RequIn, a tool for fast web traffic inference

Olivier Paul, Jean Etienne Kiba
GET/INT, LOR Department
9 rue Charles Fourier
91011 Evry, France
Olivier.Paul@int-evry.fr, Jean-Etienne.Kiba@int-evry.fr

Abstract — **As networked attacks grow in complexity and more and more Internet users get broadband Internet access, application level traffic analysis in operator networks becomes more difficult. In this paper, we describe a tool allowing web communications to be analyzed in such environment. Instead of relying on the extraction of application level parameters and pattern matching algorithms that are usually considered bottlenecks for such activity, we look at simple network and transport level parameters to infer what happens at the application level. Our approach provides the ability to perform a trade-off between analysis speed and precision that in our opinion could be useful for some traffic analysis applications like denial of service attacks detection.**

Keywords-component; Monitoring, HTTP, performance, DDoS.

I. INTRODUCTION

Over the last ten years, a part of the security functions that were previously implemented within companies has been delegated or outsourced to external organizations. The appearance of new threats (e.g. worms, DDoS attacks) has led network operators to provide intrusion detection services to their customers. In this paper we consider one of the challenges implied by this new activity; the ability to monitor user communications within operator networks. This task can be considered as challenging for several reasons:

- Operators' network internal devices usually have basic traffic analysis abilities. Most devices are currently limited to operations applied to packet headers through capture, aggregation, filtering, sampling and counting operations.
- Operators' network internal links usually carry large amounts of traffic. As a result the time that a monitor can devote to each user request is usually very short (a few hundreds of nanoseconds). As a result complex operations such as the algorithms employed in end-hosts monitoring systems are usually unusable. For example the *snort* intrusion detection tool uses pattern matching algorithms for which the best known solution in term of temporal complexity [2] is in O(n+m) where n is the size of the string to be searched in and m the size of the pattern. Such algorithms would clearly be unable to handle strings longer than a few words in very high speed environments.
- Operator networks are usually constrained in term of introducing new mechanisms or tools by two parameters one being the reliability of their network, the other one being management cost. As a result new techniques should as far as possible take advantage of existing monitoring mechanisms in order to limit the modification to existing elements.

In this paper we focus on HTTP communications. According to ISPs [3], HTTP traffic constitutes between 35 and 50% of the Internet traffic.

The goal of this paper is to present a tool that allows such communications to be analyzed in the middle of a network while complying with the aforementioned limitations. We first introduce the measurement information our analysis is based on. Section IV shows how such information can later be used to deduce users requests. Section V presents *RequIn*, an implementation of our techniques as an extension to *IPfilter* on *FreeBSD*. We then test our analysis technique and implementation by considering models and traffics originating from our web site.

II. MEASUREMENT INFORMATION

Our goal is to permit the monitoring of web communications when application level information cannot be used. In order to do so our plan is to use network and transport level information in order to infer application level behaviors. In this section we first introduce network and transport level measurement capabilities.

A. The HTTP Protocol

HTTP exchanges can be viewed at several levels. At the lowest level, the HTTP protocol is based on a request-response protocol where each request attempts to perform an HTTP operation on an object at the server. We later call this level micro-session level. Information in HTTP 1.1 messages [4] is organized into information elements called headers. Although HTTP 1.1 defines more than 40 different headers, requests and responses usually only use a few them. Requests usually include some of the following headers: The version of the protocol, a method indicating the action to be performed, a URI indicating the object the action is to be performed on, a destination identifying the targeted web server, the date at which the request was performed...

This work is funded through European Commission IST FP6 DIADEM FIREWALL and GET DDOS projects.

Similarly, a response usually includes similar headers (version, encoding, date, server) and some specific headers like a status indicating the result of the request, the content length or information targeting caches (expiration date, cache directives).

B. *Measurement Information Selection*

As mentioned earlier, measurement operations aim at understanding application level operations through the analysis of network and transport level information. Although the application level protocol has an impact on this information, this impact also depends on intermediate protocols. Additionally some application level parameters might be more difficult to infer than others. As a result a first step is to try and map application level parameters to transport and network level parameters. The strength of this mapping is later examined in the following sections.

TABLE I. PARAMETERS MAPPING.

