A Passive External Web Surveillance Technique for Private Networks

Constantine Daicos and Scott Knight

Royal Military College of Canada

Abstract. The variety and richness of what users browse on the Internet has made the communications of web-browsing hosts an attractive target for surveillance. We show that passive external surveillance of web-browsing hosts in private networks is possible despite the anonymizing effects of NATs and HTTP proxies at the gateway. These devices effectively anonymize the origin of communication streams, and remove many identifying features, making it difficult to group web traffic into mutually disjoint same-host sets called user sessions. User sessions offer a complete picture of each user's web browsing experience. Without them, passive external surveillance is of little use. This paper offers a content analysis technique called Link Chaining that aids the sessionization process by recovering large pieces of user sessions called session fragments. The technique is based on the knowledge that the majority of downloaded web resources are clicked-to from other web pages. By following hyperlinks in the bodies of HTTP messages in passively collected trace data, web traffic can be be coalesced into session fragments and used by human analysts to isolate individual user sessions. The technique gives the human analyst a significant advantage over manual methods. The implementation presented here has been tested on accumulated local data and demonstrates the feasibility of the scheme.

1 Introduction

Given a raw trace of web traffic collected from the outside of a private network, an adversary performing surveillance can be expected to take three steps:

1. Reconstruct TCP/IP connections from raw packets
2. Organize the connections into user sessions
   (mutually disjoint same-host sets)
3. Browse the web content of each user session to gather intelligence

Without the effects of gateway devices, the second step is trivial. The adversary logging packets from the outside can group them by the original host's IP address and produce user sessions. With (network address translation (NAT) and HTTP proxies however, the original IP address and other identifying information is absent, making it very difficult for to group traffic into user sessions.

Without any sophisticated techniques, an adversary performing surveillance on the outside of any of these devices would be able to reconstruct individual TCP/IP connections, but would be unable to group those connections into

user sessions. The adversary would be forced to sessionize them manually. This would involve evaluating the web content of every single connection and making a best guess at which ones belong together. The problem is akin to accurately assembling the pieces of many jigsaw puzzles jumbled together in one box.

The Link Chaining Attack (LCA) of this research aids the adversary by automatically organizing TCP connections into groups we call session fragments. Fragments are formed by following HTML hyperlinks across multiple TCP connections. These fragments are much larger than individual connections, and allow the adversary to assemble user sessions more quickly.

1.1 Related Work

The are three types of devices that pose increasing levels of difficulty to the problem of grouping traffic into user sessions (mutually disjoint same-host sets).

1. NAT
2. Plain HTTP Proxy
3. Anonymizing HTTP Proxy

Although none are designed specifically for surveillance, existing techniques [4, **?**] can be used to sessionize traffic collected from the outside of NATs and plain HTTP proxies, but not anonymizing HTTP proxies. The LCA was designed to operate under the strict conditions of anonymizing HTTP proxy. There is no known existing technique for doing this. The following three sections will explain why.

1.2 NAT

With NAT in place, a large number of private addresses are mapped to a small number of public addresses (often just one), so all traffic looks like it is coming from a single host. When all communication is with the same IP, there is no obvious way to differentiate the streams of traffic generated by individual hosts.

Existing attacks like Bellovin's IPid technique [4] can be re-purposed to group NATed web traffic into user sessions. These attacks exploit the fact that most NAT devices are configured to re-write only the IP address of packets. Other fields are left untouched, passing through NAT unchanged from their originating host. Bellovin traces the unchanged IPid field to reveal which packets come from the same host.

1.3 Plain HTTP Proxy

Web proxies are middlemen that fulfill transactions on the client's behalf. Without a web proxy, HTTP clients talk directly to HTTP servers. With a web proxy, two separate TCP connections are established: one between the client and HTTP

proxy, and one between the proxy and server. The use of this intermediary means that, unlike NAT, the TCP/IP packet headers contain no identifying features to differentiate streams emanating from different hosts. This renders attacks like Bellovin's IPid technique useless.

Original host information can still be found however, in the HTTP headers of outgoing requests. Plainly configured HTTP proxies pass these headers to the web server unchanged. If browsers in a network are not all configured identically, these headers can be used [5] to resolve at least some of the HTTP traffic to same-host sets. Of course, this assumes that the headers are present, and have not been scrubbed by an anonymizing proxy.

1.4 Anonymizing HTTP Proxy

The HTTP Anonymizing Proxy performs the same functions as a plain proxy, but scrubs all non-essential headers from outgoing requests. Without any headers to uniquely identify distinct hosts, keying on HTTP headers is not at all effective.

