Passive TCP reconstruction and forensic analysis with tcpflow*

Simson L. Garfinkel
Naval Postgraduate School
900N Glebe
Arlington, VA
slgarfin@nps.edu

Michael Shick
Naval Postgraduate School
900N Glebe
Arlington, VA
mfshick@nps.edu

ABSTRACT

Passive TCP session reconstruction essential for many kinds of network forensics and law enforcement operations, but it is is complicated by packet loss, retransmissions, and possible attacks by adversaries. The key problem is that participants in the TCP session may observe the TCP segments differently than the monitor. An Added complication is the lack of familiarity with network protocols by many forensic analysts, resulting in the need for tools that are easy-to-use and able to tolerate a wide range of data. To address these issues we rewrote the open source network forensics tool *tcpflow*, making it more robust to anomalies that had been reported to us by users. We also improved the program's usability and performance on large packet captures, and added simple visualization that produces a one-page summary PDF for packet captures of any size.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network monitoring; C.2.5 [**Local and Wide-Area Networks**]: Internet (e.g., TCP/IP); D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

TCP reconstruction; network forensics

Keywords

tcpflow, tcp/ip, visualization

1. INTRODUCTION

Law enforcement and cybersecurity investigators are increasingly monitoring networks as part of both ongoing investigations. For example, some organizations capture packets in and out of their networks and analyze the packets relating to a particular IP address if that host becomes infected with malware. Such *network forensics* efforts cannot be performed with netflow data or truncated frame headers, since investigators require access to the full network content. Investigators also require efficient tools that can reassemble TCP segments into *transcripts* that contain a byte-for-byte copy of the data delivered to the network application programs. These transcripts are then viewed, searched, and subjected to additional processing by both human examiners and automated tools.

Many technical factors complicate the process of working with full content captures. Unlike traditional network monitoring, statistical sampling cannot be used, as the event sought by investigators may be a single TCP connection—or even a single IP frame. Because most equipment subject to monitoring is not under the control of the investigator, there is typically no way to regenerate or replay targeted events. Because forensics is invariably performed with passive network taps, packet captures are frequently asymmetric or incomplete; it is not uncommon for investigators to capture only packets sent in one direction.

Packet captures are typically delivered to investigators in the form of pcap files. Yet investigators frequently lack the tools and technical skills to analyze these files[13]. Wireshark[18] is the most popular network protocol analyzer today[14, 23]. But Wireshark will only reassemble a single TCP stream at a time, and its string search capability will only find strings contained within a single packet, and only when the stream is not compressed or otherwise encoded. This is a problem, given that web servers are increasingly using gzip compression to deliver HTTP pages.

This article presents some of the requirements for passive TCP reconstruction and other aspects of modern network forensics and shows how we addressed them in *tcpflow*, an open source network forensics tool that was initially developed by Jeremy Elson in 1999, abandoned in 2003, and largely rewritten by the authors of this paper over the past four years. This article's contributions include an explanation of why the requirements for Internet measurement tools used for forensics differ from other uses; details of how we implemented those requirements in *tcpflow*; and a description of our new vi-

*Submitted to IMC 2013. This work supported in part by NSF DUE-0919593 and DUE-0919593.

sualization for packet captures. From the point of view of our users, however, it is the usability of our visualization that is key: it is available today, implemented in open source software, and runs on Windows, Linux and Macs.

2. RELATED WORK

Corey *et al.* described the requirements for a network forensic analysis tool (NFAT) in 2002[4]. Casey reviewed other tools in 2004[3]. Pilli *et al.* surveyed NFAT frameworks and presented research challenges in 2010[23], identifying the largest research challenges as collection of data and detection of attacks; data fusion and examination; analysis; investigation; and incident response. Surprising to some, virtually every digital forensics technical problem or requirement identified over the past decade remains with us today. Computers are faster and hard drives are larger, but bandwidth, the number of users, and the complexity of data have also increased. In many ways forensics is harder now than it was in 2002 due to the increasing diversity of devices, protocols, data formats, relays, and the increasing use of encryption[9].

