SIPPING	V. Gurbani	
Internet-Draft	Bell Laboratories, Alcatel-Lucent	
Intended status: Informational	E. Burger	
Expires: September 10, 2009	This Space for Sale	
	T. Anjali	
	Illinois Institute of Technology	
	H. Abdelnur	
	O. Festor	
	INRIA	
	March 09, 2009	

TOC

The Common Log File (CLF) format for the Session Initiation Protocol (SIP)
DOCNAME

Status of this Memo

Copyright Notice

Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

Well-known web servers such as Apache and web proxies like Squid support event logging using a common log format. The logs produced using these de-facto standard formats are invaluable to system administrators for trouble-shooting a server and tool writers to craft tools that mine the log files and produce reports and trends. Furthermore, these log files can also be used to train anomaly detection systems and feed events into a security event management system. The Session Initiation Protocol does not have a common log format, and as a result, each server supports a distinct log format that makes it unnecessarily complex to produce tools to do trend analysis and security detection. We propose a common log file format for SIP servers that can be used uniformly for proxies, registrars, redirect servers as well as back-to-back user agents.

Table of Contents

---

1. Introduction

TOC

Well-known web servers such as Apache and Squid support event logging using a Common Log Format (CLF), the common structure for logging requests and responses serviced by the web server. It can be argued

that a good part of the success of Apache has been its CLF because it allowed third parties to produce tools that analyzed the data and generated traffic reports and trends. The Apache CLF has been so successful that not only did it become the de-facto standard in producing logging data for web servers, but also many commercial web servers can be configured to produce logs in this format.

The Session Initiation Protocol (Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol," June 2002.) [1](SIP) is an Internet multimedia session signaling protocol that is increasingly used for other services besides session establishment. SIP does not have a CLF today.

As SIP becomes pervasive in multiple business domains and ubiquitous in academic and research environments, it is beneficial to establish a CLF for the following reasons:

*Allows for a common reference for interpreting the state of SIP transactions in SIP servers across multiple vendor implementations and open-source alternatives.

*Allows for the training of anomaly detection systems that once trained can monitor the CLF file to trigger an alarm on the subsequent deviations from accepted patterns in the data set. Currently, anomaly detection systems monitor the network and parse raw packets that comprise a SIP message -- a process that is unsuitable for anomaly detection systems [3] (Rieck, K., Wahl, S., Laskov, P., Domschitz, P., and K-R. Muller, "A Self-learning System for Detection of Anomalous SIP Messages," 2008.). With all the necessary event data at their disposal, network operations managers and information technology operation managers are in a much better position to correlate, aggregate, and prioritize log data to maintain situational awareness.

*Allows independent tool providers to craft tools and applications that interpret the CLF data to produce insightful trend analysis and detailed traffic reports.

*Allows for automatic testing of SIP equipment and establishing a concise and standardized diagnostic trail of a SIP session.

Establishing a CLF for SIP is a challenging task. The behavior of a SIP entity is more complex when compared to the equivalent HTTP entity. Base protocol services such as parallel or serial forking elicit multiple final responses. Ensuing delays between sending a request and receiving a final response all add complexity when considering what fields should comprise a CLF and in what manner. Furthermore, unlike HTTP, SIP groups multiple discrete transactions into a dialog, and these transactions may arrive at a varying inter-arrival rate at a proxy. For example, the BYE transaction usually arrives much after the

corresponding INVITE transaction was received, serviced and expunged from the transaction list. Nonetheless, it is advantageous to relate these transactions such that automata or a human monitoring the log file can construct a set consisting of related transactions.
ACK requests in SIP need careful consideration as well. In SIP, an ACK is a special method that is associated with an INVITE only. It does not require a response, and furthermore, if it is acknowledging a non-2xx response, then the ACK is considered part of the original INVITE transaction. If it is acknowledging a 2xx-class response, then the ACK is a separate transaction consisting of a request only (i.e., there is not a response for an ACK request.) CANCEL is another method that is tied to an INVITE transaction, but unlike ACK, the CANCEL request elicits a final response.
While most requests elicit a response immediately, the INVITE request in SIP can wait at a proxy as it forks branches downstream or at a user agent server while it alerts the user. RFC 3261 (Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, “SIP: Session Initiation Protocol,” June 2002.) [1] instructs the server transaction to send a 1xx-class provisional response if a final response is delayed for more than 200 ms. A SIP SLF log file needs to include such provisional responses because they help train automata associated with anomaly detection systems and provide some positive feedback for a human observer monitoring the log file.
Finally, beyond supporting native SIP actors such as proxies, registrars, redirect servers, and user agent servers (UAS), it is beneficial to derive a CLF format that supports back-to-back user agent (B2BUA) behavior, which may vary considerably depending on the specific nature of the B2BUA.

