Review – a Tool For Reviewing Tcpdump Packet Logs

Steven M. Romig
University Technology Services
The Ohio State University
Columbus, Ohio 43210
614-688-3412
romig@net.ohio-state.edu

April 30, 1997

Abstract

The Ohio State University uses **tcpdump** to log the network traffic of crackers who use our modem pool or other computer resources so that we can identify them and monitor their activities and targets. Reading through the logs is tedious and time consuming, due in part to the quantity of information, and because some of the network traffic is hard to decipher. Picking text out from among terminal escape sequences is painful, and deciphering traffic to or from X Windows servers is very difficult to do without special tools.

We have written some tools to make it easier to read through our **tcpdump** logs. The main program, which we call **review**, allows one to view various summaries of the logs and to view the packet contents. The program also allows the user to replay remote login and X Windows sessions at a controlled rate, so the user can reconstruct what would have appeared on the cracker's screen.

This paper describes the inner workings of the **review** system and its uses. We also discuss some of what we have learned from reading our logs, and we list some of our future plans.

1 Introduction

The Ohio State University (OSU) is a large land-grant, state supported school with roughly 50,000 students and 20,000 faculty and staff. At least 50,000 of these people have access to the computer services supported by the campus computing center, which includes access to the Internet through walk-in computer labs and dialup access through a large modem pool using the Serial Line Internet Protocol (SLIP) or the Point-to-Point Protocol (PPP).

Most of the computer security incidents handled at OSU involve the Internet in some fashion. For example, forged electronic mail in a mail bombing attack is bounced across the Internet from OSU through a remote site and back. People break into our computers from across the world to gain access to disk space and set up "warez" sites (repositories of stolen software or other goods) which their friends access through the Internet. People launch security attacks or denial of service attacks across the Internet to or from our computers.

We have found it useful to log the contents of network traffic on some occasions. We most often use programs such as **tcpdump**[8] to capture packet headers or to record the full packet contents which we will later analyze to determine what happened during an attack. For example, we might look at the Internet Protocol (IP) packet headers and Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) port numbers to see what types of network activity the logs contained and infer the type of activity from the port numbers. If we look at the data portion of the packets in a TCP connection to a Simple Mail Transfer Protocol (SMTP) server we can see the mail messages sent from the client to the server and the response codes the server sends in reply. Examining the packet contents from Telnet or Rlogin sessions (two remote login protocols) might reveal how someone is breaking in or what they are doing after they have broken in. If we read through the contents of packets sent to or from Internet Relay Chat (IRC) servers (IRC is a sort of "citizen's band radio" for the Internet) we might learn who they are as they talk to their friends or what they plan to do next.

Analyzing detailed packet logs such as the ones produced by **tcpdump** can be time consuming and tedious. Suppose we have a simple log containing packets for a single TCP connection to a Telnet server. If we want to see what the user typed we just extract the packets sent from Telnet client to the server with a suitable **tcpdump** filter (e.g. `src host A and dst host B and dst port 23`). Then we extract the printable ASCII portion of the data payload of each packet with a tool like **tcpshow**. Looking at the traffic sent from the Telnet server to the client (which is mostly material that would be displayed on the user's screen) is also possible, though made somewhat more difficult by the fact that plain text is often interspersed with terminal escape sequences to do things like cursor positioning or to blank portions of the screen. The situation is especially bad when viewing a session where someone uses a "full screen" type of program such as **emacs**, **pine** or **vi**. In these cases fragments of text are drawn at various locations on the screen as needed and simply extracting printable ASCII text is insufficient for getting a clear idea of what the screen looked like. Figures 1 and 2 show the printable ASCII contents of the Telnet client and server communications stream.

Most TCP logs are not nearly so simple – many of our crackers run multiple, often concurrent network applications during their login sessions. While they talk to their friends on IRC they also send several mail messages, check for new mail on their Post Office Protocol (POP) server several times, visit several dozen web sites, try a few **phf** probes (**phf** probes check and abuse security problems in **phf** scripts on some web servers), transfer files with the File Transfer Protocol (FTP), and play a game of Quake. The packets for all of these network activities would be intermingled in a single log file – to make sense of them one would first have to list all of the things that are present and then extract and view the separate sessions one by one. It is straight-forward to do this with **tcpdump**, though tedious to do so repeatedly for many different sessions.