Dharmapurikar and Paxson[5] presented a hardware-based approach for reassembling TCP streams in the presence of adversaries. Their requirements are different from ours, in that their system is designed for an inline NIDS that must make real-time decisions as to whether or not to pass individual packets. Because their system is inline, it is guaranteed to observe both directions. Another key difference is the adversary model: Dharmapurikar and Paxson assume an adversary that must participate in TCP handshake and is attempting to exhaust the memory space or memory bandwidth of the monitor. Like Hahn *et al.* [12], we are additionally concerned with adversaries that use overwriting attacks, possibly with changing TTLs, so that the monitor assembles a different TCP transcript than the target system

Passive TCP stream reassembly is implemented in many network intrusion prevention and detection tools (e.g. Bro[22] and Suricata[27]). The requirements for these systems is subtly different from ours, as an IPS can simply terminate connections that it cannot reassemble. Likewise, many of these systems were designed to work with flows in a manner similar to that of the Windows or Linux stack. For example, Wojtczuk asserts that "libnids predicts behaviour of protected Linux hosts as closely as possible."[29] The challenge is that reorderings observed by the monitor may not the same as those observed by the participants: in is occasionally necessary to make a best-effort attempt to recover the stream content even if a well-behaved TCP implementation would not. This is particularly true when the monitor receives data segments before the first SYN or after a FIN or RST. We have examined the source code Bro and libnids and found that they will not reassemble streams that have such reorderings.

Another challenge for passive TCP reassembly is the possibility of an attacker mounting a denial-of-service attacks on the NIDS CPU; Dreger *et al.* discuss this in more depth[6].

Much of our effort has been aimed at developing a command-line tool that is easy to use by examiners and easy to integrate with automated processing. For example, *tcpflow* can create a few million transcript files in a few thousand directories; these transcripts can then be searched with *grep*, viewed with *emacs*, or analyzed with other programs. We originally implemented this approach based on personal preference; Botta *et al.* [2] studied security practitioners and found that generic text and information management tools such as `grep` offer significantly usability and performance advantages over integrated GUI-based systems for experienced administrators.

Users of our tool requested a simple one-page visualization to provide an overview of the pcap file being analyzed. There are more than two decades of work on the visualization of network capture data and we cannot summarize it here. Much of this work has been to develop interactive visualizations for use by analysts, but there has also been some work on one-page displays (*e.g.* [1, 24, 26]). Estan *et al.*'s Autofocus[7] attempted used a clustering algorithm to identify and then visualize the most significant parameters of a resource-consuming network flows. Palomo *et al.* explored the use of hierarchial SOMs for visualizing network forensics traffic data[21]. Our work differs from these in part because our visualization is designed to produce consistent imagery no matter whether it is given a hundred packets or a hundred million, making it possible to directly compare different graphs from different captures.

An alternative to TCP reconstruction is to use a proxy such as the Charles Web Debugging Proxy[28]. This approach has the advantage of perfect capture, but the disadvantage of requiring network reconfiguration.

The sketch data structure has been widely used to tally counts in high-performance network monitoring systems[8, 16, 25]. Unfortunately sketches introduce probabilistic errors, so we avoid them and developed an approach based on radix trees instead.

3. REQUIREMENTS

In 2008 NPS created a network forensics analysis exercise that required the ability to perform web browser identification to overcome scrambling from network address translation. Frustrated by Wireshark, we developed a solution that involved creating a separate Unix file for each TCP session, using `grep` to find the TCP session associated with the fictional criminal activity,

examining the TCP session with `more` to find browser-identifying strings, and then using `grep` a second time to find all of the other TCP streams associated with that browser. Those streams were then manually examined for information that identified the perpetrator.