Parameter	Network/Transport level parameter
Method	Request data size, Response data size
URI	Response data size
Source	Source IP address, Port
Destination	Destination IP address, Port
Time/Date	External: Time/Date
Compress.	External: Server configuration
Caching	Response data size.
Status	Response data size.

As indicated in table I, sizes are expected to be a significant source of information in order to infer several application level parameters. More specifically our assumption is that object sizes and object identifiers are closely connected and that object sizes and transport/network level measured sizes are also connected. While this last relation is obviously true with HTTP 1.0 where a connection is used for each object, HTTP 1.1 uses several improvements that can render this relation weaker.

C. *HTTP/TCP Relationship*

HTTP 1.1 [4] provides the ability for web clients and servers to multiplex several HTTP request-responses exchanges over a single TCP connection. Among persistent connections we can also distinguish connections using pipelined requests from regular connections. Pipelined connections are used by the client to perform several requests without waiting for an answer from the server. This ability is however usually limited by the structure of html objects where imported objects can only be requested after the html document linking to them is received by the client.

Therefore in connections, request-response sessions can be distinguished at the network level by either looking at:

- Connections set-up and ending in the case of non persistent connections (whether pipelined or not).
- Request-Response session patterns [6] in the case of non-pipelined persistent connections. These patterns can be found at the network level by considering TCP sequence numbers evolutions. As the sequence number from the client only increases when a new requests is sent to the server, we can set the beginning of each new session when a client sequence number increase occurs. Since several requests cannot be served simultaneously over the same connection this also represents the end of the previous session.
- Request-Response session patterns in the case of persistent pipelined connections. In the case of pipelined requests, only the first request-response session can be distinguished from other exchanges. The next session may include one or several request-response sessions.

As a result pipelined, persistent connections can make the relation between Network/Transport level information and application level information so weak that it can hardly be used. As a result an interesting question is whether pipelined connections are supported in the real life. Reference [5] shows that most browsers (MS *IE*) are not able to use pipelined connections. Moreover browsers (*Firefox*, *Netscape*) that do support pipelining are usually configured to avoid using it.

Being able to distinguish micro-sessions allows us to measure the amount of data transported by TCP for a request or a response by looking at TCP sequence number evolution during a micro-session. Note that this also renders our measurement scheme independent from IP packet losses.

III. METHOD AND OBJECTS SIZE INFERENCE

A mentioned earlier, our assumption is that objects sizes can be inferred from network or transport level measurements. As a result being able to perform that operation as correctly as possible is critical to our scheme. Several factors like HTTP headers can play a role in making this process more difficult. Our assumption is that the size of HTTP headers can take a limited number of values. For a given server these values depend on the server configuration.

A. *Response type and method inference*

Fig. 1 provides the relation between header sizes, types of response and total sizes in the case of our web server. These values where obtained by capturing responses packets from the server over 24 hours. Six types of responses (identified by code numbers) were captured.

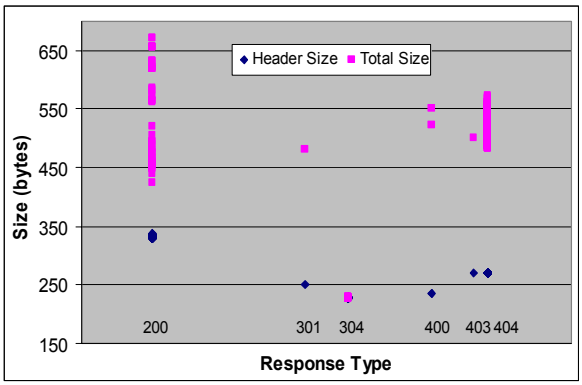

Figure 1. Response Type/Size Relationship.

Responses carrying 200 (“Ok”) result codes can be distinguished from other responses by looking at the total size (total size > 570 bytes). As show in fig. 1, some 200 responses have a size that collides with other types of responses. However objects carried by these requests constitute less than 1% of existing objects. 304 ("Not modified") responses can also be distinguished from other responses by looking at the total size (total size <250). Other responses cannot be distinguished as they carry objects whose size can vary widely.

Additionally, our tests showed headers for non persistent connections as well as headers for persistent connections which included additional headers with a computable size.

As a result knowing whether a connection is persistent is sufficient to deduce the influence of the persistence on the HTTP header size. This knowledge can be obtained using the session delimitation scheme described in section II. Non persistent connections are distinguished by looking for multiple connections establishment-teardown over short periods of time. table II provides the relation between response size and response codes for persistent connections.