Finally, one can accumulate stunning amounts of log data in a short amount of time. Since we use automated mechanisms for collecting **tcpdump** logs of some of our crackers (Section 4) we have accumulated well over 15 gigabytes of **tcpdump** logs in the last 8 months. Reading through all of these logs required

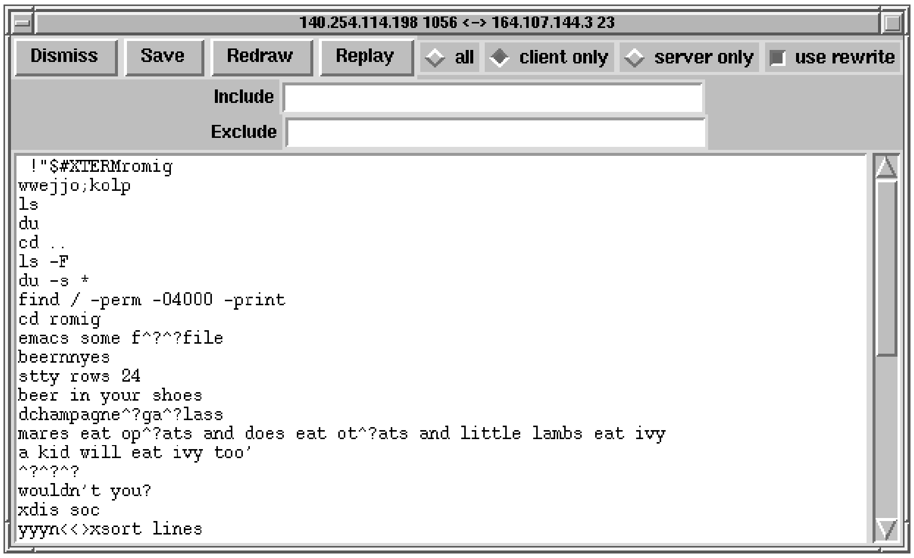

Figure 1: Session browsing, window showing Telnet client traffic (what the user typed). Yes, that is the author's old password at the top.

more powerful and flexible tools than were available when we started.

We have written a set of tools that help automate the collection and analysis of **tcpdump** style logs and which make it quicker and easier to view the contents. We call the main program **review** since we use it to review what our crackers have been up to lately.

The rest of this paper is organized as follows. Section 2 covers related work. Section 3 discusses the **review** program and its features. Section 4 covers our tools and procedures for gathering and processing packet logs. Section 5 discusses some of what we have observed from reading our logs. Section 6 covers our future plans. Section 7 presents a summary of this paper.

2 Related Work

Several tools are available for collecting packet logs from networks. **Tcpdump** is probably the most widely used. It does not provide for viewing the data portion of the packets captured (at least not in human readable format), but it does allow for printing the complete packet in hexadecimal. The SunOS 4 **etherfind** program and the SunOS 5 **snoop** programs can be used in a similar fashion. Other vendors often provide similar programs for their systems.

Several people have written programs to convert the hexadecimal packet dumps from **tcpdump** into readable ASCII text. Michael Ryan's **tcpshow** program [7] gives a summary of packet headers and allows the user to view the printable ASCII portion of the each packet's data. We also like Mark Fullmer's **cleanup** program [6] which skips the packet header summaries making it easier to see what people were typing in Telnet or Rlogin sessions. We used an extended version of **cleanup** in **review**.

Brecht Claerhout has written a program called **sniffit** [1] which can be used

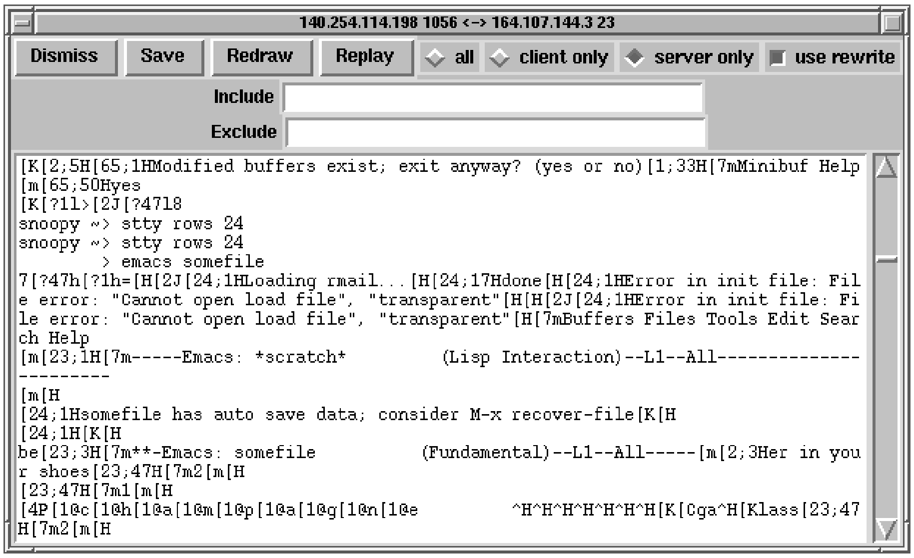

Figure 2: Session browsing window, showing Telnet server traffic (what was sent to the user's screen).

to monitor and log network activity. This program seems to lend itself more to monitoring of real-time network activity rather than toward making and reviewing logs of past activity.

Texas A&M University's TAMU security package [2] includes programs for logging TCP and UDP activity on a net (**tcplogger** and **udplogger**) and for detecting unusual network activity (**netwatch**). The logging tools are useful for recording network activity but do not record the contents of network sessions. The **netwatch** utility examines the contents of certain types of sessions and reports suspicious activity within them. This could be useful in other contexts, but we would prefer to restrict such intrusive monitoring to specific network sessions belonging to suspected crackers and we are interested in capturing all of their traffic, not just the suspicious traffic. Of course, we would also like to be able to detect overtly suspicious activity in the sessions that we log as well.