Since then, we have maintained *tcpflow* as a freely available open source tool for digital forensics education and demonstrations. The tool's primary requirement is to ingest pcap files and produce a separate transcript file for each side of each TCP session. Examiners then analyze these files using existing tools. *tcpflow* has also been incorporated into automated analysis systems. Over the years we have been contacted by a variety of practitioners that have used to tool in their work and have required additional features. Here we present some of those requirements.

3.1 Ease of Installation and Use

The primary requirement we have observed among practitioners is that tools be easy-to-use. Surprisingly, ease-of-use frequently trumps performance, functionality and even correctness: practitioners will use tools that lose data and occasionally produce incorrect results in preference to other tools that are technically superior but beyond the practitioner's ability to compile, run, or interpret the results.

Although such attitudes may seem at odds with the legal requirements for courtroom admissibility, in practice few cases actually go to trial, few trials involve the direct presentation of digital evidence, and few defense attorneys have the technical sophistication to dispute the correctness of a tool[20]. Nevertheless, we believe it is important to have tools that are both easy-to-use and technically correct.

3.2 Content discovery and attribution

For many practitioners the goal of a network forensics investigation is to identify content within a collection of packets and determine what was sent, who sent it, and where it was received. Most examiners therefore perform offline analyses. The tool need not run at wire speed, but it must be able to handle any amount of data with consumer hardware within a timeframe consistent with the investigation.

Many users have an additional requirement that objects sent by HTTP or email be extracted from the pcap file and saved in the file system as individual files with descriptive filenames and appropriate extensions. ZIP and gzip-compressed HTTP streams must be decompressed. In many cases it is helpful to compute the MD5 hash value of each object sent by HTTP so that the hashes can be compared with a database of malware or known content.

3.3 Live Analysis

We were surprised to discover that a significant number of *tcpflow*'s users were not performing forensic analysis, but were using the program for a variety of other purposes, such as protocol debugging and art projects. These users relied on *tcpflow*'s ability to perform real time capture and reassembly of data sent by HTTP and wrote their own post-processing software to ingest the resulting transcript files. Thus it was important that all aspects of the program continued to support live analysis.

3.4 Resilient Passive TCP Reassembly

TCP is an interactive two-party protocol designed to reliably transport byte streams across a best-effort network that may occasionally corrupt, drop, duplicate or reorder frames. These mechanisms rarely help a passively monitoring third party. Instead, the monitor must accept all frames that are received and attempt to make the best possible use of the data.

Additional complications include: 1) The monitor may miss any frame. In the extreme case, asymmetric routing may result in the monitor capturing only one side of the connection; 2) The monitor may capture frames sent but not received; 3) The monitor may capture frames not sent but received, or neither sent nor received. (In both cases, such frames may be sent by an attacker with the goal of disrupting the monitoring.)

Analysts are often unable to diagnose network capture problems—or are simply uninterested in doing so. They need tools that work with the data that they have on the hardware that's available.

3.5 Simple Graphical Reporting

Several of our users wanted a way to quickly review the contents of gigabyte sized pcap files and answer questions such as: *What kind of data were collected? Is there a time gap? Which protocols were in use? How much of the data is to/from the target hosts?* Although there has been substantial work on TCP visualization, most of this work has been research and has not produced graphs that are readily easy to interpret without training, and few have been operationalized into tools that are available to examiners.

4. IMPLEMENTATION

With version 1.4, *tcpflow* has been rewritten in C++ with an automake-generated build system and restructured to use the plug-in API that we have developed for *bulk_extractor*[11], a high-performance open source computer forensics triage tool.

4.1 Support for Existing Forensics APIs

Essentially, *tcpflow* is now a framework that loads plug-ins and then reads one or more pcap files. Each plug-in can register to process captured frames or re-

assembled streams. The passive TCP is implemented by the *scan_tcpdemux* plug-in, which reassembles the TCP streams and submits the streams to the framework for further processing. Although the current version could use *bulk_extractor*'s *scan_base64* and *scan_gzip* to extract and decode HTTP and email MIME objects, we instead opted to write a new plug-in that uses the Joyent http_parser[15] to decode and extract the embedded objects. These objects are then recursively reanalyzed using the framework.