---

2. Terminology

TOC

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.) [2].
RFC 3261 (Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, “SIP: Session Initiation Protocol,” June 2002.) [1] defines additional terms used in this document that are specific to the SIP domain such as "proxy"; "registrar"; "redirect server"; "user agent server" or "UAS"; "user agent client" or "UAC"; "back-to-back user agent" or "B2BUA"; "dialog"; "transaction"; "server transaction".
This document uses the term "SIP Server" that is defined to include the following SIP entities: user agent server, registrar, redirect server,

a SIP proxy in the role of user agent server, and a B2BUA in the role of a user agent server.

---

3. Relationship between SIP CLF and Call Detail Record

TOC

With the success of SIP in traditional telephony domains, it is tempting to think of the SIP CLF as a replacement for call logs and Call Detail Records (CDRs). However, this is expressly not our intent. The charging system of a telephone exchange produces a CDR. Insofar as a SIP entity is acting as a telephone exchange, it can continue producing CDR irrespective of whether it also produces a SIP CLF.
A SIP CLF is a standardized text file format used by SIP Servers, proxies, and B2BUAs. A SIP CLF is simply an easily digestible log of past and current transactions. It contains enough information to allow humans and automata to derive relationships between discrete transactions handled at a SIP entity. It is amenable to quick parsing (i.e., well-delimited fields) and it is platform and operating system neutral.

---

4. CLF Format

TOC

The inspiration for the SIP CLF format is the Apache CLF. The structure of the Apache CLF, including the format string that appears in the Apache configuration file, is as follows.

```
     %h      %l     %u       %t    \"%r\"   %s    %b
remotehost rfc931 authuser [date] request status bytes
```

remotehost: Remote hostname (or IP number if DNS hostname is not available, or if DNSLookup is Off.

rfc931: The remote logname of the user.

authuser: The username by which the user has authenticated himself.

[date]: Date and time of the request.

request: The request line exactly as it came from the client.

status: The HTTP status code returned to the client.

bytes: The content-length of the document transferred.

Section 5 (A CLF for SIP Servers) outlines the SIP CLF. Section 6 (Proxy Servers and B2BUA Correlation Directives) contains additional logging data elements that correlate forked transactions in proxies or similar transactions at a B2BUA that require correlation.
While based on the Apache CLF, the SIP CLF does not use the RFC 931 (St. Johns, M., "Authentication server," January 1985.) [4] identification service. RFC 931 and its successor, RFC 1413 (St. Johns, M., "Identification Protocol," February 1993.) [5], provide the identity of a user associated with a particular TCP connection. Such a service does not work for SIP because SIP runs over multiple transports. More importantly, in today's networks, firewalls often block access to port 113 (decimal), the port associated with the identification service, rendering it completely useless.
While this document defines the log string in terms of parameter strings (the "%" tokens), this is done in order to facilitate the subsequent discussion only. More specifically, these format strings allow us to describe the format of the SIP CLF file, they must not be used as log configuration strings for individual SIP servers. The definitions in this document for a log file line are the only definition for SIP CLF records.

4.1. ABNF

TOC

> NB: The ABNF below is not formal, nor is it completely closed-end. At this point, we would like to write the data elements down and produce the formal BNF corresponding to them in the next revision.

```
sip-clf = request-clf / response-clf
request-clf = %d %h %u %m %r %f %t %i "%c" %x %y [delim extension]
response-clf = %d %x %y %s %m %t "%c" [delim extension]
delim = "--"
extension = to be defined (other headers) / message-body

Notes:
message-body is defined in RFC3261.
%m in response-clf was added in individual submission -01
   because we need it to differentiate the response of a CANCEL
   from that of an INVITE.  Both CANCELs and INVITEs will have
   the same %x value.
```

4.2. Data Elements

TOC

date (%d):
Date and time of the request as the number of seconds since the Unix epoch. [Note 1]

remotehost (%h): The DNS name or IP address of the upstream client.

authuser (%u): The user name by which the user has been authenticated. If the user name is unknown or when a request is challenged, the value in this field MUST be "-" [Note 2].

method (%m): The name of the SIP method. MUST appear in upper-case in the log file.