TABLE II. RESPONSE CLASSIFICATION USING RESPONSE SIZE (RS).

Result	Response Size
200	RS >570 250>RS>460
304	240<RS <250
301, 400, 403, 404	460<RS<570

Using a similar methodology, we define a set of classification criterion in order to infer the method used in HTTP requests. However, we found that determining the methods type using solely the response size could not be performed efficiently. In order to do so, we use the combination of request and response sizes.

B. Object Size Inference

Fig. 1 shows that 200 responses can carry HTTP headers whose sizes are not fixed. As a result using an average HTTP header size value to estimate objects size in the case of GET requests can lead us to some errors. By looking more closely at headers fields we can classify them according to their behavior:

- Some headers never change (e.g. response code, server identifier; accept range,…).
- Some header values change but have a fixed size (e.g. last modified, date and Etag).
- Some header values change depending on the associated object (e.g. content type and length).

As a result for a given object, the response size should remain constant. This means that by keeping the relation between response sizes and object sizes, we can get an exact estimate of objects sizes.

Popular HTTP servers support objects compression prior to sending them to the client. This can cause a difference between the number of bytes measured in the network and the size of the object. The compression option is used when an appropriate configuration is performed on the server side and when the client supports compression. However as most clients support compression, knowing if compression is used is only a matter of knowing if the server is configured to use it. In this case HTTP servers provide the ability to log both compressed and original sizes for each requested object. The inference process in the case of compressed object therefore remains the same.

IV. URI INFERENCE

Our assumption for URI inference is that network and transport level measurement parameters can be used to infer objects identifiers for GET requests:

- Each given object has a single size. As a result knowing an URI can help us explaining objects sizes and reciprocally.
- Users originating from different locations have different interests. For example local students are usually more interested in schedules and courses related information while users connecting from remote research institutions are more interested in research related objects.
- For the same reasons people residing in different time-zones use different parts of the server.

In order to understand the relations between measurement parameters, we use access logs available on web servers. These logs are usually made of a set of entries, each of them describing an action performed on the server. Because each entry links IP addresses, time and date information, object sizes and object identifiers, we can use log entries in order to build a model that will later be used to infer objects identifiers when provided other parameter values.

A. Inference Model

The model we selected to perform inference operations is a Bayesian network. Bayesian networks are graphical models that can be used to represent causal relationships between variables. A Bayesian network is usually defined as:

- An acyclic directed graph $G = (V,E)$, where V is a set of nodes and E a set of vertexes.
- A finite probability set (Ω,Z,P).
- A set of variables defined on (Ω,Z,P) such as:

$$P(V_1, V_2, \quad , V_n) = \prod_{i=1}^{n} P(V_i | C(V_i))$$

Where $C(V_i)$, is the set of causes for V_i in the graph.

The inference in a causal network consists in propagating one or more unquestionable information within the network, in order to deduce how beliefs concerning the other nodes are modified. If node X_i is located downstream from node X_j, we can write:

$$P\langle X_i | X_j \rangle = \sum_{X_{i-1}} P\langle X_i | X_{i-1} \rangle \cdot P\langle X_{i-1} | X_j \rangle$$

If node X_i is a direct descendant of X_j, the computation is over. In the other case we can break up $P\langle X_{i-1}, X_j \rangle$ until we reach a direct descendant of X_j.

If node X_i is located upstream from node X_j it is necessary to propagate the information starting from the beginning of the chain, to know the unconditional probability for each node $P(X_k)$, $(1 \le k \le j)$. In order to do so, we can use the property of inversion of the conditional probability:

$$P\langle X_i \mid X_{i+1} \rangle = \frac{P\langle X_{i+1} \mid X_i \rangle \cdot P(X_i)}{P(X_{i+1})}$$

As with the downward propagation, if X_i is a direct ascendant for X_j the computation stops here. In the other case we can perform the same operation on X_{i+1} ascendants.

B. *Variables Selection*

In order to obtain an efficient model, we first performed some aggregation on variables.

- IP addresses were aggregated into country codes.
- Seconds, minutes and hours information was aggregated into a single hour variable.
- Day of the week, month, year information was aggregated into a single day of the week variable.