We also provide a *scan_md5* scanner to calculate the MD5 of each stream, and a *scan_netviz* module (§4.4) to create a visualization of the processed packets. The plug-in architecture makes it easy for organizations to add their own functionality with internal, non-public modules.

tcpflow was adapted to output a list of every connection using Digital Forensics XML[10]. Despite DFXML's verbosity, the format makes it relatively easy to associate each flow with the specific file used to hold the transcript. The DFXML file can be readily augmented with the MD5 hash value the flow's contents, and can readily represent objects that are extracted from HTTP flows, even when the extraction requires decompression or other forms of decoding. DFXML allows post-processing of flows with existing disk forensic tools.

Using the *bulk_extractor* framework minimizes the training required for programmers to switch between the projects, and helps assure that the tool will be maintained by the very limited development capabilities in the computer forensics community.

Finally, we have added numerous convenience features to *tcpflow*, such as the ability to read frames directly from gzip and ZIP-compressed files.

4.2 Passive TCP Reconstruction

The *scan_tcpdemux* module implements a passive TCP that maintains a separate TCP state machine for every flow observed. As a result, no special code is required to reconstruct TCP connections for which only one direction is captured. Flows are identified by their five-tuple.

As the missed segments may include one of the segments from the initial three-way handshake, *tcpflow* creates a new state machine on the first segment it receives of any TCP flow. If the first segment does not have the SYN bit set, the initial sequence number (ISN) is assumed to be one less than that segment's sequence number. In testing we discovered that a segment several segments into a connection might be the first to be observed by the monitor, followed several seconds later by retransmission of segments earlier in the connection (but still without the initial SYN). To handle this situation, *tcpflow* computes a new ISN whenever a segment is discovered before the ISN and inserts the bytes from that segment at the start of the transcript file, shifting the rest of the content towards the rear to make room. The current implementation performs this shift on disk, which is an expensive operation, but it avoids the need to buffer the data for all open connections in memory, which offers protection against so-called "elephant flows." Such huge flows can be further defended against by setting *tcpflow*'s *-b max_bytes* parameters.

The passive TCP performs further processing on TCP connections when a FIN or RST is received, but only if all bytes in the connection are present in the transcript file. Otherwise the TCP state machine is kept active in memory until either the retransmitted segments are received or a user-configurable timeout is reached. *tcpflow* distinguishes resent frames that have different content by comparing the frame's content with the transcript file. Segments received after a FIN or RST are also processed, as segment may be an attack sent to the monitor and not observed by the participants.

The transcript filenames can be built from IP addresses, ports, connection numbers, and other information; the pattern can be specified on the command line. Three built-in patterns will automatically bin connections by connection number in one, two or three directory layers, assuring that *tcpflow* will put not more than 1000 transcripts in a directory while allowing the program to process 10^6, 10^9 or 10^{12} individual connections with two, three or four directory levels. (With 10^{12} files, the examiner would hopefully use some kind of high-performance cluster file system for analysis.)

4.3 Defending Against Attacks

It is relatively easy to attack a passive TCP using maliciously crafted TCP segments.

Consider a four segment sequence with a A SYN-ACK segment (ISN of 1); a 300-byte segment (seq of 2); a 300-byte segment (seq of 2^{30}); a FIN (seq of $2^{30}+1$). Reading these segments, many passive TCPs will create a gibibyte-sized transcript filled with zeros; *tcpflow* has a user-settable *[-m maximum gap];* gaps detected that are larger result in the current transcript being terminated and new one started.

Another attack is to send segments that appear to come from every possible IPv4 address, which will cause any associative array keyed on source IP address to consume a significant amount of memory. We defend by shrinking the `tr1::unordered_map` as necessary to free memory pressure and through the use of the IP address radix tree, discussed in section 4.5.