En Garde Systems's **ip-watcher** [4] allows the user to monitor network activity in real time. This program is especially appealing because it reconstructs what the client's view would look like for Telnet or Rlogin sessions, making it easy to see what the crackers were doing when they user "full screen" programs like **pine**, **vi**, **pico** or **emacs**. We have used this at OSU for monitoring past incidents and have found it to be very useful. However, we found **tcpdump** to be more suitable for use in automated log collection. Furthermore, we could not readily extend **ip-watcher** to cover protocols other than Telnet, Rlogin, SMTP and remote shell. Although it addresses replaying Telnet and Rlogin sessions very nicely it does not handle the replaying of X Windows sessions at all.

What we are missing is a tool that allows one to conveniently move among log files, examining the contents of each in increasing detail, extracting interesting packets from the total log so that one can focus on one part of it, and coordinating the extraction and presentation of the contents of each session so that we can more readily view sessions using more complicated protocols such as the X Windows protocol.

3 The Review System

The front end for our log reviewing software is a Perl/Tk script which we call **review**. It is typically used by passing a list of log files or directories containing log files on its command line. **Review** allows the user to navigate among and through the log files using the mouse to select log files or sessions within the log. **Review** allows the user to filter and reform the contents of the packets logged according to user defined parameters, and also supports protocol specific defaults for convenience.

3.1 The Log Listing Window

One reviews a set of log files by running **review** with a list of log file names (or log file directories) on its command line (e.g., `review log1 log2 logdir`). **Review** then creates a window containing a scroll-able list of the log files, which we call the log listing window (Figure 3).

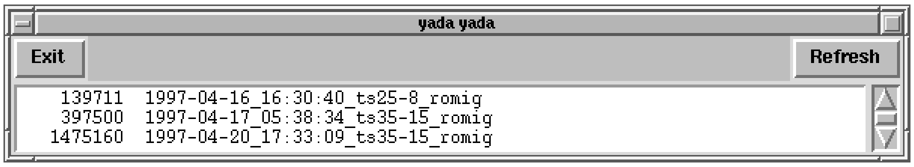

Figure 3: A typical log listing window.

Review lists two pieces of information for each log file: the name of the log file and its size in bytes. To select a log for review triple click on its entry in this window. **Review** then creates a "session summary" window for this log (Section 3.2).

Clicking the **refresh** button at the top of the log listing window causes **review** to redraw the list of log files which is useful when files have been added, renamed or deleted. The **exit** button is used to quit **review**.

3.2 The Session Summary Window

Triple clicking on a log file in the "log listing" window creates a "session summary" window for the selected log (Figure 4). The session summary shows a summary of the traffic that the log file contains. Rather than showing individual packets (ala `tcpdump -q`) the summary shows the real or implied sessions within the log. For TCP traffic the sessions correspond to individual TCP connections. For example, a connection from a mail client on a cracker's host to a remote mail server is a single session and is represented by a single entry in the session summary window. For non TCP traffic we simply group similar traffic together – UDP traffic between port 3248 on one host and port 2049 on another would be denoted as a single session, regardless of whether there were 1 packet or 1000. Each line of the session summary shows the number of packets in the session and gives information about the session such as the source and destination IP addresses and port numbers.

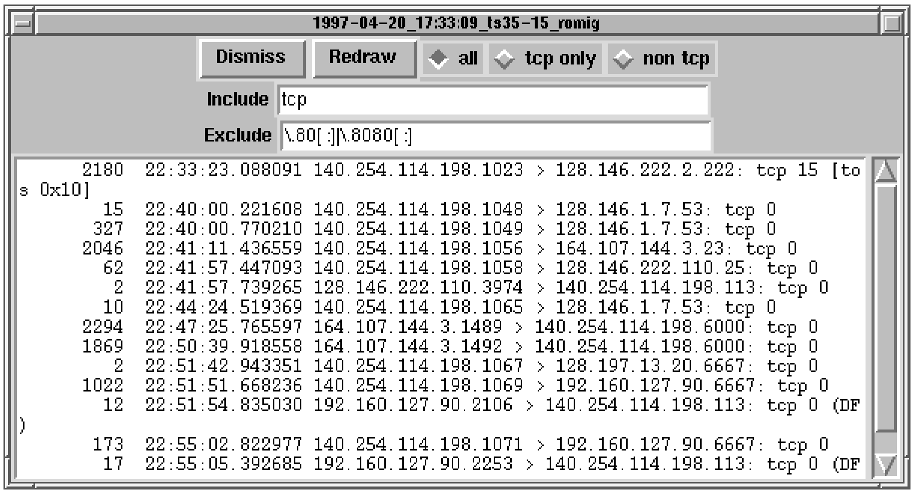

Figure 4: A typical session summary window, showing TCP traffic without the web traffic.

