Intel Xeon) and operating system (new version of Red Hat Linux), (2) added some new features, and (3) and are deploying on a faster and more complex network.
- **Validate some of the basic assumptions of the approach.** While the experiment will only test a limited set of data, we hope to identify some attacks in that data, and we will determine if there is clustering behavior in that data.
- **Determine an initial set of questions we would like to ask about the data.** With recorded session records in hand, we plan to ask an initial set of questions of that data, and we suspect the answers to those questions will lead us to new questions. The questions will identify where and to what extent session records can add value to securing an organization.

- **Develop an initial set of post-processing tools to answer the questions.** To answer the questions about the data set, we will need to develop algorithms and logic (code) process the data. These tools will be used in future experiments.
- **Identify improvements to session records in general and the software used in our experiments in particular for future experiments.**

1.3 Roadmap to the Report

Section 2 briefly describes the software, the development environment, experiment environment, and a summary of the observed data rates. Section 3 describes the session record model we developed for this experiment. Section 4 describes our use of both sequence numbers and observed payload bytes for tracking the amount of data one host sent to another. Section 5 summarizes a technical issue we encountered with respect to the interaction of TCP control flags, TCP sequence numbers, and the order of session shutdown.

Sections 6 and 7 summarize the analysis of a small subset of session records tracked during this experiment. Section 6 discusses how we identified attacking hosts in the data set, and Section 7 dives into detail on the session behavior of three attacking hosts.

Finally, Section 8 summarizes the results of our experiments, including a list of "lessons learned."

2 Overview of the Experimental Environment

Our software was initially developed on a Macintosh PowerPC running the Mac OS X 10.2 (Jaguar) operating system, and for the experiments we re-compiled and ran it on a Dell Xeon-based computer using Red Hat Linux operating system. Most of the post-analysis work was performed on the Macintosh as well. The Dell Xeon system monitored multiple fiber-based Ethernet networks, and the organization had two Class B networks and a handful of Class C networks.

We ran the software for several days, and there were no crashes or apparent memory leaks. For the post-mortem analysis described in the rest of this report, we only analyzed approximately four hours and 20 minutes worth of data, because this is all that fit onto a single CD. For those four hours we had the basic following statistics: 9,582,680 sessions were identified and tracked – a rate of approximately 526 new sessions per second.

3 The Session Flow Model

While a number of organizations such as the Air Force Computer Emergency Research Team (AFCERT) and the Department of Energy's Cooperative Protection Program (CPP) have their own representation for a network sessions, we chose to use our own model developed as part of the Network Monitoring Framework (NMF) and Network Radar research efforts funded in part by the United States Air Force Research Laboratories (AFRL) and the Defense Advanced Research Projects Agency (DARPA). We believe our network session model is sufficiently close to existing models so that our analysis can be mapped to their systems, and since we control the code, we can make minor adjustments or extensions to the network session model and determine how these changes affect the types and quality of analyses that can be performed.

For this round of experiments we only analyzed TCP/IP traffic, and a network session is defined to be a single TCP/IP connection or connection attempt. Our network model captured 25 features associated with each network session.

- **Session ID** – Every network session is given an ID, which is simply a number that increments with each new session.
- **Server Identification Confidence** – The NMF code attempts to determine which host is the client (the host that initiates the connection) and which host is the server (the host that accepts/rejects the connection request). The code uses TCP control flags, TCP port numbers, and the order of packets to determine which host in a connection is the client and which is the server, and because the not all the information is always available to the sensor, the code indicates its confidence that the server (and client) have been correctly identified by assigning a confidence value of 0 (low), 1 (medium), or 2 (high).
- **Client IP Address** – The client's IPv4 address for the session.
- **Client TCP Port** – The client's TCP port number for the session.
- **Server IP Address** – The server's IPv4 address for the session.
- **Server TCP Port** – The server's TCP port number of the session.
- **Transient Server Flag** – Some services (e.g., FTP) set up transient, one-time only servers to exchange data. If the sensor believes this session is to one of the transient servers, this flag is set to 1.
- **Inbound Flag** – A user can define a set of IP addresses he wants to protect. If he does define a protection zone, then this flag is set to 1 if the connection comes from outside the protected zone into the protected zone.
- **Full Analysis Flag** – The NMF code can layer in additional analysis based on certain conditions (e.g., in the session in inbound or not). If optional analysis is performed, this flag is set to 1.
- **Client First TCP Control Flags** – The TCP control flags in the first packet sent by the client.
- **Server First TCP Control Flags** – The TCP control flags in the first packet sent by the server.
- **Client Composite TCP Control Flags** – The set of all control flags set in all packets sent by the client.
- **Server Composite TCP Control Flags** – The set of all control flags set in all packets sent by the server.
- **Client Payload Bytes** – The number of bytes in the TCP payload section for all packets sent by the client and observed by the sensor. No effort is made to identify packet loss or retransmissions.
- **Server Payload Bytes** – The number of bytes in the TCP payload section for all packets sent by the server and observed by the sensor. No effort is made to identify packet loss or retransmissions.
- **Client Packets** – The number of packets sent by the client.
- **Server Packets** – The number of packets sent by the server.
- **Client First TCP Sequence Number** – The TCP sequence number of the first packet sent by the client. If this is for the SYN packet, this number is the host's initial sequence number (ISN).