4.4 A Useful 1-Page Visualizations

In response to user requests, we created a plug-in that would produce 1-page PDF file that would summarize the processed capture.

Our users have a relatively simple problem that was unmet by existing visualization tools: they are frequently

provided with one or more pcap files that they need to be able to validate before passing on for sophisticated analysis (a process that may be backlogged). Our users needed to know the time span that a network capture encompasses, whether or not there are holes, the hosts and the protocols involved. The result must be suitable for inclusion in a printed report, and different examiners must produce the same visualization when given the same data.

We address these requirements with a single-pass visualization that shows a series of bar graphs showing data received over time, the top senders and receivers by IP address and by TCP port. To address varying time scales we build histograms at multiple time resolutions (minute, hour, day week, month, year, decade, and semicentury), with each bucket being a map with slots for corresponding to each protocol port. We use the same color scheme for all graphs, so that the bar graphs showing popular ports is a key for the time-based histogram. A cumulative distribution function is drawn across all bar graphs, with the line in a layer that is over the bars but behind the text labels for maximum legibility. The result is a visualization that is completely automated, conveys information that is case-relevant, and can be generated at little computational cost (The visualization of the sanjose capture shown in Figure 1 involved roughly 48 million packets and required 562 seconds of clock time and 72MB of RAM (measured by maxrss) on a Mac Pro running MacOS 10.8.3.) In a future version we hope to include a two-dimensional map showing host-pair communications and relying on the same color scheme.

4.5 IP Address Radix Tree

The two address bar graphs in the visualization use a sparse radix tree to store counts associated with individual IP addresses.

Each node of the tree consists of a counter and pointers to two nodes; IP addresses are encoded as the location of leaves on the tree. When the tree has more than a predetermined number of nodes, the tree is walked to find the deepest node with one or two leaves that have the lowest counts; the leaves are then pruned and their counts added to node's, which is now a leave. In this way hosts that have relatively little traffic are naturally combined into subnets, while hosts with lots of traffic are maintained with individual counts, even if they have adjacent IP addresses on the same subnet. (For example, note that the top source address in Figure 1 is 48.1.136.0/24.) While such summarization is not appropriate for flow analysis or a detailed report, we believe it is appropriate for our one-page visualization, where data reduction is essential. Added benefits are that the radix tree requires no configuration, and that IP addresses are combined into rarely used blocks with common prefixes, similar to CIDR blocks. Finally, a single tree can store both IPv4 and IPv6 addresses. It is also possible to store source-destination pairs in the tree by bit-interleaving the addresses, although more work is required to understand the behavior of our implementation before it is ready for general use.

5. EVALUATION

Figure 2 shows the visualization applied to one of the anonymized Internet traces from IPv6 Day available from CAIDA; for comparison, we also show the time-based histogram applied to the "outside" capture files from the MIT Lincoln Labs 1999 DARPA Intrusion Detection Evaluation[17]. Our website[1] shows the visualization applied to other data sets.

6. FUTURE WORK

The original *tcpflow* was developed at a time when computers had relatively small memories and, as a result, all connections are buffered on disk. A passive TCP that keep segments in memory until a connection is complete or memory is exhausted might significantly improve performance; currently such buffering is performed by the operating system.

We hope to make *scan_tcpdemux* a linkable library and implement the *libnids* interface, allowing direct comparison of speed and accuracy of the two passive TCP implementations. We plan to publish a set of pcap files that demonstrate specific failures of existing passive TCP systems that *tcpflow* handles properly.

Since many users of *tcpflow* are not comfortable coding in C++, we hope to create a gateway that will make it easy to process completed TCP streams in Python.

Finally, we suspect that additional profiling will allow us to make additional performance improvements to the passive TCP state machine.

7. CONCLUSION

We have created a high-performance passive TCP implementation and visualization designed to assist investigators with network forensics tasks. Rather than creating an integrated system, our approach turns captured packets into individual transcript files, after which they can be processed with other tools. The result is a system that makes it relatively easy for today's digital forensics practitioners to perform sophisticated network forensics tasks with relatively little training.