The session summary for a log can be created on the fly if necessary, but one can also create them ahead of time using the `review -summary` or `review -cache` options. Creating them ahead of time makes reading through long logs less time consuming (Section 3.6).

Summaries for long sessions can be correspondingly long. The **include** and **exclude** entries at the top of the window cause **review** to include or exclude session entries that match the given regular expressions. For example, setting **include** to
`.80[ :]` would include only sessions to or from port 80 on any host. The **redraw** button causes **review** to redraw the session summary to make changes to the include/exclude settings take effect. In Figure 4 **include** is set to "tcp" so that only TCP sessions are shown, and **exclude** is set such that most web sessions will not be shown.

The **dismiss** button deletes this summary window.

Triple clicking on a session in the summary window creates a "session browsing" window.

3.3 The Session Browsing Window

The session browsing window displays the contents of the data portion of the packets in a scroll-able window, with optional rewriting and inclusion/exclusion based on regular expression matching (Figure 5). This is the heart of **review** and most of its features will be described in this section.

The user can choose between viewing the contents of all packets, client packets (sent from what is assumed to be the client to the server) or server packets (sent from what is assumed to be the server.) The defaults depend on the type of session being viewed – for Telnet and IRC sessions, we have generally found it more useful to view the client to server traffic (to see what the alleged cracker typed), but for connections to POP or other mail servers it is more edifying to view the server to client traffic (to see the mail being read).

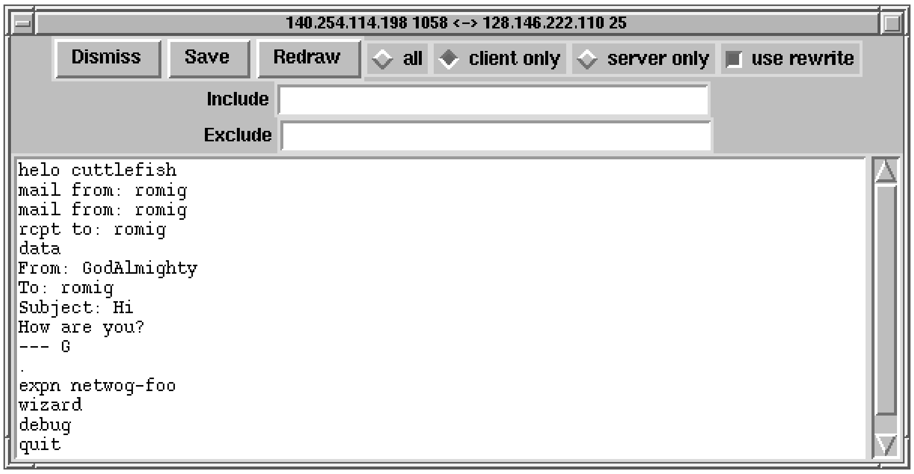

Figure 5: Session browsing window, showing SMTP client traffic.

As with the session summary window, user specified regular expressions are used to include or exclude matching lines from the display. The **redraw** button causes **review** to redisplay the text to make changes take effect.

Review has built-in rewriting rules that are used to clean up the text being displayed. For example, the contents of IRC sessions are often obfuscated with uninteresting commands (and their replies) that the clients send to the server to inquire about the status of the channel, detect the presence of friends, check the identity of other channel members, or simply to keep the connection alive. When **review** displays an IRC session the default **exclude** patterns are set to exclude these sorts of messages from the display. The output is also passed through an IRC rewrite script that reformats some of the messages to make things more readable (e.g. converting IRC nicknames to standard form). The use of the rewriting script can be enabled or disabled by clicking on the **use rewrite** check box.

The session type is determined using an internal table that matches known port numbers to session types (e.g., port 21 is FTP, port 23 is Telnet, port 25 is SMTP, port 6666 is often IRC). Since crackers often set up network services to run on strange or unpredictable port numbers, **review** allows the user to augment the default list of session types through a configuration file which allow the user to specify host specific ports. In this manner one can instruct **review** to treat sessions to port 1234 on host X as IRC sessions, which will be treated just like any other IRC session.

FTP and IRC sessions often include special commands that give directives to the client or server to create additional sessions. For example, a client using FTP to down-load a file sends the port number the server must use to connect back to the client when it creates its data stream. Clicking on that **port** message in the browsing window creates another browsing window viewing the associated data session. IRC uses a similar protocol to establish connections between IRC clients for private conversations or file transfers. These can also be selected to create browsing windows for those sessions.

It is often convenient to save the contents of a session to a file for additional analysis. The **save** button brings up a dialogue box that allows the user to specify which side(s) of the traffic to save, whether to save the printable ASCII or literal packet contents, and where the data to be saved. If the literal contents are selected **review** will reassemble the TCP packets in the correct packet order, discard duplicates and extract the data payload of the packets. We have often found that runnable programs, unpackable zip or tar archives and viewable images can be reconstructed in this fashion from the packet logs.

Protocols such as SMTP, POP, FTP and IRC are easy to read since they typically restrict themselves to printable ASCII characters and the protocols use human readable constructs. Sessions using protocols such as Telnet, Rlogin and X Windows present additional challenges.