- **Server First TCP Sequence Number** – The TCP sequence number of the first packet sent by the server. If this is for the SYN-ACK packet, this number is the host's initial sequence number (ISN).
- **Client Last TCP Sequence Number** – The TCP sequence number of the last packet sent by the client.
- **Server Last TCP Sequence Number** – The TCP sequence number of the last packet sent by the server.
- **First Packet Time** – Timestamp (provided by the sensor's operating system) of the first packet for the session. This value consists of the number of seconds and microseconds since Jan 1, 1970 UTC.
- **Last Packet Time** – Timestamp (provided by the sensor's operating system) of the last packet for the session. This value consists of the number of seconds and microseconds since Jan 1, 1970 UTC.
- **Client First Data Fingerprint** – A numerical score (a type of hash value) representing the data in the first packet containing data from the client.
- **Server First Data Fingerprint** – A numerical score (a type of hash value) representing the data in the first packet containing data from the server.

Below is a single record of a network session captured for this experiment. For privacy concerns the IP addresses have been scrambled, and we have removed some excess spacing used for formatting the record in the log file.

```
7038678  2  63.162.172.250  4305  168.236.226.95  80  0  0  0
____S_  _A__S_  _APRSF  _AP_SF  41  554  6  7  667076575
465154797  667076618  465155076  1068854712 726492  1068854723
881319  -4445  -2968
```

4 On Sequence Numbers And Payload Bytes

One session feature we tracked was the sequence numbers of the first and last packet. Many session analysis efforts, including some of our own past efforts as well as this experiment, tracked the number of payload bytes transmitted in a connection, but these attempts are prone to error due to dropped and retransmitted packets. Since the TCP sequence numbers are used by the TCP protocol to correct for dropped packets and duplicate data, we chose to leverage this information to gain an easy and accurate measure of the number of bytes a host transmitted.

By and large, our experimental results support this hypothesis. Table 1 shows some of the values for several sessions associated with a scanning for port 554, a port used for the Real Audio server. Each row represents a single session, and the fields shown are all associated with the client's activities and include: number of packets sent, number of payload bytes in all of those packets, the difference between the sequence numbers of the first and last packets, and the fingerprint of the data for the first packet containing data.

Table 1: Payload Counts Vs. Sequence Number Deltas

Packets	Payload Bytes	Sequence Delta	Fingerprint
39	39	5	507
18	18	5	507
55	55	5	507
56	56	5	507
3	23	24	-2315
5	23	24	-2315
6	23	24	-2315

As can be from the table, while the number of payload bytes counted extend through a range of values, the difference between the sequence number deltas cluster around just two values: 5 and 24. As we move towards using statistical clustering techniques (e.g., data mining) to identify various types of behaviors, the cleaner the data is that we feed to the statistical algorithms, the more accurate and easier to interpret our results will be. For example, the table's "Sequence Delta" column clearly shows there were two types of behavior, but the table's "Payload Bytes" column does not provide such a simple and obvious view.

> **Lesson Learned:** *Using TCP sequence numbers provides a more accurate account of the number of bytes a host sent than simply counting the number of bytes in the packets.*

If sequence number deltas provide a more accurate measurement of the number of bytes a host sent, should we not record the number of payload bytes observed? The extra variables increase the size of the session records in memory and on disk, and the extra variables increase the number of instructions needed to process each packet and archive each session. However, the variables can provide additional information that cannot be captured by sequence number differences. For example, a difference between the number of payload bytes observed and the difference between the sequence numbers might indicate a problem with one of the hosts (e.g., a host is not responding, and the other host rebroadcasts data). Thus, until we have carried out additional experiments to understand the value of such data, we will continue to record the number of payload bytes observed.

5 On Sequence Numbers and TCP Control Flags

While the difference between initial and final TCP sequence numbers can provide a more accurate measurement of the number of bytes sent by a host, from this round of experiments we discovered careful attention must be paid to subtle interactions between the TCP sequence numbers and the TCP control flags.

Table 1 illustrates the problem. The table has four columns grouped into two meta-columns. The two columns on the left represent the number of recorded payload bytes and the difference between the first and last sequence number for the client host, and the two columns on the right represent the same information for the server. Each row represents a single session. These sessions are associated with a port 80 scan, and, unlike the previous example, the "Payload Bytes" field is very consistent indicating few errors due to packet drops or retransmissions. The two features to observe is that difference between the "Sequence Delta" and "Payload Bytes" payload and how this difference varies.

Table 2: Sequence Number Offsets

Client		Server	
Payload Bytes	**Sequence Delta**	**Payload Bytes**	**Sequence Delta**
22	23	114	116
22	23	114	116
22	23	161	162
22	23	161	162

As can be seen in Table 2's client data, the difference in the sequence number delta and observed payload bytes is consistently 1, while the difference in the sequence number delta and payload bytes can be either 1 or 2. After conducting additional research we discovered that these differences were a product of (1) the interaction between the TCP control flags, primarily the SYN and FIN control flags, and the TCP sequence numbers and (2) which host initiated host session shutdown.

Because of the complexity of the issue, we wrote a separate technical report detailing the problem and how to correct it. However, the sensor used for this round of experiments did not adjust for these issues, so when reading this report, keep in mind that the difference in the sequence numbers may be off by either 1 or 2 from the actual data transmitted by the host.