7.1 Acknowledgments and Disclaimer

Acknowledgements omitted for review.

The views expressed in this document are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

[1] `http://digitalcorpora.org/tcpflow/`

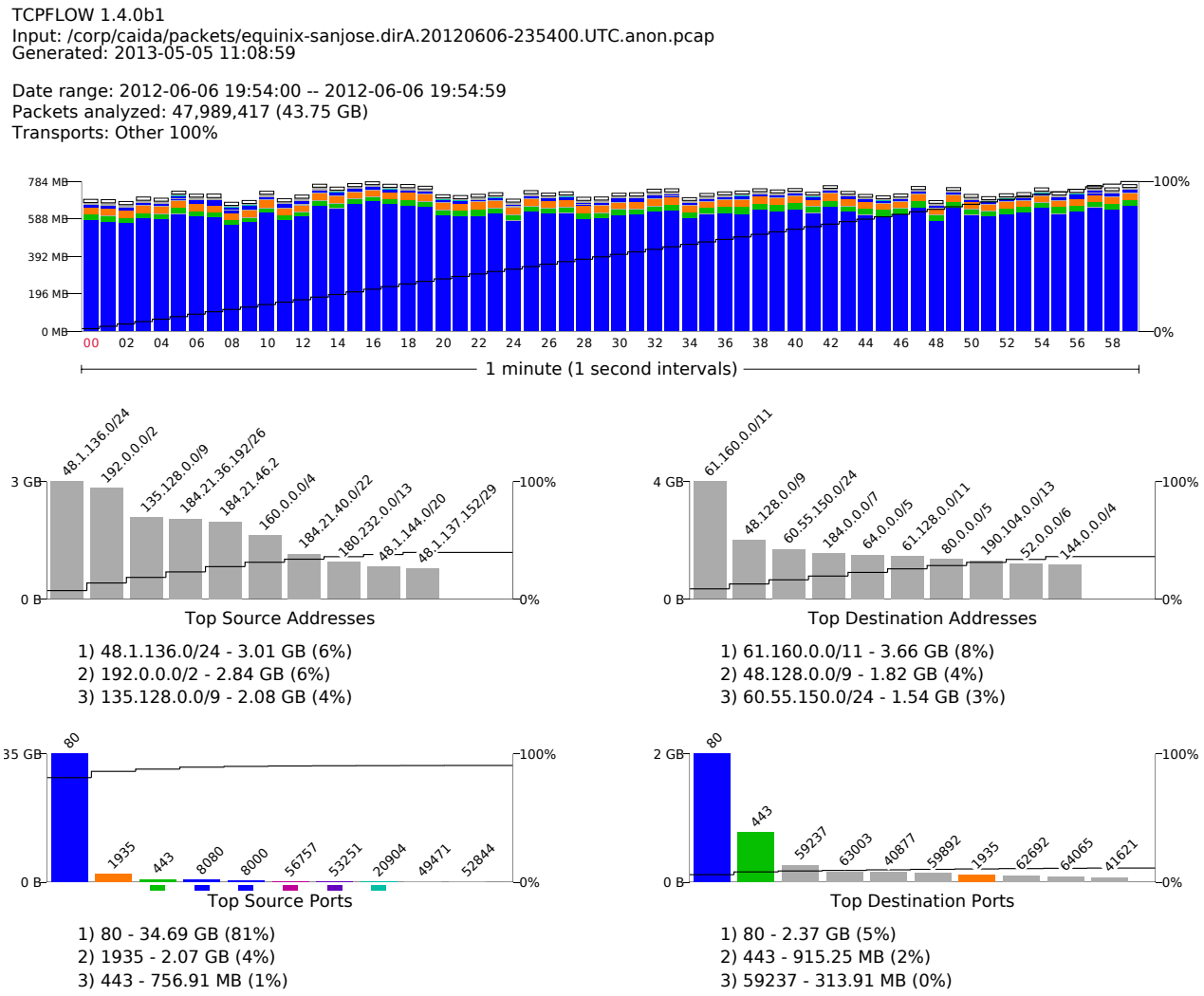

Figure 1: *tcpflow*'s one-page visualization. The color key for the timeline stacked bargraph is presented in the source and destination graphs. Each graph includes a CDF. Here we shows the CAIDA equinix-sanjose.dirA.20120606-235400.UTC.anon.pcap capture