For Telnet and Rlogin one can readily view (and often understand) the client to server traffic, which is the text the user typed (Figure 1). This typically includes the commands the user tried to run at the server side, the text that they wrote in mail messages they sent and so on. The server to client traffic is harder to interpret. If distilled to printable ASCII it is often reduced to fragments of text mixed with fragments of the escape sequences used to control the display (Figure 2). If the output is not distilled to printable ASCII the text is often totally obfuscated by the escape sequences. In either case if the user entered some "full screen" application such as **vi**, **pine**, **pico** or **emacs** it can be nearly impossible to tell what they were doing since one cannot see what they would have seen on their screen.

X Windows traffic is even harder to interpret. One cannot easily discern the keystrokes that the user typed since they are sent from the server to the client as key press events where the actual characters typed are encoded with a key mapping table (Figure 6). One needs the applicable key map from the session and code to interpret the X Windows key press event structures to decode the events. The printable ASCII text could be pulled from the requests that the client sends to the server to paint the screen image, but the text tends to be fragmented even more than is the case for Telnet or Rlogin traffic since X Windows applications often use direct (x, y) coordinates to position new text, rather than space or tab characters (Figure 7).

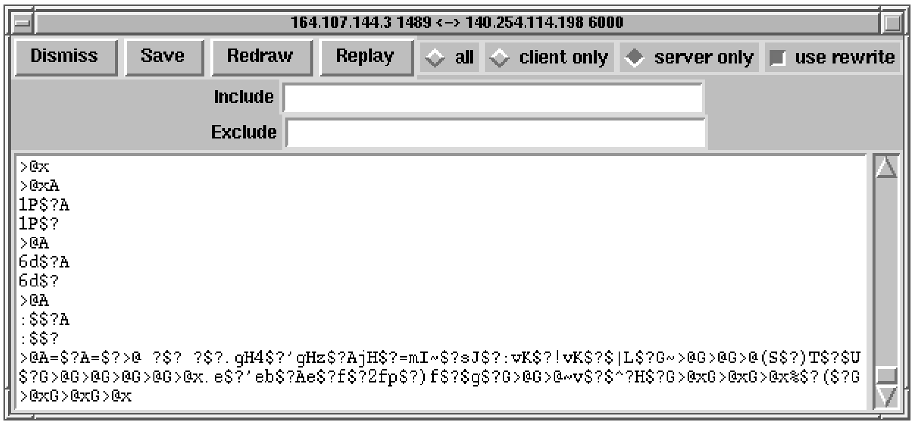

Figure 6: X Windows server to client traffic (replies and key press events – what the user typed).

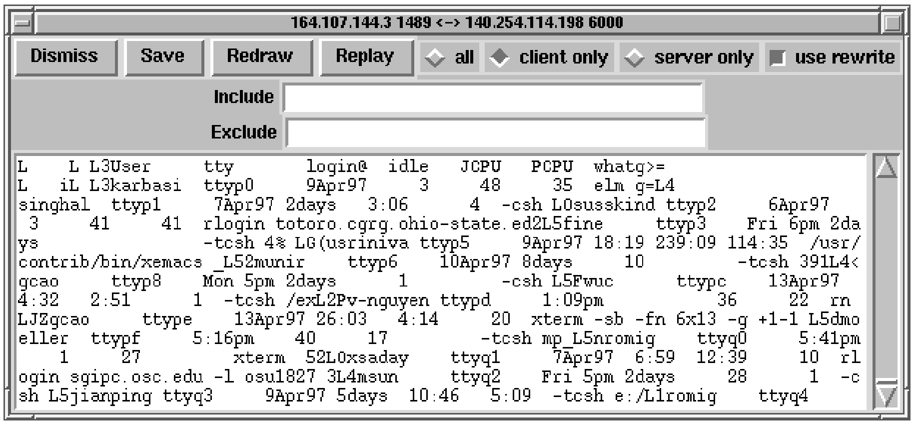

Figure 7: X Windows client to server traffic (what the user “sees”).

In the case of Telnet, Rlogin and X Windows sessions the **replay** button causes **review** to create session replay windows (Sections 3.4 and 3.5).

As with the other windows the **dismiss** button deletes this session browsing window.

3.4 Telnet/Rlogin Replay

When the **replay** button in a Telnet or Rlogin session is pressed **review** selects the packets sent from the server to the client, reorders them into the correct order, removes duplicates, extracts the literal packet contents and send that at a user controllable rate to an **xterm** window emulating a vt100/ANSI compatible terminal (Figure 8). The user controls the rate at which data is displayed by pressing the space key to read a chunk of data from the converted session and write it to the xterm window. As data is read from the session and written to the terminal emulator the viewer sees a replay of what the cracker would have seen on their screen, allowing the user to effectively view even “full screen” programs like **pine**, **pico** or **emacs**. Other keys allow the user to change the read size, rewind the replay buffer or to quit the replay window.