As the cost of inference in a Bayesian network increases exponentially with the number of variables in the network, it is essential to limit that number. In order to do so, we evaluate the ability for each parameter (size, country, time and date) to explain object identifiers. For each couple of variables (URI,X), we do so by computing $P(URI \cap X)$ and comparing it with $P(URI).P(X)$ by computing:

$$I(URI, X) = \frac{\sum_{i=1}^{N} \left([P(URI \cap X_i) - P(URI)P(X_i)] \right)}{N}$$

The ranking between I(URI,X) values lead us to the simple Bayesian network presented in fig. 2.

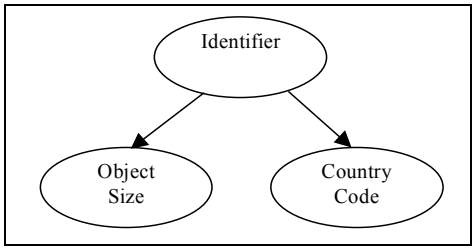

Figure 2. Resulting Bayesian Network.

V. Implementation and Tests

A traffic analyzer was implemented as an extension to *IPFilter* [7]. The HTTP session handling function is implemented as a part of the TCP state maintenance function. This function extends the TCP connections data structures by allowing multiple HTTP micro-sessions to coexist within a TCP connection. Micro-sessions are delimited as specified in section II and specified using the *IPFilter* filtering policy.

When a session ends, the corresponding information (Source IP address and port, Destination IP address and port, timestamps, number of TCP bytes transported in both directions, Number of packets and bytes transported, type of connection) is handed to the kernel syslog part. This information is later exported to the user space and retrieved by *RequIn*. *RequIn* is first used to transform timestamps into time and date values as well as IP addresses into country codes. To do so we use a static IP address database for performance reasons. When started, *RequIn* first uses logs from the server to monitor in order to build the corresponding Bayesian network and method-response codes classes. When such models are built, classification and inference models can be used to infer users' actions.

A. *Validation Tests*

Our validation tests were performed using our departmental web server. This server runs with *Apache 1.3* and includes roughly 15k objects, most of them being static pages and receives 7k requests a day. In order to perform consistent tests over a long period of time, a copy of this server was made on a similar computer. This copy was later used for the tests.

In order to check that our server did not have a structure that would have tainted our tests, we performed a comparison between requests sizes to our server and the ones usually found on the internet [6]. Fig. 3 shows both cumulative distribution functions. Sizes smaller than 500 bytes have been ignored since header sizes distribution is unknown in [6]. Overall there is little difference between the two distributions except in the $[10^5;10^6]$ range where the difference should not have a large impact on our scheme.

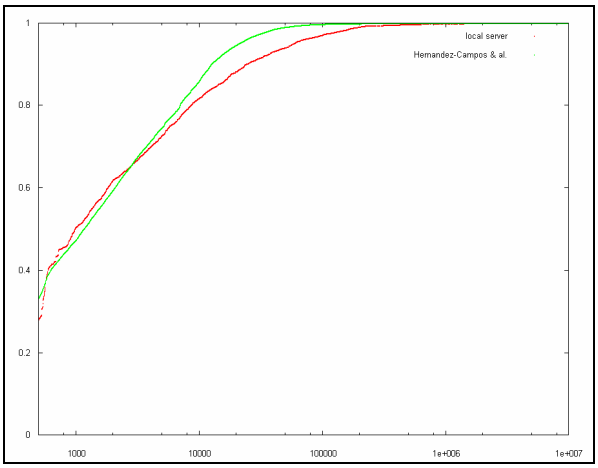

Figure 3. Responses sizes cumulative distribution.

Using the model defined in section IV, we built a Bayesian network for this web server using a 309k entries log file gathered over 43 days from the original server. In order to test the ability of the model to predict future requests, we first investigated the influence of time on the estimation accuracy.

Fig. 4 provides the evolution of the correct estimate rate over three weeks when using a three weeks log to build the model. As shown in fig. 4, the percentage of correct estimates remains around 75% during roughly 10 days (records 1 to 70k).

It then slowly falls to 71% over the next 12 days as new objects are stored in the server.