---

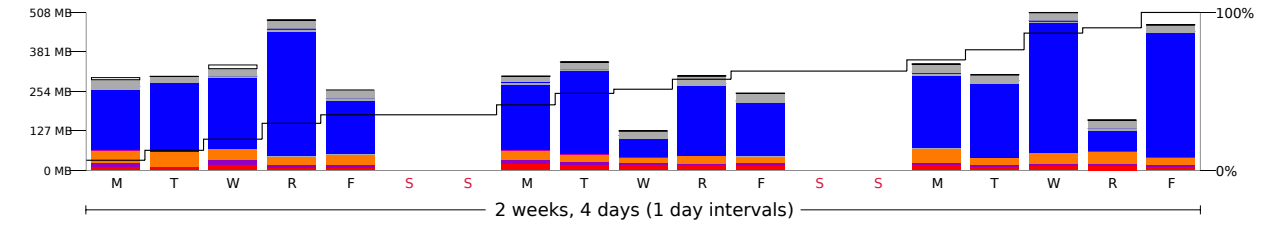

Figure 2: The timeline of the three week DARPA 1999 "outside" dataset[19] makes it easy to see some of the criticisms of the data—for example, the lack of traffic on weekends.

8. REFERENCES

[1] Richard Blum. *Network Performance Open Source Toolkit Using Netperf, tcptrace, NISTnet, and SSFNet*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2003. ISBN 0471433012.

[2] David Botta, Rodrigo Werlinger, André Gagné, Konstantin Beznosov, Lee Iverson, Sidney Fels, and Brian Fisher. Towards understanding it security professionals and their tools. 2007.

[3] Eoghan Casey. Tool review: Network traffic as a source of evidence: tool strengths, weaknesses, and future needs. *Digit. Investig.*, 1(1):28–43, February 2004. ISSN 1742-2876. http://dx.doi.org/10.1016/j.diin.2003.12.002.

[4] Vicka Corey, Charles Peterman, Sybil Shearin, Michael S. Greenberg, and James Van Bokkelen. Network forensics analysis. *IEEE Internet Computing*, 6(6):60–66, 2002. ISSN 1089-7801.

[5] Sarang Dharmapurikar and Vern Paxson. Robust TCP stream reassembly in the presence of adversaries. In *Proceedings of the 14th conference on USENIX Security Symposium - Volume 14*, SSYM'05, page 5. USENIX Association, Berkeley, CA, USA, 2005. http://dl.acm.org/citation.cfm?id=1251398.1251403.

[6] Holger Dreger, Anja Feldmann, Vern Paxson, and Robin Sommer. Predicting the resource consumption of network intrusion detection systems. In *Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection*, RAID '08, pages 135–154. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 978-3-540-87402-7. http://dx.doi.org/10.1007/978-3-540-87403-4_8.

[7] Cristian Estan, Stefan Savage, and George Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 137–148. ACM, New York, NY, USA, 2003. ISBN 1-58113-735-4. http://doi.acm.org/10.1145/863955.863972.

[8] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. *SIGCOMM Comput. Commun. Rev.*, 32(4):323–336, August 2002. ISSN 0146-4833. http://doi.acm.org/10.1145/964725.633056.

[9] Simson Garfinkel. Digital forensics: The next 10 years. In *Proc. of the Tenth Annual DFRWS Conference*, volume 7. Portland, OR, 2010.