> **Lesson Learned:** *The TCP control flags and the order of session shutdown affects how the difference between sequence numbers values must be adjusted to arrive at an accurate representation of the number bytes transmitted by a host.*

While determining the underlying cause of the sequence number offset we discovered that, in general, when a host's sequence number is off by 2 from the actual number of bytes transmitted, that host initiated session shutdown, but when the sequence number is off by then the other host initiated session shutdown. For example in the first two session in Table 2 the servers' sequence number deltas, 116, are off by 2 from the number of bytes (probably) sent, 114, indicating that the servers initiated shutdown. For the last two sessions, the difference between the sequence number deltas, 162, and the number of bytes sent, 161, is 1 indicating that the client initiated session shutdown.

The observation that when the server transmitted 114 bytes it also shutdown the session but when the server transmitted 161 bytes the *client* shutdown the session intrigued us. Why is there a correlation between the number of bytes transmitted by the server and which host initiated shutdown? For this round of experiments we did not pursue the answer, but we did realize that knowing which host initiated shutdown might be useful. For example, if a client sends an attack and the server properly handles it (i.e., it is patched or the vulnerability does not exist), the server may shutdown the connection, but if the attack is successful the connection may remain open waiting for the client to shut it down. For the next round of experiments we will augment the sensor to identify which host initiated shutdown.

> **Lesson Learned:** *Knowing which host initiated session shutdown may indicate how the server handled an attack or probe. Future sensors should record this information.*

6 Initial Survey: Looking for Automated Attacks

To help test the hypotheses that (1) attacks or probes can be identified by detecting repeated behavior by clients (not necessarily the same client for each session), (2) variations on attacks can be identified by separate clusters of behaviors by the clients, and (3) the effects of the attacks (success or failure) can be identified by clusters of behaviors by the servers, we needed to observe some attacks in the wild. Since we did not have a traditional IDS sensor running for this

experiment, we were not using honeypots to validate attacks, and in general did not have ground truth as to what attacks were occurring on the monitored network, we decided to identify possible attacks by simply looking for some egregious patterns of behavior on the network. This section summarizes our approach for identifying the candidate attacking hosts.

Our first attempt to find attacking hosts simply identified the clients that initiated the largest number of failed connection. This approached was based on the observation that many attacks scan large swaths of networks or randomly select IP addresses to find potential targets, and since most IP addresses do not have the targeted service running most of these session requests would fail.

This approach of identifying large numbers of *failed* sessions differs slightly from many IDS sensors that simply identify clients that initiate large number sessions. We are aware of many reports that these "scan detectors" often generate false alerts because of a host's legitimate but heavy activity on the network. By focusing on failed connections we believed we could eliminate many of these false positives.

For our analysis we tracked client address and server port pairs. That is, for each client address we tracked its connection attempts to a specific server port. We then tracked the following metrics for each client and server port pair:

- **Rejects** – the number of sessions in which the server rejected the session request with an RST packet.
- **Silents** – the number of sessions for which the server never responded. In many cases a server may not respond because there is no machine at that IP address or the server is currently down. Also, a firewall (either on the host, performed by a router, or performed by a firewall hardware device) may simply silently drop the client's connection request.
- **Success** – the number of sessions that were accepted by the server (the server sent back a SYN-ACK packet).
- **Data** – the number of sessions that were accepted by server and for which the client send some data. Since we are interested in probes and attacks by the client and the various responses by servers, we need to identify sessions in which the client sent data.

From these variables we also constructed a **Count** field that represents the sum of the **Reject** and **Silents** variables; this value represents the total number of failed sessions requests from the client to the given server port.

6.1 First Results

Table 3 shows a partial result of this summary (note: for privacy reasons the client IP addresses have been replaced with "Host 1", "Host 2", etc.). Each row represents the scanning activity of a single host to a single server port, and the rows are sorted on the Count field – the number of failed sessions. For example, the first row shows "Host 1" had 132,267 failed session requests to port 135, the vast majority of those session requests failed because the server never responded.

Table 3: Scanning By Connection Count

Count	Client	Port	Rejects	Silents	Success	Data
132267	Host 1	135	8	132259	0	0
117653	Host 2	135	8	117645	0	0
108642	Host 3	135	8	108634	0	0
71796	Host 4	135	8	71788	0	0
70279	Host 5	135	8	70271	0	0
64831	Host 6	135	8	64823	0	0
40949	Host 7	554	12372	28577	38	33

As can be seen from Table 3 scans to port 135 dominate the list. These are hosts inside the organization that are probably infected with the Welchia or Blaster worms. The organization has their border router configured to block transmission of traffic to port 135, so the router drops most of the session requests. A few sessions did reach a host (probably an internal address), but in each case the session request was rejected. Not until the seventh host do we see a scan/probe that actually connected and sent data to server.

Since we are interested in profiling behavior, we focused only on scanning behavior for which the client connected and sent data to servers. Table 4 shows the first seven rows of scanning activities for which the client sent data to at least one server. The first row shows a host scanning port 554, probably trying to find vulnerable RealNetworks media servers. We will go into additional details on this scan later. Of more immediate concern to us are the apparent scans to ports 4662 and 25 (lines 2, 3, 4, and 6 in the table).