The Link Chaining Attack can be an effective technique under the conditions of an anonymizing web proxy because the technique uses only those HTTP artifacts which cannot be changed by the proxy. For example, HTTP headers can be changed by intermediate devices, but the web content itself cannot be changed in any meaningful way without affecting the browsing experience. The Link Chaining Attack takes advantage of this by reconstructing individual web pages from the traffic stream and following the links they contain forward in time to chain TCP connections into user session fragments.

1.5 The Effect of Web Caching

If some of the visited pages within a session are cached while others are not, the ones that are cached will not be requested. Their absence on the outside of the network means the chains of links will be broken, resulting in smaller fragments. From an intelligence-gathering perspective, the content loss is likely insignificant; caches store popular resources, and knowledge of a user's access to a popular resource reveals nothing unique about that user.

1.6 Research Goals

The aim of this work is to develop a technique that aids the analyst's manual sessionization by grouping TCP connections into fragments that are as large and accurate as possible. The technique follows the hyperlinks in HTTP messages to identify the TCP connections that belong together. The theory is described in section 2 and the experiment is outlined in section 3. Before presenting the results in section 5 we propose some metrics to evaluate the quality of fragments isolated by our technique. In the analysis of section 6 we validate the work by establishing a lower bound on the effective analyst speedup.

2 Theory

The Link Chaining technique coalesces independent TCP connections into same-host groups by following hyperlinks in web pages. By matching the URLs contained in the body of an HTTP response of one connection to the URLs in the HTTP requests of all other connections, and judiciously removing impossible or improbable links, it is possible to assemble fragments of user sessions.

Fig. 1. Chaining Two Independent TCP Connections

The TCP connection is the basic building block in this process. Figure 1 depicts the HTTP requests and responses of two independent TCP connections. The figure illustrates how the independent connections TCP 1 and TCP 2 can be chained by matching URLs. The hyperlink B in the first HTTP response of the first connection is matched with the URL B in the first HTTP request of the second connection.

The four phases of the Link Chaining technique are: Naive Chaining, Impossible Link Removal, Unlikely Link Removal, and Session Fragment Isolation. The first phase produces a tangled mass of edges and nodes representing all possible links between all connections. The two subsequent phases chip away at this mass, selectively removing impossible and unlikely links. By traversing the edges of the isolated graphs that remain, connection nodes are aggregated into groups. These groups of connections form session fragments. The process is summarized in Figure 2.

The raw inputs to the LCA are reconstructed TCP streams, HTTP messages, and the HTML hyperlinks they contain. Although these inputs are extracted from logged packets using known methods, the difficulty of this process should not be discounted. Before links can be extracted from web pages, the pages must be accurately reconstructed from individual packets. In many cases, the pages must also be decoded, uncompressed, parsed, and normalized. Relative links must then be resolved to their absolute form, stored with contextual meta data like timestamps and connection origin, and indexed appropriately for use in the LCA. For link extraction to be comprehensive and accurate, the software

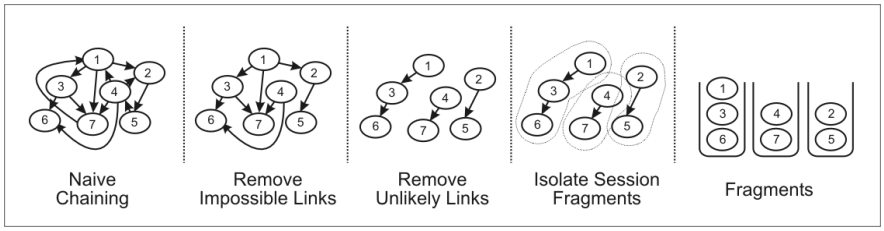

Fig. 2. Four Phases of the Link Chaining Attack

must also accomodate imperfect implementations of web protocols. These specifications essentially require the development of TCP/IP assembly and HTTP parsing facilities comparable to those of a full-fledged web browser.

2.1 Naive Chaining

The first step of the Link Chaining technique is to naively match all response URLs with all request URLs across all connections. A "URL match" is defined as a literal match between a URL in any response of one connection (e.g. in a web page) and a URL in the first line of any request in another connection (e.g. in a GET request). The complete set of URL matches can be represented as a list of adjacencies (ordered pairs) forming one or more directed graphs, where each node is a TCP connection.

Naive chaining identifies every single adjacency. This includes adjacencies representing link traversals that never actually occurred. By including all adjacencies, naive chaining produces a set of comprehensive starting graphs for the Link Chaining Attack. Many edges must be removed from these graphs before individual user session fragments can be isolated.