[10] Simson Garfinkel. Digital Forensics XML. *Digital Investigation*, 8:161–174, February 2012.

[11] Simson Garfinkel. Stream-based digital media forensics with *bulk_extractor*. *Computers & Security*, 32, February 2013.

[12] Timothy Hahn, Michael L.H̃all Jr. Mohit jaggi, and Nicholas Leavy. Use of packet hashes to prevent TCP retransmit overwrite attacks, March 3 2009. US Patent 7500264.

[13] H. Hibshi, T. Vidas, and L. Cranor. Usability of forensics tools: A user study. In *IT Security Incident Management and IT Forensics (IMF), 2011 Sixth International Conference on*, pages 81–91. IEEE, 2011.

[14] Guillaume Jourjon, Salil Kanhere, and Jun Yao. Impact of an e-learning platform on cse lectures. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, ITiCSE '11, pages 83–87. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0697-3. http://doi.acm.org/10.1145/1999747.1999773.

[15] Inc. Joyent. http-parser, 2013. https://github.com/joyent/http-parser. Last access April 21, 2013.

[16] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, IMC '03, pages 234–247. ACM, New York, NY, USA, 2003. ISBN 1-58113-773-7. http://doi.acm.org/10.1145/948205.948236.

[17] MIT Lincoln Laboratory. DARPA intrusion detection data sets, 2000. http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/.

[18] Ulf Lamping, Richard Sharpe, and Ed Warnicke. *Whireshark User's Guide*. Wireshark Foundation, 2012. http://www.wireshark.org/docs/wsug_html_chunked/.

[19] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Comput. Netw.*, 34(4):579–595, October 2000. ISSN 1389-1286. http://dx.doi.org/10.1016/S1389-1286(00)00139-0.

[20] Committee on Identifying the Needs of the Forensic Science Community. *Strengthening Forensic Science in the United States: A Path Forward*. National Research Council, February 2009.

[21] E. J. Palomo, J. North, D. Elizondo, R. M. Luque, and T. Watson. 2012 special issue: Application of growing hierarchical som for visualisation of network forensics traffic data. *Neural Netw.*, 32:275–284, August 2012. ISSN 0893-6080. http://dx.doi.org/10.1016/j.neunet.2012.02.021.

[22] Vern Paxson. Bro: A system for detecting network intruders in real time. *Computer Networks*, December 1999.

[23] Emmanuel S. Pilli, R. C. Joshi, and Rajdeep Niyogi. Network forensic frameworks: Survey and research challenges. *Digit. Investig.*, 7(1-2):14–27, October 2010. ISSN 1742-2876. http://dx.doi.org/10.1016/j.diin.2010.02.003.

[24] Pin Ren, Yan Gao, Zhichun Li, Yan Chen, and B. Watson. Idgraphs: intrusion detection and analysis using histographs. In *Visualization for Computer Security, 2005. (VizSEC 05). IEEE Workshop on*, pages 39–46, 2005.

[25] Robert Schweller, Zhichun Li, Yan Chen, Yan Gao, Ashish Gupta, Yin Zhang, Peter A. Dinda, Ming-Yang Kao, and Gokhan Memik. Reversible sketches: enabling monitoring and analysis over high-speed data streams. *IEEE/ACM Trans. Netw.*, 15(5):1059–1072, October 2007. ISSN 1063-6692. http://dx.doi.org/10.1109/TNET.2007.896150.

[26] Timothy Jason Shepard. TCP packet trace analysis. Technical Report MIT/LCS/TR-494, Massachusetts Institute of Technology Laboratory for Computer Science, February 1991.

[27] Suricata : Open source ids / ips / nsm engine, 2013. http://suricata-ids.org. Last accessed April 21, 2013.

[28] Karl von Randow. Charles proxy, 2013. http://www.charlesproxy.com. Last Accessed April 21, 2014.

[29] Rafal Wojtczuk. Libnids, March 2010. http://libnids.sourceforge.net.