3.5 X Windows Replay

The X Windows replay tool is similar in intent to the Telnet/Rlogin re-player. The X Windows replay tool is a modified version of **xmond** [5]. The original **xmond** was designed to be an X Windows debugger. One sets it up as a proxy server pointing at a real server and then aims client applications at the **xmond** proxy. **Xmond** reads the requests that the clients send to the server and the replies, errors and events that the server sends to the client and displays a summary of the traffic as human readable text in the shell window it was started from.

We have modified **xmond** so that in addition to its normal functions it creates a “replay window” and replays certain safe X Windows requests sent from the client to the server in that window, showing the user a representation of what the client was trying to draw on the server (Figure 9).

Figure 8: A Telnet replay window. Compare this to Figure 2.

As with the Telnet/Rlogin replay, the X Windows replay feature extracts the client and server sides of the traffic from the log, reorders the packets, suppresses duplicates, extracts the literal packet contents to reconstruct the client and server data streams, and feeds these to **xmond**. **Xmond** is written to pause before executing "destructive" operations that might erase part of the screen (such as any clearing operation). To advance the replay the user simply hits the space key in the replay window.

This works surprisingly well for simple X Windows applications but more work is required to adequately handle more sophisticated X applications (Section 6).

3.6 Log Pre-Processing Features

Tcpdump logs of around 20 to 30 megabyte occur frequently in our experience and longer logs of 100 or more megabytes are not unheard of, especially when the crackers have been playing Quake. Using the **review** program to view such logs can be time consuming since to view each session contained in the log **review** has to process the entire log because packets for each session could be scattered anywhere within the log.

Running `review -summary` on a set of log files will create log summary files which future invocations of **review** use to create the session summary windows (Section 3.2). Doing so in advance saves time when one views the logs since **review** will not need to read the entire log to create the summary. One can also extract and cache "interesting" sessions from the logs using `review -cache`. When the user selects a session to browse **review** looks for a cache file for that session. If found, **review** reads the session packets from that cache file rather than from the (possibly long) original log which decreases the time it takes to process session display requests.

Review has a built in table that describes interesting sessions that are

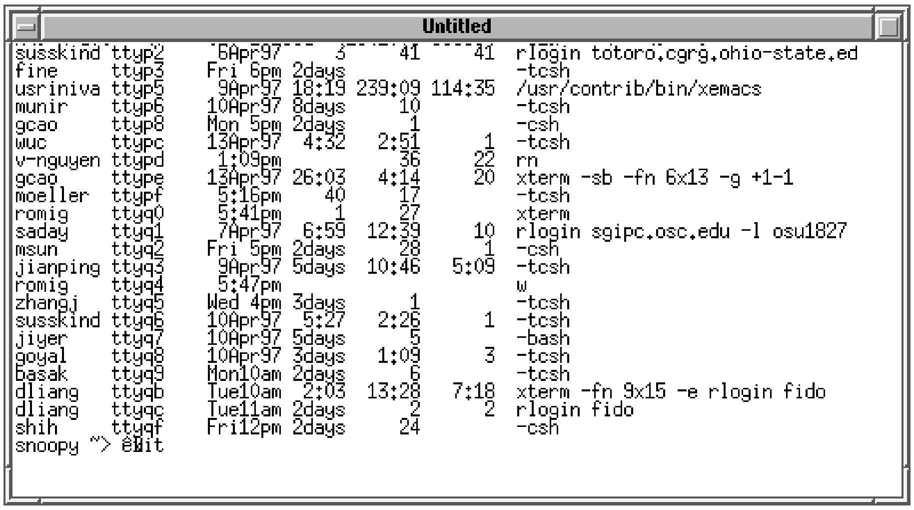

Figure 9: An X Windows replay window. Compare this with Figure 7.

cached. This currently includes Telnet and Rlogin sessions, X Windows sessions, SMTP and FTP sessions. POP and world web web sessions are not cached since they are typically numerous and expensive to process in this fashion. If a log contained 60 POP sessions (not uncommon since some mail client programs automatically check for new mail periodically) creating the cache for these would require reading the whole log 60 times. The table of interesting sessions can be extended using the same configuration file described in Section 3.3.

4 Collecting and Processing Packet Logs

In some cases we collect packet logs "by hand" by starting **tcpdump** manually on a collection host that can spy on cracker network activity. For example, we have done this on occasions where a cracker was repeatedly breaking in to a particular host despite our efforts to stop them. We collected packet logs covering the next break in and then read through them to see how they did it.

In most of our recent cases the crackers are coming through our labs or dialup modem pool using different IP addresses which makes it hard to set up **tcpdump** somewhere to catch their network activity since we cannot predict what address they will be using in the future and so cannot direct **tcpdump** to capture their network traffic.