2.2 Removing Impossible Adjacencies

In the second phase of Link Chaining, the impossible edges in the graphs are removed. An edge is considered impossible if the link traversal it represents could never happen. The TCP and HTTP protocol mechanisms impose structural and temporal constraints on the traversal of links. Certain connections cannot be chained because it would imply an impossible link traversal. Two impossibilities are defined based on these constraints:

1. Connections Chained Backward in Time
2. URLs Chained Backward in Time

Each is discussed in turn.

Connections Chained Backward in Time When a page containing URL pointers to other resources is downloaded, it is followed by a flurry of requests. Some of these are due to the browser automatically requesting resources associated with the page, others are due to a user's clicking of a hyperlink. These are implicit and explicit requests respectively. In terms of HTTP protocol specification, no distinction is made between implicit and explicit requests.

If the requested resources are on the same server, and the web server and browser are so configured, HTTP requests may be issued on the same, already open TCP connection used to download the initial page. Otherwise, a new connection is opened to issue the request. HTTP requests can also be sent on older connections to the same server that are still open. This flexible connection reuse policy is made possible by HTTP/1.1 [1], and it affords us only one temporal constraint on the chaining of connections:

> **Constraint 1**: For any two TCP connections A and B, if B is closed before A is opened, A cannot be chained to B.

URLs Chained Backward in Time The second important temporal constraint is due to the fact that a resource request cannot be made if the URL pointer to that resource has not yet appeared in a response. For implicit requests, this simply means the browser cannot request a URL that has not yet been downloaded. For explicit requests, it means that users cannot click on URL hyperlinks that have not yet appeared on screen. This constraint is summed up as follows:

> **Constraint 2**: A link traversal is impossible if the request URL appears before the response URL. A URL Match representing such a traversal is invalid. Two connections cannot be chained if every URL Match between them is invalid.

Since the packets of connections are interleaved on the wire, the content of connections is interleaved in time. To determine the validity of a URL Match, the timestamp of the request URL must be compared with the timestamp of the response URL. In a timing diagram, HTTP events in a connection might look like Figure 3.

Assuming a URL in the response of connection 1 matches the URL in the request of connection 2 in the figure, we must decide whether it is temporally possible that the request in connection 2 was initiated from 1. If not, the connections cannot be chained. To do this, URLs must be tagged with the time their containing packet appeared in the traffic stream.

2.3 Marking Likely Adjacencies

The preceding step identifies adjacencies that are definitively impossible, and can therefore be removed from the connection graphs. The remaining adjacencies cannot be removed this easily. Since they do not violate any of the constraints, every remaining adjacency is a potential candidate for inclusion.

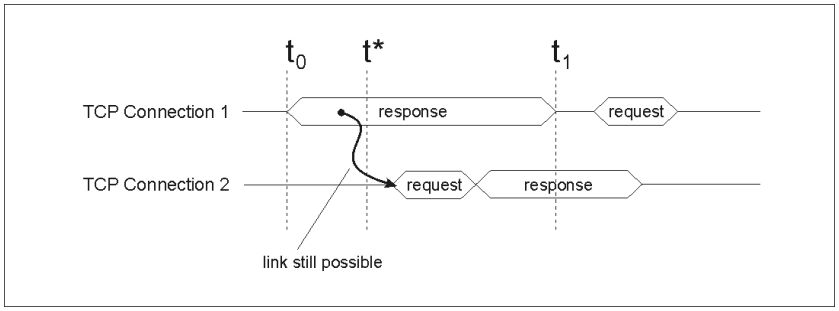

Fig. 3. Chaining Two Independent TCP Connections

To accurately isolate user session fragments, the most likely of the remaining adjacencies must be identified. A time oriented heuristic was developed to do this. The heuristic is based on the time between the appearance of a URL, and the request for the resource it points to. This time is called think time, and it is defined differently for browsers and users.

User Think Time (Δ_e): Length of time between a page download and a hyperlink click (explicit request). User think time includes the browser's parsing and rendering time.

Browser Think Time (Δ_i): Length of time between a page download and an ancillary, automatic request (implicit request). Browser think time includes browser parsing time.

Think time corresponds exactly to the length of time between matching URLs in distinct connections. It can be represented by a label on each edge in a TCP connection graph. An example of this is shown in Figure 4.

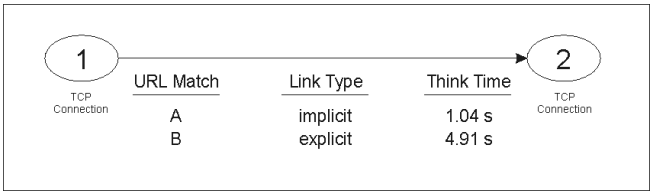

Fig. 4. Think Times for Two Links between Two Connections

The figure shows two potential URL matches linking connections 1 and 2. The first URL match implies an implicit request (an ancillary request made automatically by a browser fetching embedded content), while the second implies an explicit request (a request resulting from a human user click). Think times are calculated for every URL match, including matches implying link-traversals

that never occurred. The marking of likely adjacencies is based on the length of these think times.