Table 4: Scanning By Connection Count (Client Data Sessions)

Count	Client	Port	Rejects	Silents	Success	Data
40949	Host 7	554	12372	28577	38	33
24370	Host 9	4662	12991	11379	13827	13734
24189	Host 10	25	7100	17089	14304	14237
18310	Host 12	25	4718	13592	1561	1501
16530	Host 13	1433	4700	11830	362	123
13670	Host 14	25	3245	10425	1233	1154
12922	Host 15	80	3534	9388	596	595

While the “scans” to ports 4662 and 25 included a large number of failed sessions, they also had a large number of successful connections – numbers that are much higher than we would expect from blind scanning activity. Port 25 is the default sendmail port, and port 4662 is the default port for eDonkey, a file sharing system. Upon further analysis we decided these clients were not scanning since they repeatedly tried to connect to the same server, but their heavy and repetitive connection attempts (e.g., trying to deliver email to a server that is overloaded, down, or non-existent) generated large numbers of failed sessions. In general, sorting client behavior by the number of failed connections to a given port tended to identify email servers trying to deliver email and hosts participating in peer-to-peer networks.

6.2 Second Results

After determining that identifying scanning activity by tracking the number of failed sessions tended to generate false positives because of repeated attempts to deliver email or contact a peer-to-peer host, we changed our counts to reflect the number of unique servers a host tried to contact. Now the **Rejects** column refers to the number of unique IP addresses that sent back a RST packet in response to a client’s connection request. This way, if an email agent or

peer-to-peer host made repeated but rejected connections to a given host, all those attempts would count as a single event. The **Silents**, **Success**, and **Data** columns have similar interpretations.

The one caveat is that while the **Count** column is still the sum of the **Rejects** and **Silents** columns, this field does not necessarily reflect the number of *unique* servers that did not accept connections from the server. The reason for this is that a single server may have at one time failed silently and then later rejected a session, thus the same IP address would be counted twice in the **Count** field. More precisely, the **Rejects** and **Silents** sets are not disjoint, so the sum of the elements in these sets may over count the actual number. Nevertheless, we still feel the **Count** field represents a relatively accurate measurement of the number of servers that did not accept connection requests from the client.

Table 5 show the result of our new approach for identifying scanning hosts. The number of the host listed in the **Client** column represents their order of output using our new counting method and does *not* correspond to the host numbers used in Table 3 and Table 4. Once again, we have removed any rows for which the client did not send any data to the servers. As can be determined by the host number in the **Client** column, the top six scanning clients did not send any data to servers, and of the top 23 (the number of the last host displayed in the table), 16 scanning clients did not send any data.

Table 5: Scanning by Unique Servers (Client Data Sessions)

Count	Client	Port	Rejects	Silents	Success	Data
40949	Host 7	554	12372	28577	38	33
16527	Host 10	1433	4700	11827	239	17
12922	Host 11	80	3534	9388	596	595
9443	Host 18	80	8129	1314	2046	2044
7326	Host 20	80	1839	5487	1137	1136
6503	Host 21	80	2128	4375	1303	1302
4406	Host 23	80	1727	2679	845	845

This approach of counting unique servers as opposed to total number of connections appears to perform much better. In particular, the email agents and peer-to-peer hosts that were making repeated but failed connections to connect to specific hosts have been pushed much further down the list. Furthermore, upon further analysis we are confident that the first three hosts listed in Table 5 were indeed scanning, probing, or attacking other hosts.

> **Lesson Learned:** *Counting the number of unique servers that did not accept a connection from a client is a better predictor of scanning activity than simply counting the number of failed connections.*

6.3 Second Refinement

While our new approach of counting connection failures to unique IP addresses is a better approach than simply counting failed connections, we still believe there are better approaches. The problem can be seen in the fourth row of Table 5, where Host 11 appears to be scanning port 80. The count of 9,443 servers that rejected connection attempts seems very high, but at the same time the Host 11 successfully connected to 2,046 unique servers. That ratio of success to failure is too high for a client blindly scanning the network. Furthermore, since the set of unique addresses in each column set do not represent disjoint sets, some of the addresses in the **Rejects** and **Silents** columns may have also accepted connections at one time but because of various reasons (e.g., server was overloaded) the host later did not accept the connections. Thus, the **Count** column does not necessarily represent addresses for which there is no server (hence,

indicating a blind probe by the client) but addresses for which, for whatever reason, a connection request failed.

A more appropriate approach is to treat failed connections differently depending on whether at one point in the recent past there did exist a server at the target IP address and TCP port. For example, if host 10.0.0.2 usually runs a web server running on port 80 and host 10.0.0.7 has never run a web server on port 80, then a failed connection to 10.0.0.2 should be considered *less* suspicious than a failed connection to port 10.0.0.7.

To treat failed connections differently the sensor must know which IP addresses had servers and on which ports. With this information the sensor can more accurately and more quickly identify blind scanning attempts by a host. In short, the more information a sensor has about the environment the better the sensor can perform. Our sensor supports server identification and traffic profiling,

> **Lesson Learned:** *By identifying active servers in the network, and possible common clients for those servers, a sensor can more accurately measure the intent of a failed connection and more precisely and quickly detect probing activities.*

7 Analysis of Client Activity

In this section we analyze the activity of several hosts that appear to be scanning, probing, or attacking the network.

7.1 Scanning Port 554

Count	Client	Port	Rejects	Silents	Success	Data
40949	Host 7	554	12372	28577	38	33

The first scanning activity we examine is Host 7 scanning port 554, a port commonly used by RealNetwork's media server. By our accounting method of identifying the number of servers that failed to accept the connection (see Table 5), this was the top scanning activity that actually managed to connect to some servers and send data. Roughly 41,000 servers rejected connection attempts, while 33 servers accepted the connection and data from the client. This high failure to success ratio leads us to believe that Host 7 was blindly scanning for RealNetworks media servers with a buffer overflow vulnerability that was announced in August of 2003 [CERT 03].