We have written tools that watch the authentication logs for our dialup services or labs (computers in our public labs do not have network access until someone authenticates against an authentication server). These tools read a configuration file that lists the compromised accounts and the actions to take when that account logs on or off. For example, we can specify that we should be paged when accounts "steve" or "bill" log on or off, and that we should be paged and that network logging should be turned on or off when account "alice" logs on or off. To start or stop logging, the tools get the target IP address (which is dynamic for SLIP or PPP connections through our modem pool, static for connections in our lab), consults a table to determine whether there is a suitable logging host somewhere which can be used to capture that traffic, and if so, sends it a command to start or stop tcpdump with suitable filters to capture only that traffic. In this fashion we automatically collect packet logs for many of the people who have compromised accounts on our systems.

Someone in our network security group periodically collects the completed **tcpdump** logs from the logging hosts and processes them to collect information about cracker activities. When we retrieve a new set of logs from the logging hosts we run `review -cache` to create summary and cache files to improve the performance when we get around to reviewing the logs (Section 3.6).

5 What We Have Learned From the Logs

When we read through the logs we are looking especially for two sorts of information: who the crackers are, and what they are doing. Learning who they are does not necessarily help us arrest or prosecute the perpetrators directly since we need to go through the established trace/trap and dial number recorder procedures to collect evidence. However, what we learn about their identities can be used to confirm what we find through phone tracing or other avenues, and can provide important leads that link these cases to others.

Learning what they are doing is of vital importance – in many situations the only way we can find out what they have broken into and how they did it is to actually watch them do it. This part of our log analysis is especially important given that most incidents go unnoticed. When we detect intrusion activity from the logs and report this to the affected sites, in most cases our warning was their first indication that an intrusion had occurred. The logs contain enough details that we can usually tell affected site exactly which files have been modified (and how), which have been deleted, and so on.

We have had great success in recovering some of the tools that the crackers use in their activities. When they download them through IRC or FTP, edit them, view them, or mail them we can often extract complete copies from our logs. Although our packet capture tools drop a certain percentage of the packets transmitted, that is offset by the fact that most of the phone connections are noisy enough that there is a high retransmit rate for their network traffic so if the packet logger missed a packet the first time it might still see it again.

We have also learned a considerable amount about the local cracker culture and society – who is doing what, which crackers hang out together, and so on. We have seen that they communicate effectively with one another and cooperate amazingly well together to "beat the system". We have been able to watch them teach one another and practice using new exploit tools or techniques, often using each other's hosts as practice targets.

Although some of our crackers have started to use encryption tools such as **pgp** (pretty good privacy) or **ssh** (secure shell), this has not significantly impaired our ability to gather intelligence about them. For one thing, we gain most of our information about their identities and such through their IRC sessions, which would be more difficult to encrypt due to the key exchange problems. Their intrusion activity is never encrypted, since they typically break in through non-encrypted avenues such as web or Telnet servers. Finally, most of the crackers have not been careful with their passwords for accounts, their private keys, or the pass phrases for those keys. We have most of these in our logs from when they were typed across unencrypted links or transfered from one system to another. We should be able to decrypt many of the encrypted sessions if we ever felt the need.

6 Future Plans

We have many plans for our future versions of **review**. Here are some of the more interesting.

Review currently does all of its I/O synchronously which means that when it has to run **tcpdump** to extract packets and then convert the packet contents to readable form the whole program is hung up until that processing is finished. Converting **review** to use asynchronous I/O would make it more responsive when slow background tasks are occurring in other windows and will allow us to eventually create a version that can be used for monitoring network traffic in real time.

We currently gather most of our intelligence “by hand” by reading through the logs with **review**. While **review** makes this much easier and considerably faster than it used to be there are some parts that could be automated.

We plan to write scripts that scan the logs and generate reports for each session describing the network traffic seen in each (e.g. 3 Telnet connections to site X, 4 FTP connections to site X and Y, 1 IRC session, 82 web sessions), and giving a summary of activity such as web pages browsed or files transferred with FTP. We also plan to look for special events like host and port probes, **phf** probes, or other suspicious activity in a fashion similar to what TAMU's **netwatch** does.

Although the include/exclude features can be used to focus on output that matches certain patterns, it would also be good to be able to search for strings or regular expressions in the session summary and session display windows. It would also be nice to be able to “search” in the Telnet or X Windows replay windows, since we frequently find that we want to advance past the next several hours of recorded IRC babbling in a Telnet window to the point where we think that they broke into another host.

The X Windows replay features are useful, but need more work. The X Windows re-player uses a single window for all output. If the session being replayed is from an application that created and used multiple windows then the requests that would have been made in different windows will be drawn on top of each other in the single replay window, confusing the display. Similarly, the X Windows re-player uses a single graphics context for all of its graphics. Some applications create and use multiple graphics contexts, and the current X Windows replay code does not handle these well. This is especially true when the application uses the different graphics contexts to erase window contents that had previously been drawn or to draw objects in different colors or line styles.

We plan to change the X Windows replay program so that it follows CreateWindow and CreateGC (create graphics context) requests by creating windows or contexts with the same properties on the current display. We can make a table mapping the old references (window or graphics context identifiers) to the current references. As we replay requests from the session's packet stream we can map the old references to current references, which means that requests would be played into separate windows of the appropriate sizes and shapes and using similar visual appearance as they originally had. This would allow much more faithful reproduction of what would have been seen on the original display.