The time oriented heuristic is a simple set of think time limits outside which link traversals are deemed unlikely and removed. Link traversals (represented by URL matches) are removed according to the following rules:

Implicit URL Match (Browser Request): if think time $t_t > \Delta_i$, remove.
Explicit URL Match (User Request): if think time $t_t > \Delta_e$, remove.

Borrowing from the traditional sessionization techniques of web analytics [5], the values of Δ_i and Δ_e are 20 seconds and four minutes respectively.

The heuristic is only applied to those connection nodes having an indegree greater than one. That is, nodes with multiple incoming edges that imply the node was linked-to from more than one other connection. An example is shown in Figure 5.

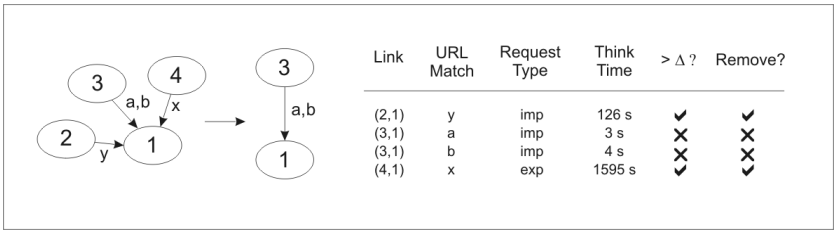

Fig. 5. Removing Unlikely Adjacencies from a Multi-Indegree Node using the Time Oriented Heuristic

Multi-indegree nodes (MINs) are an ideal target for edge removal because they are over-represented in the adjacency graphs. Although naive chaining produces lots of them, MINs only happen for real when requests initiated from multiple connections are being issued on a single, already open, connection. This is a connection reuse scenario that web browsers do not experience often. MONs (multi-outdegree nodes), on the other hand, happen all the time. They represent the situation where multiple connections are being initiated from the same connection, like when a flurry of implicit requests are made for objects embedded in a page.

Because it focuses only on MINs, the time oriented heuristic is consistently optimistic. It leaves most out-links intact. The only out-links it removes are those associated with MINs.

2.4 Fragment Isolation

The Link Chaining process begins as a tangled graph of naively chained connections. This graph is then processed to remove the impossible and unlikely adjacencies. The remaining graphs of connected nodes form the fragments that

the analyst will use to assemble user sessions. The fragments are isolated by simply tracing the edges of each graph and aggregating the connection nodes.

3 Experimental Setup

Network traffic was collected passively from the inside of a live campus network with a high volume (2 GB/hour) of web traffic and later written to a database. The logging point was situated at the gateway before any NAT or proxy so that individual host IP addresses were visible. A real attack would tap external to this gateway, but IP address visibility was necessary here to validate the results. All traffic features that would not normally appear in the presence of NAT or proxy were selectively ignored for each experiment. The tap and network under test are illustrated in Figure 6.

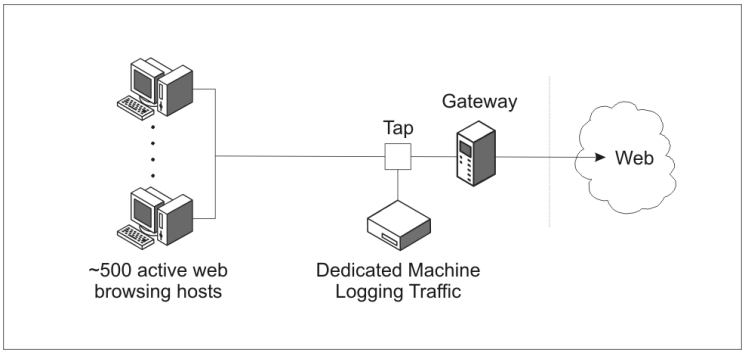

Fig. 6. Network Under Test

Traffic collection was performed using Snort 2.0. Snort is an open source network intrusion detection system, capable of performing real-time packet sniffing, analysis, and logging on IP networks [6]. In this experiment, it was used exclusively for its packet sniffing and packet logging capabilities. The tool was configured to break out packets into their constituent fields and write them to a MySQL [8] database.

Figure 7 shows the three tools used to prepare the data. The first tool labels all packets by TCP connection and removes broken or empty connections. The second reconstructs the contents of every TCP connection while preserving the relationship of those contents with their underlying packet features. The final tool parses all relevant HTTP features and statistics from each TCP stream. The results from each of these steps are written back to the database.