To further analyze the client's activity we ran a program on the original session logs to extract several of the core flow information. In particular, the number of packets, payload bytes observed, sequence number difference, and fingerprint (F-print) for the client and server are extracted, and a sample of some of the sessions is shown in Table 6.

Table 6: Sample Flow Vectors for Port 554 Scan

Client				Server			
Packets	Payload	Sequence	F-print	Packets	Payload	Sequence	F-print
39	39	5	507	37	0	1	0
55	55	5	507	54	13	14	1154
5	23	24	-2315	4	107	109	4027
7	46	25	-2315	5	107	108	4027

Next, we surveyed the value distributions for several of the fields in the vector to identify clusters for a single variable. Figure 1 shows the distribution of values for sequence number delta for the client. As the figure shows, there are two core clusters for the number of bytes transmitted

by the client. The first cluster sequence delta is 5 (12 sessions), and the second cluster is 24 (18 sessions). There is also a small blip (2 sessions) at a sequence delta value of 25. As mentioned in Section 6, TCP control flags and the order of session shutdown affects the sequence number delta values, and our sensor used in this first round of experiments did not adjust for this behavior. For this reason we believe the 24 and 25 sequence number deltas really represent the same amount of data transmitted by the client. Thus, we believe the client sent three different amounts of data.

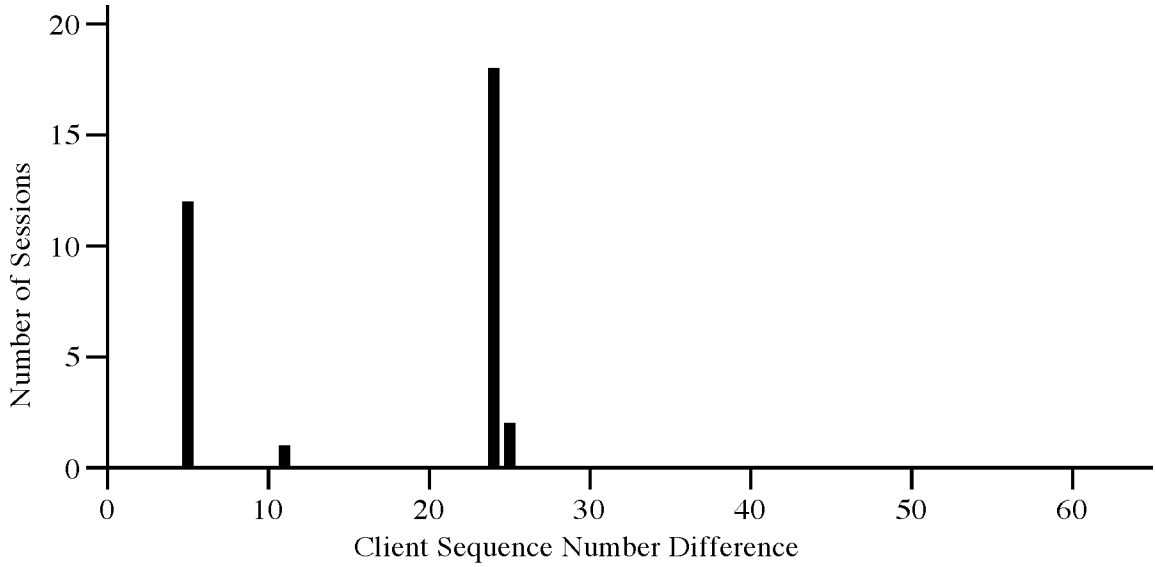

Figure 1: Data Amount from Client in Port 554 Scan

Next, we looked at the fingerprint of the first packet of the data. A survey of the values showed three fingerprint values: -2315 (for 20 sessions), 507 (for 12 sessions), and 1745 (for 1 sessions). Not surprisingly, with three clusters of the amount of data transmitted and three clusters of fingerprints, there is a strong correlation between the amount of data and the content of the first packet of data. In particular:

- If sequence delta is 5, then the fingerprint value is 507 100% of the time.
- If sequence delta is 11, then the fingerprint value is 1745 100% of the time.
- If sequence delta is 24 or 25, then the fingerprint vale is -2315 100% of the time.

To summarize this initial survey of client analysis, it appears that Host 7 has three types of behavior.

7.2 Scanning Port 1433

Count	Client	Port	Rejects	Silents	Success	Data
16527	Host 10	1433	4700	11827	239	17

The second scanning behavior we examine is Host 10 scanning port 1433, one of the ports used by MS-SQL server, the target of slammer worm. Once again the large failure to success ratio indicates blind scanning for servers.

7.2.1 Success and Data Column Differences

The first question we had was why the client sent data to only 17 servers despite successfully connecting to 239 servers? Upon deeper investigation we identified a number of connections that, despite the server accepting the connection request by sending back a SYN-

ACK packet, appeared to be a failure. In these connections the server tended to only send a single packet, the SYN-ACK packet, but the client tended to send 7-8 packets. Either the client did not receive the SYN-ACK and continued to send SYN packets, or the client received the SYN-ACK and continued to wait for the server to send the first data.