We would also like to integrate these tools with other tools that handle other aspects of incident response. For example, we currently maintain a catalog of packet log files and for each log file we record which cracker was responsible for the activity and what activity we observed, or what information we obtained from that log. This is currently done using a text editor on a separate file. It would be more convenient to integrate this into **review** itself so that when viewing a log or a session within a log the user can more easily update the associated documentation with their observations. We are also working on tools for handling other logs, such as our various authentication logs, the logs we obtain through the phone company in phone traces, the logs we obtain through tools such as dial number recorders, and through network traffic logs, such as the flow logs we obtain from our Cisco routers. Ideally one would be able to jump from a report in our incident database, to the **tcpdump** log for the IP address used in the break in, to the authentication log showing what account was used to dial up and which terminal server and port number, to the dial number recorder log showing that person X Windows made a call to our dial up modem pool that spanned the same time period, and then to the flow logs that confirm the network activity recorded in the **tcpdump** logs.

We think that visualization tools such as these could be especially useful in court cases where it may prove otherwise difficult to make the massive volume of information readily accessible or understandable to non-experts.

7 Concluding Remarks

The information we have gathered through packet logging has proven invaluable to us. We have learned who the crackers are in real life, which has helped us both in linking our incidents to others and also in confirming what we have learned through other avenues such as phone traces. More importantly, we now know exactly what they are doing when they attack our computers or use them as staging areas to launch attacks on other systems across the Internet. We have detected many intrusions that might otherwise have gone unnoticed and have been able to follow up more effectively on some other incidents which were reported to us, since we later found evidence implicating "our" crackers in the packet logs. This information would have been difficult, if not impossible to collect before we wrote **review**, due to the complications in extracting, processing and viewing packet contents for the hundreds of login sessions we have logged.

The practice of logging full packet contents raises important legal and ethical concerns. In the U.S., the Electronic Communications Privacy Act [3] (ECPA) makes it illegal to log and view electronic communications except under certain very specific situations. Our legal counsel agreed that in situations where we know that someone has compromised the security of our computers and where we can be fairly assured of logging their activity and not the activity of our legitimate network users such logging is probably allowed under the ECPA. The ECPA has not been widely tested in court however, so we cannot say with certainty where we stand. The laws in other countries differ widely, of course. We strongly urge others who are considering using full packet logging to consult with a legal counsel to discuss the issues and determine whether and how to proceed.

The full sources and documentation for **review** and its associated log gathering tools are available upon request. Requests should be made by mail to security@net.ohio-state.edu. The current version is a collection of programs written in C and Perl. It requires **tcpdump**, Perl 5.003 or higher, and the Perl/Tk module. It is known to run under SunOS 4.1.4, BSDI 2.1, FreeBSD and Linux using X11R6. It should be readily portable to other UNIX platforms.

8 Acknowledgments

Michael Neuman's **ip-watcher** program was a real eye-opener, and provided some of the initial inspiration for this work.

Mark Fullmer wrote the **cleanup** program used for extracting the contents of **tcpdump** hex packet output. We have extended his original program so that it can also emit ASCII text with time stamps or the literal, reordered packet contents with duplicates suppressed.

Greg McFarlane is the author of **xmond**, which we extended to replay X Windows sessions.

Clifford Collins, Justin Dolske, Tom Fine, Mark Fullmer and John Hogue of The Ohio State University provided suggestions, support, inspiration and clues. Tom, Scott King, and Frank Adelstein made valuable comments on earlier drafts of this paper.

And special thanks to our crackers for providing continued motivation to write tools like this.

References

[1] Brecht Claerhout (coder@reptile.rug.ac.be). The **sniffit** software package. See `http://reptile.rug.ac.be/čoder/`.

[2] Douglas Lee Schales David R. Safford and David K. Hess. The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment. In *Proceedings of Fourth USENIX UNIX Security Conference*, available by anonymous ftp from coast.cs.purdue.edu in `/pub/tools/unix/netlog`, 1993.

[3] U.S. Federal Statute, Title 18 (Crimes and Criminal Procedure), Chapter 119 (Wire and Electronic Communications Interception and Interception of Oral Communications), Section 2510/2511 et seq.

[4] EnGarde Systems, LLC, `http://www.EnGarde.com/`. *IP-Watcher v1.2 User's Guide*, 1996.

[5] Greg McFarlane (gregm@nms.otc.com.au). The **xmon** software package. Available by anonymous ftp from ftp.cdrom.com in `pub/X11/contrib/devel_tools`.

[6] Mark Fullmer (maf@net.ohio state.edu). The **cleanup** software package. Available as part of the **review** package - send email to security@net.ohio-state.edu.

[7] Michael Ryan (mike@networx.ie). The **tcpshow** software package. Available by request from the author.

[8] Craig Leres Van Jacobson and Steven McCanne. The **tcpdump** software package. Available by anonymous ftp from coast.cs.purdue.edu in `/pub/tools/unix/tcpdump`.