These tools process the raw packets to produce multiple views of the data across all relevant protocols. They provide a convenient, granular, and relational breakdown of every traffic feature of interest. All tools were written in C++ and made extensive use of MySQL++ [8], an object oriented API used to

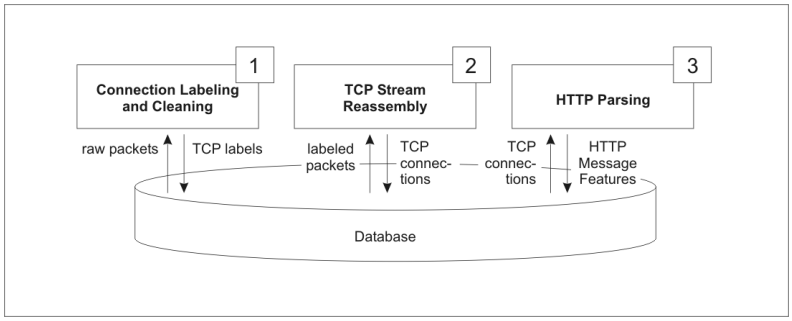

Fig. 7. Three Data Preparation Steps

access the database. The API allows queries and query results to be handled as STL Containers. Shell scripts were used to drive the compiled tools. Perl was employed for some ancillary tools.

The TCP reassembler reconstructs TCP streams accurately despite packet retransmissions or out-of-order delivery. The reassembler operates on a database of packets (as opposed to a raw log) and preserves the mapping between a stream's content and its constituent packets.

The HTTP Parser extracts information from the HTTP transactions in reassembled TCP stream files. It parses individual HTTP headers as well as the web resources contained in the bodies of HTTP responses. For example, the parser can rebuild sounds, images, and documents from the HTTP stream. It can also inflate or unzip HTML web pages that have been compressed by web servers. This is necessary for extracting the valuable hyperlinks that allow the Link Chaining Attack to chain TCP connections together into user sessions. The parser very much emulates the parsing functionality of a web browser.

Data preparation constituted a significant effort before the Link Chaining Attack could be applied.

3.1 Experimental Inputs and Procedure

The experiment was performed for five sets of Port 80 traffic data. Each set was collected in the same hour on different week days. In raw TCPdump [9] format, the data sets were roughly 550Mb each. They each contained about 30 minutes of traffic generated by approximately 500 active hosts. Each set contained about 750,000 packets, 25,000 TCP connections, and 100,000 HTTP messages.

Data Set	Active Hosts	Log Size (Mb)	Time (min)	Packets	Connections	HTTP Msgs
0	509	550	27.35	737,861	23,368	103,261
1	460	561	31.02	902,147	21,877	170,021
2	411	534	27.14	690,144	14,581	132,350
3	517	591	26.98	754,369	19,378	147,200
4	442	612	33.33	887,213	26,601	178,446

3.2 Two Versions of Fragment Isolation

Fragment isolation was performed in two ways for each data set. In the first, fragments were isolated from all possible adjacencies. In the second, fragments were isolated only from those adjacencies marked as likely by the heuristic. The two tests were labelled A and B respectively.

Fragment Isolation Tests
A - All possible adjacencies
B - Adjacencies marked as likely by the heuristic

Both tests are versions of the Link Chaining Attack. Test A should be considered a naive implementation. It was conducted to establish a baseline for the performance of the heuristic in test B.

4 Link Chaining Evaluation Metrics

For session fragments to be useful to a human analyst, they must be as large and accurate as possible. The evaluation of the Link Chaining Attack is based on a series of metrics that measure how the test fragments compare to actual whole user sessions. Actual user sessions are complete sets of same-host connections, organized by IP address. The IP address of every TCP connection is recorded in the experiment so that actual user sessions can be isolated and easily compared with fragments.

The measures for fragment quality are based on the degree to which actual user sessions are reconstructed by fragments. These measures consider the number of TCP connection elements in the intersection of a fragment and an actual user session. They are described in the following sections.

4.1 Coverage

Coverage is the degree of overlap between the connection elements in recovered fragments and actual user sessions. Coverage measures the size of the fragment in relation to the size of the actual session. For a given fragment f and actual session s, coverage C is given by:

$$\text{Coverage} \quad C = \frac{|f \cap s|}{|s|} \tag{1}$$

4.2 Accuracy

The fraction of fragment elements that have been correctly assigned. It is calculated as follows:

$$\text{Accuracy} \quad A = \frac{|f \cap s|}{|f|} \tag{2}$$

Ideally, the Link Chaining attack would reproduce entire user sessions. That is, it would produce fragments of unit coverage and accuracy. This is highly unlikely. Instead, the goal is to consistently isolate non-trivial session fragments of high accuracy. Regardless of their size, non-trivial fragments decrease the user session assembly time for an analyst as long as they are accurate.