One approach to resolving the mystery would be to analyze the protocol specification to determine which host, the client or the server, should send the first data. In general for commercial network services, however, the protocol specification may not be publicly available. Furthermore, for fast moving attacks, locating, reading, and understanding the protocol specification may be too slow of a process.

An alternative approach would be to look at sessions that did exchange data to determine which host sent the first data. In addition to helping resolve the mystery of why apparently successful connections failed to exchange data connections, it could help explain which host drives the other hosts behavior. For example, if the server responds with data first (e.g., identifying the server name and software version), the client can modify its first data based on the server's announcement. On the other hand, if the client sends data first, the observed behavior from the server is driven by client's data. Unfortunately, the sensor developed for this experiment did not record which host sent the first data.

> **Lesson Learned:** *Recording which host sent the first data in a session may help identify a cause and effect vector when the sensor identifies a statistical correlation between the client and server behavior.*

7.2.2 Client Fingerprint Distribution

Next, we turned our attention to the distribution of the client's fingerprint. Figure 2 shows a distribution of fingerprint values for the client's first data packet. The X-axis is simply a set of columns, where each column represents a unique fingerprint value. There is no linear scaling or any other type of scaling implied by the X-axis. We chose this representation because our interests are on the number of unique values and any clusters in those values. The values for the three largest clusters are shown. The Y-axis represents the number of sessions for a given fingerprint.

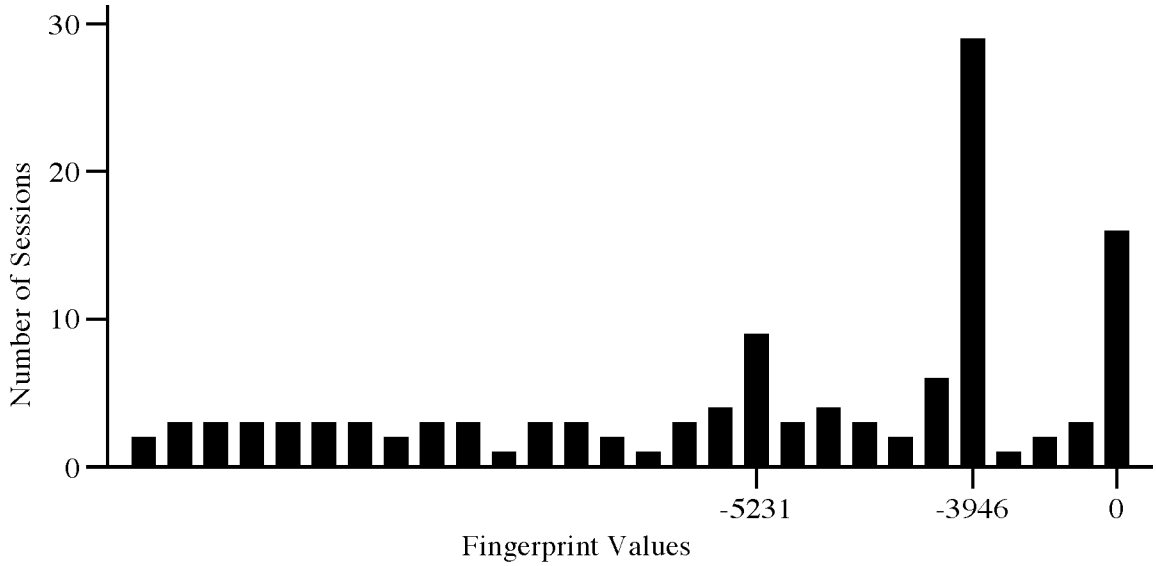

Figure 2: Distribution of Client Fingerprint Values

The first most noticeable feature of this distribution is that is not as well clustered as we had expected. In the previous scan of port 554 there were three clusters of client behavior with fingerprint values of 507 (12 sessions, each with a sequence number delta of 5), -2315 (20 sessions associated with the sequence delta of 24 or 25), and 1745 (1 sessions with the sequence delta of 11). In Figure 2 we can see that there were 28 unique fingerprint values.

Figure 2 shows three clusters with fingerprints of –3946, -5231, and 0. Upon seeing this distribution our initial assumptions was that the client had a range of attacks or probes that it could use, each which created a different fingerprint, and that for whatever reason the attacker preferred, or at least used most often, the attacks/probes that generated the fingerprints –3946, -5231, and 0. However, upon a deeper analysis we discovered that all sessions with the fingerprint –3946 went to only one target host; the same was true for sessions with the fingerprint –5231. For fingerprint values of 0, only two target hosts were targeted.

In general, the fingerprints varied with each target, so from a checksum or fingerprint point of view we will not see strong clustering of content for this type of attack. Because of the high success to failure ratio, we are confident that this activity represents some type of malicious probing or attacking activity. Also, because the attack was so noisy (fast, lots of failures) and stands out to simple scan detectors, we do not believe the attack is designed to be polymorphic to avoid detection, so we believe the polymorphism was an accidental side effect of the attack design.

This accidental polymorphism is not unprecedented. Figure 3 shows the data from two packets captured during the Blaster Worm attack in the summer of 2003. Each attack actually involved several connections, and this portion shows the original attacking machine sending a Trivial FTP command to the backdoor installed during a previous leg of the connection. The TFTP command is used to retrieve the rest of the worm from the attacking host. For our purposes, the interesting aspect of the connection is the IP address included in the first packet of data, the portion of the connection we were fingerprinting for this round of experiments. For each new machine that is infected, this address portion of the attack data will change to reflect the address of the newly infected host. Thus, from a sensor observing this leg of the attack, every connection from one infected host will look slightly different than a similar connection from another infected host. Thus, while the attacker did not intend to build a polymorphic attack, by embedding address information into the attack the attacker has created a mildly polymorphic attack.