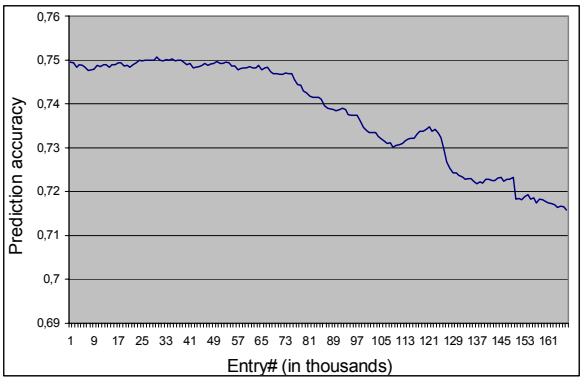

Figure 4. % of correct estimates over time.

The validation of the method and response code inference methods were performed using a similar process. Estimation results are provided in table III.

TABLE III. RESULT AND METHOD INFERENCE.

Estimated parameter	% correct estimation
Method	95
Operation result	96

This first estimation does not take into account the perturbation that might be introduced by the measurement part of *RequIn*. In order to validate the whole software we generated sequential requests for each object identifier found in the full log file. Requests were analyzed by *RequIn* which produced the inferred user actions. These actions were later compared to original requests. Results are provided in table IV. This test is however biased by two parameters:

- The inference part is unable to take advantage of the country code information. This should decrease the accuracy of the inference.
- The inference process is not affected by aging as the server configuration is static. This should increase the accuracy of the inference.

Given the variation of accuracy over time (fig. 4) we however believe that this last parameter should have a small effect over the first ten days. Consequently we expect to get slightly better results with real life traffic.

TABLE IV. VALIDATION OF WHOLE SOFTWARE.

Scenario	% correct estimation
URI	74
Method	90
Operation result	94

B. *Performance Tests*

RequIn was tested on *FreeBSD* 5.2 on a 2.4Ghz *Pentium Xeon* processor with a 512KBytes cache. During our tests we benchmarked several aspects of the inference process including the time required to build the models, the size of the models and the time required to infer a request once models are built.

For the test we used an access log file including 77k entries to build the inference model. We then used (IP address, object size) couples from a 232k entries log file to perform the performance test. We performed 50 series of tests and averaged the results.

TABLE V. PERFORMANCE RESULTS.

Parameter	Value
Time to build the models	4s
Size of the model	1.5 Mbytes
Time per request	0.9us

These results (table V) show that our inference process, when used independently from the request-response measurement mechanism should be able to analyze roughly 1.1M requests per second. Assuming an average Internet HTTP traffic this would allow us to treat a 20Gb/s full duplex link.

VI. CONCLUSION

In this paper, we introduce a new technique to analyze traffic between clients and web-servers. Unlike existing techniques, this proposal provides the ability to trade some accuracy (in term of what information can be retrieved and the precision of such information) against an increased analysis speed. We believe such analysis speed might be useful against some threats like denial of service attacks where speed is the major concern. This would allow the usage of application level resources to be controlled at the network level.

Although our technique is probably not applicable to every web server (HTTP servers that are very large or contain mostly dynamic content) our feeling is that it would work for a large proportion of existing HTTP servers making it useful in practice. We believe this approach could be further improved by looking at HTTP communications at levels other than the micro-session level looking for example at relations between objects within html documents. We are currently working on DDoS detection methods that might take advantage of the information inferred by *RequIn*.

REFERENCES

[1] H. Nielsen et al.. Network Performance Effects of HTTP/1.1 CSS1, and PNG. In Proceedings of SIGCOMM 1997, August-September 1997.

[2] G. Navarro and M. Raffinot. Flexible Pattern Matching in Strings. Cambridge Univ. Press, 2002.

[3] Sprint IP Monitoring project, available at: ipmon.sprint.com/, 2004.

[4] R. Fielding and al. HTTP 1.1, RFC 2616. Internet Engineering Task Force, June 1999.

[5] Balachander Krishnamurthy, Martin Arlitt, PRO-COW: Protocol Compliance on the Web, A Longitudinal Study, USITS '01, March 26–28 2001.

[6] F. Donelson Smith, F. Hernandez, K. Jeffay, and D. Ott, What TCP/IP protocol headers can Tell Us About the Web, In proceedings of ACM SIGMETRICS 2001, June 2001.

[7] Darren Reed, IPFilter, available at coombs.anu.edu.au/~avalon/, 2004.