4.3 Matching Fragments to Actual Sessions

There are always more session fragments than actual user sessions. Before applying any metrics, each fragment must be matched to the user session of which it is a part. The best matching user session is the one that shares the largest number of connection elements with the fragment. For a given fragment f, and the set of all user sessions S, the matching session m, is given by:

$$\text{Matching Session} \quad m = \Big\{ m \epsilon S \Big| |f \cap m| = max\big\{ |s \cap f| \Big| s \epsilon S \big\} \Big\} \tag{3}$$

4.4 Ambiguous Fragments

Some fragments will match multiple user sessions. Such fragments are inaccurately chained and contain equal numbers of connections from two or more sessions. For example, the following fragment f matches sessions s_1 and s_2 equally:

$$f = \big\{1, 2, 3, 4\big\} \quad s_1 = \big\{1, 2, 5, 9, 13\big\} \quad s_2 = \big\{0, 3, 4, 12, 26, 52\big\}$$

To evaluate these fragments effectively, they must be assigned to, and compared with, a single whole user session. There is no way to do this meaningfully. Such an assignment would be essentially arbitrary. Fragments that are too ambiguous to evaluate in the context of this experiment would be similarly confusing to the analyst in practice. Measuring the quality of such fragments is pointless; they are all bad. For this reason, the metrics are not applied to ambiguous fragments. Instead, the fragments are counted separately, and presented as an index of ambiguity, indicating one aspect of the performance of the LCA overall.

$$Ambiguity = \frac{AmbiguousFragments}{AllFragments} \tag{4}$$

4.5 Trivial Fragments

By definition, fragments made up of one connection element always match one session and have unit accuracy. Their effect is to increase the aggregate accuracy in a meaningless way. For example, if half of all fragments are trivial, the aggregate accuracy is guaranteed to be at least 0.5. This is an unnaturally inflated score that does not represent the accuracy of non-trivial fragments. To correct this, accuracy is not measured for trivial fragments, and aggregate results are presented with a triviality score.

$$Triviality = \frac{TrivialFragments}{AllFragments} \tag{5}$$

5 Results

5.1 Trivial and Ambiguous Fragments

Trivial fragments accounted for 5.25% to 9.33% of all fragments in Test A and 12.81% to 16.81% in Test B. The larger number of trivial fragments in Test B is to be expected, as the naive method of Test A chains connections into fragments much more readily than the discerning heuristic of Test B. It is important to mention that some fragments were small because the user sessions themselves were small. Specifically, 3.48% to 7.21% of actual user sessions were trivial.

Ambiguous fragments accounted for 2.25% to 4.41% of all fragments in Test A and 1.14% to 4.02% in Test B. There was no statistically significant difference in ambiguity between the two methods.

5.2 Coverage

The distributions of coverage scores for Tests A and B are shown in Figure 8 and 9. The coverage of the fragments isolated by the heuristic appear to be exponentially distributed, with about 75% of them having session coverage less than 25%. The naively isolated fragments are distributed much differently, with generalized peaks at coverages less than and greater than 50%.

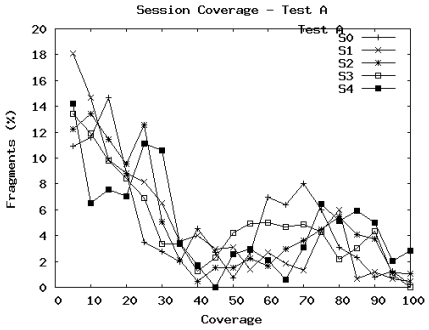

Fig. 8. Distribution of Session Coverage of Fragments (Test A)

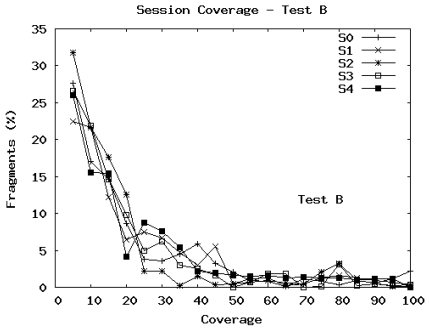

Fig. 9. Distribution of Session Coverage of Fragments (Test B)

5.3 Accuracy

The distribution of fragment accuracy for Tests A and B is shown in Figures 10 and 11. The figures show clearly that the heuristic isolates fragments that are much more accurate than those of the naive method.