Two data packets from a client to port 135 on a honeypot captured during the MS Blaster Worm on 8 Aug 2003. The actual IP address has been changed from the original data.

`tftp -i 192.168.1.12 GET msblast.exe`	First Data Packet
`start msblast.exe`	Second Data Packet

Figure 3: Example of Accidental Polymorphism

Lesson Learned: *Polymorphism can negatively affect clustering-based analysis.*

Lesson Learned: *Some attacks may inadvertently include mild polymorphism.*

7.3 Scanning Port 80

Count	Client	Port	Rejects	Silents	Success	Data
12922	Host 11	80	3534	9388	596	595

The third scanning behavior we examine is Host 11 scanning port 80, the default port for the web server and the source of many vulnerabilities.

In contrast to the previous example where the client exhibited some polymorphism, this client had no variation – all 595 data-carrying connections had a client data payload of 22 bytes and fingerprint of 1503, both very strong cluster signals. The only interesting variation for the client was in the delta sequence number: 460 sessions has a delta sequence of 23 while 135 sessions had a delta sequence of 135.

The servers showed a more interesting and varied behavior, but there were still very strong clusters. Figure 4 shows the all the server sequence number deltas and how many sessions had that value. Once again, the X-axis is not scaled; there is one column for each unique value observed. There are four clear clusters at sequence delta values 2, 111, 162, and 409.

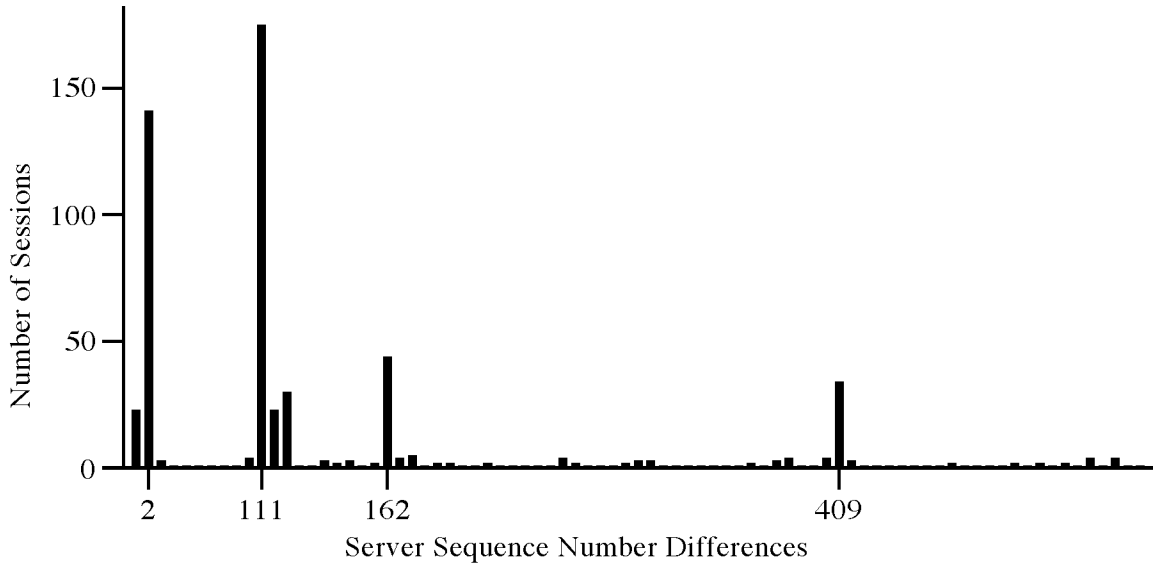

Figure 4: Server Sequence Number Deltas for Port 80 Scanning

The server fingerprint values also showed strong clustering, with four primary clusters at fingerprint values of 0, 6433, 8253, and 14988 (see Figure 5). Once again the X-axis simply represents all the uniquely observed fingerprint values (151) and has no implied scaling.

Given that there were four clusters for both the sequence deltas and fingerprints values, we thought there would be a strong correlation between these values. The following were the strongest correlation values:

- When the sequence delta was 111, the fingerprint value was 8253 100% of the time.
- When the sequence delta was 2, the fingerprint value was 0 100% of the time.
- When the sequence delta was 162, the fingerprint value was 14988 82% of the time.

Slightly more interesting was that the sequence delta of 409 did not strongly correlate with any single fingerprint value, but when the sequence delta was 5409, the fingerprint value was in the range of 62693 – 65647 100% of the time. The reverse largely held true as well: when

the fingerprint was in the range of 62693 – 65647, the sequence delta value was 409 85% of time and either 408 or 409 95% of the time.

This last observation demonstrates the value of our fingerprint approach over traditional checksum values. Our fingerprints of data streams are designed such that similar data generate fingerprints that are similar in value, and this gives us a better chance at identifying clusters of similar behavior. With traditional checksums, any small difference in the data streams will produce wildly different checksum values, and this prevents statistical correlation algorithms (e.g., data mining) from finding the correlations.