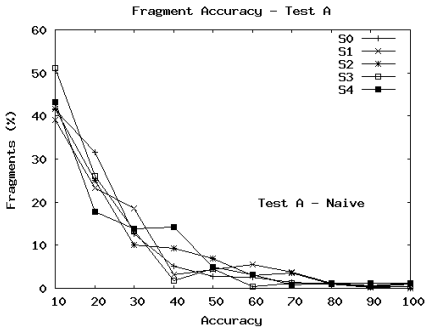

Fig. 10. Distribution of Accuracy Scores, Naive Chaining (Test A)

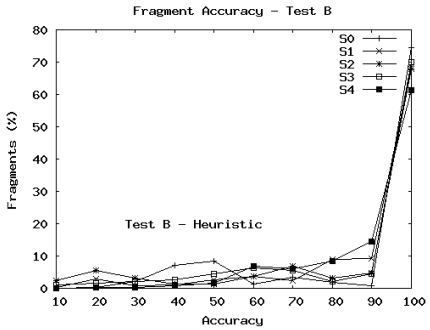

Fig. 11. Distribution of Accuracy Scores, Heuristic (Test B)

6 Analysis

The previous section showed that the Link Chaining Attack was able to group TCP connections into non-trivial fragments with moderate success. The indegree heuristic proved to be far more accurate than naive chaining, although the fragment sizes it produced were much smaller. The averages for each metric are summarized in Table 1 below.

Test	Fragment Size	Coverage	Accuracy	Triviality	Ambiguity
A. Naive	58.67	31.48	24.15	6.96	3.28
B. Heuristic	10.62	12.63	88.41	14.3	3.32

Table 1. Summary of Link Chaining Performance Averages

This research has been predicated on the notion that it is desirable for human analysts to group the contents of passively logged TCP connections into user sessions for the purpose of surveillance. The above results are now used to show how Link Chaining aids this process.

6.1 Modeling Sessionization Time

Without Link Chaining, or a similar technique, the largest unit of network traffic that can be rebuilt from the stream automatically and reliably is the TCP connection. After TCP connections are rebuilt, it is assumed the analyst would sessionize them by analysing hyperlinks, content, semantics etc. Since no real data on human sessionization time is available, the time t_s, to sessionize n con-

nections is modeled as follows:

$$\text{Sessionization Time Model 1} \quad t_s = t_c \frac{n(n-1)}{2} \tag{6}$$

Where the time to compare one connection or fragment to another, t_c, is constant, and is multiplied by the maximum number of comparisons required (i.e. the comparison of all possible connection pairs or $\binom{n}{2}$). This is a conservative model. A sample sessionization of four connections is shown in Figure 12.

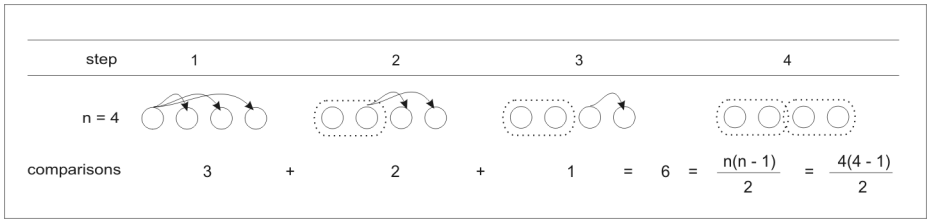

Fig. 12. Sample Sessionization of Four Connections

Modeling sessionization time without empirical data is admittedly clumsy. The following relationship is used to model the best case sessionization time achievable by an analyst, $t_s^\star$, which is linear with respect to the number of connections. It is impossible to argue that a human (or even a computer) can do better than compare all connections in one pass simultaneously, so the model is used as an ultra-optimistic benchmark.

$$\text{Sessionization Time Model 2} \quad t_s^\star = n \cdot t_c \tag{7}$$

6.2 Time Savings

The average size of fragments isolated by the heuristic in the Link Chaining Attack was 10.62 connections. Based on this average, the number of pieces, n, that an analyst would have to sessionize is reduced to $\frac{n}{10.62}$. Figures 13 and 14 illustrate the effect of such a reduction on sessionization time using both models M1 and M2. Figures 15 and 16 are plots of the original sessionization time over the reduced time, again with both models.

The first model shows that based on the average fragment size of the experiments, a human analyst working with fragments (as opposed to individual TCP connections) would experience a speedup of greater than 100 when based on a conservative model of analyst efficiency. When based on an optimistic model for analyst efficiency, the LCA represents a ten-fold speedup. Since the optimistic model represents the best possible case for a human analyst's unaided performance, it is expected that the actual speedup would be significantly better than the indicated ten-fold speedup.