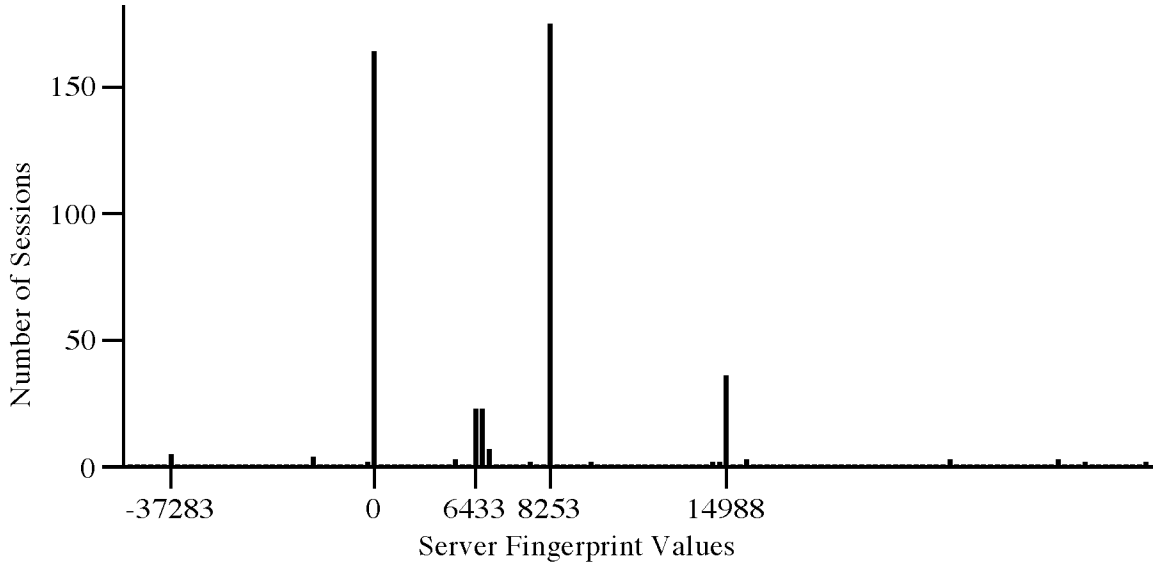

Figure 5: Server Fingerprints for Port 80 Scanning

8 Conclusions

This report describes the first in a series of experiments designed to determine how we can leverage network-based session records in our Environment Aware project. Traditional network security sensors (e.g., intrusion detection systems such as RealSecure and Snort) primarily look for known attack patterns and only report attacks. However, a number of organizations such as the Air Force and the Department of Energy use sensors that create session records for all network activity observed on part of their networks, typically on the border between the site and the rest of the Internet.

We believe these session records can provide a number of capabilities, and our goals for the series of experiments include (1) identifying how session records can add value to an organization, (2) quantifying that value, and (3) determine how the session records can be modified or extended to increase value.

For this first experiment we had several more specific goals, including (1) validating that our software works and is stable enough to serve as a foundation for future experiments, (2) validating of some of our assumptions (e.g., attacks should exhibit statistical clustering behavior), (3) determining a set of questions to ask of the network audit trails, (4) developing an initial set of tools to answer those questions, and (5) identifying areas of improvements and new questions to answer for future experiments.

As documented in Section 2 our software performed well on multiple gigabyte optical networks with no crashes or memory leaks. Section 3 described our session record format used

for this series of experiments. Sections 4 and 5 identified some of the first lessons learned and areas for future improvements. In particular, while TCP sequence numbers are an inexpensive and accurate means for measuring the number of bytes one host sent another, careful attention must be paid to the interaction of the TCP control flags with the TCP sequence numbers.

Sections 6 and 7 documented the questions we asked of our data and the answers we received. For the analysis in these sections we had to develop a number of analysis and presentation tools. Also, throughout these sections we identified a number of areas of future improvements and hypotheses we wish to test in future experiments. Section 6 focused on relatively simple techniques to identify attacks in the data set (we were not using traditional intrusion detection sensors during the experiment), and Section 7 drilled down into a couple of specific attacks and examined the clustering behavior in these data sets.

8.1 Lessons Learned

Throughout this experiment we learned a number of lessons, some of them unexpected, and we identified areas we believe require testing in future experiments. These items and the sections in which they are documented are summarized below:

- Using TCP sequence numbers provides a more accurate account of the number of bytes a host sent than simply counting the number of bytes in the packets. (Section 4)
- The TCP control flags and the order of session shutdown affects how the difference between sequence numbers values must be adjusted to arrive at an accurate representation of the number bytes transmitted by a host. (Section 5)
- Knowing which host initiated session shutdown may indicate how the server handled an attack or probe. Future sensors should record this information. (Section 5)
- Counting the number of unique servers that did not accept a connection from a client is a better predictor of scanning activity than simply counting the number of failed connections. (Section 6.2)
- By identifying active servers in the network, and possible common clients for those servers, a sensor can more accurately measure the intent of a failed connection and more precisely and quickly detect probing activities. (Section 6.3)
- Recording which host sent the first data in a session may help identify a cause and effect vector when the sensor identifies a statistical correlation between the client and server behavior. (Section 7.2.1)
- Polymorphism can negatively affect clustering-based analysis. (Section 7.2.2)
- Some attacks may inadvertently include mild polymorphism. (Section 7.2.2)

9 References

[CERT 03] *Vulnerability Note VU#934932*, CERT Coordination Center, 29 Aug 2003.