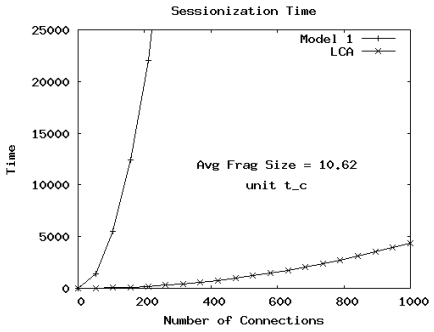

Fig. 13. Sessionization Time Functions, Original and With LCA, Model 1

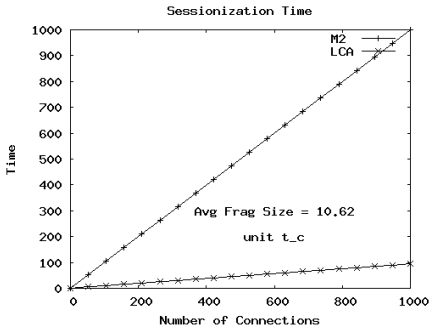

Fig. 14. Sessionization Time Functions, Original and With LCA, Model 2

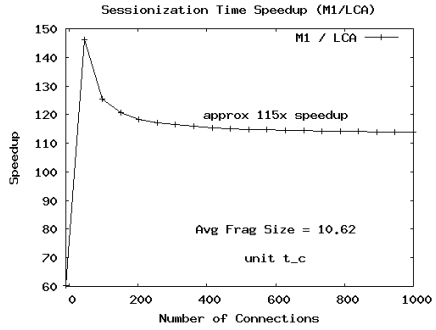

Fig. 15. Sessionization Time Speedup ($\frac{Original}{LCA}$), Model 1

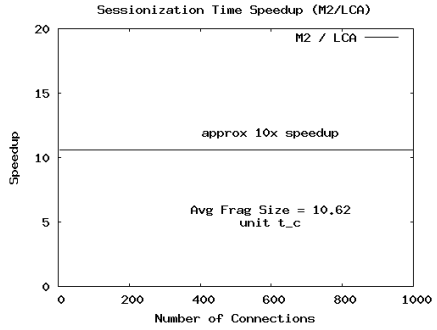

Fig. 16. Sessionization Time Speedup ($\frac{Original}{LCA}$), Model 2

The amount of content visible in each fragment has a definite impact on sessionization speed. Individual TCP connections offer only a small window onto a user's browsing sessions, while fragments made up of multiple connections offer a much larger window. This larger window provides the analyst with much more semantic context, allowing him to infer user sessions more easily than he could with individual TCP connections.

For example, some of the fragments in this experiment were rendered in a web browser. These fragments revealed stock research pages, online education seminars, and shopping pages. In a few cases, whole webmail sessions were contained in one fragment and could be rendered in their entirety, including email attachments.

7 Conclusion

By reducing the high cost of sessionizing connections manually, the Link Chaining Attack makes passive external surveillance of private networks a real possibility. The results suggest a minimum ten-fold speed improvement for a human analyst with acceptable accuracy. This number may be closer to 100 when using a reasonable model of human sessionization speed.

The fact that the indegree heuristic performed more accurately than the naive method of fragment isolation demonstrates that web traffic contains an exploitable relationship that is more descriptive than that marked by hyperlinks alone. Web browsing is governed by a discernible pattern of user and browser think times that can be used — together with tracing hyperlinks — to group connections.

The Link Chaining Attack capitalizes on navigation and time oriented heuristics to sessionize fragments of user sessions. Proposed improvements include the tuning of user and browser think time thresholds, the identification of new impossibilities for link removal, and the discovery of impossible event sequences spanning multiple connections. A method for assessing the likelihood of a link based on a recursive calculation of the likelihood of its adjacent links is also being considered.

It is believed that evolved versions of the technique will take advantage of these small improvements to enable the uncomplicated passive external surveillance of private networks — despite the anonymizing effects of NATs and HTTP proxies.

References

1. R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. Leach, and T. Berners Lee. Hypertext Transfer Protocol: HTTP/1.1. Internet Draft Standard RFC 2616, June 1999.
2. Clinton Wong. HTTP Pocket Reference. O'Reilly, July 30, 2000.
3. Gourley, David et al. HTTP: The Definitive Guide. O'Reilly, Cambridge, September 2002.
4. Steven M. Bellovin. "A Technique for Counting NATed Hosts". www.research.att.com/ smb/papers/fnat.pdf, AT&T Labs Reseach, 2003.
5. Bettina Berendt, Bamshad Mobasher, Myra Spiliopoulou. "Web Usage Mining for E-Business Applications" ECML/PKDD-2002, 19 August 2002.
6. Snort IDS. http://www.snort.org/about.html.
7. MySQL. http://www.mysql.com/.
8. MySQL++. http://tangentsoft.net/mysql++/.
9. TCPdump. http://tcpdump.org/.