request-uri (%r): The Request-URI, including any URI parameters.

from (%f): The From URI, including the tag. Whilst one may question the value of the From URI in light of RFC 4474 (Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)," August 2006.) [6], the From URI, nonetheless, imparts some information. For one, the From tag is important and, in the case of a REGISTER request, the From URI can provide information on whether this was a third-party registration or a first-party one [Note 3].

to (%t): The To URI, including the tag.

callid (%i): The Call-ID value.

status (%s): The SIP response status code returned upstream.

contactlist (%c): Contact URIs in the response, if any. If there are no Contact URIs, the SIP Server MUST log a "-" for the contactlist. If there are multiple URIs, the SIP Server MUST delimit the list of URIs by a commas (,) [Note 4].

server transaction association code(%x): The transaction identifier associated with the server transaction. Implementations MAY reuse the server transaction identifier (the topmost branch-id of the incoming request, with or without the magic cookie), or they MAY generate a unique identification string for a server transaction (this identifier needs to be locally unique to the server only.) This identifier is used to correlate ACKs and CANCELs to an INVITE transaction; it is also used to aid in forking as will be explained in Section 6 (Proxy Servers and B2BUA Correlation Directives).

client transaction association code (%y): This field is used to associate client transactions with a server transaction for forking proxies or B2BUAs. It is explained further in Section 6 (Proxy Servers and B2BUA Correlation Directives).

[Note 1]
Do we need sub-second resolution? A simple solution is to go date.mmm, where mmm is the time in milliseconds. Thoughts?

[Note 2] The realm is not specified here. HTTP CLF does not specify it either, but that does not mean that we cannot do better. Should we? Thoughts...

[Note 3] Go with addr-spec in ABNF here and handle the tag, which is actually a header parameter and not a URI parameter.

[Note 4] Dale pointed out that quotes to delimit Contact may not be adequate because name-addr can contain a quote. Coming up with delimiters for Contact is as yet an open issue.

If a field is not applicable to the event, or if the SIP Server does not know the value of the field, the SIP Server MUST use the field value "-", without the quotes. If the field value in a current event is identical to a field value from a prior event with the same server transaction identifier, the SIP Server MAY use a "+", without the quotes, as the field value to indicate such repetition. The SIP Server MUST NOT use a "+" for the date or server transaction fields.

4.3. Request CLF

TOC

SIP Servers generating a SIP CLF log for a SIP request MUST follow the format string in the following figure. A SIP CLF log entry for a SIP request MUST have these 11 fields in the order listed below. A SIP Server MAY add additional fields after these 12 fields.

```
%d %h %u %m %r %f %t %i "%c" %x %y
```

4.4. Response CLF

TOC

SIP Servers generating a SIP CLF log for a response event MUST follow the format string in the following figure. A SIP CLF log entry for a response event MUST have these five fields in the order listed below. The server transaction (%x) field MUST be present. This allows the automata or observer monitoring the log file to correlate late responses with pending transactions. This construct is also used in Section 6 (Proxy Servers and B2BUA Correlation Directives) to associate responses arriving on client transactions and responses being sent on a server transaction in case of a forking proxy or a B2BUA.

```
%d %x %y %s %m %t "%c"
```

The To header (%t) is part of the response CLF because it contains the tag associated with the recipient of the request; this tag, in turn, is used to identify a dialog.

> TODO: Should we just put the To-tag here instead of the whole URI?

---

5. A CLF for SIP Servers

TOC

A SIP CLF record MUST occupy one physical line in the log file. A line is a string of octets terminated by a CRLF.
A SIP CLF record MUST conform to the patterns described in Section 4.3 (Request CLF) and Section 4.4 (Response CLF). A SIP CLF record may have other fields appended to the patterns described here.
A SIP CLF record for SIP Servers MUST contain exactly one request CLF line (Section 4.3 (Request CLF)) and one or more corresponding response lines (Section 4.4 (Response CLF)). A SIP CLF record for the ACK request MUST NOT contain any corresponding response lines (Section 4.4 (Response CLF)).
Illustrative examples of the SIP CLF follow. These examples use the <allOneLine> tag defined in RFC 4475 (Sparks, R., Hawrylyshen, A., Johnston, A., Rosenberg, J., and H. Schulzrinne, "Session Initiation Protocol (SIP) Torture Test Messages," May 2006.) [7] to logically denote a single line.
In the following example, Alice is registering herself with her domain's registrar and is challenged for HTTP Digest:

```
<allOneLine>
1230756550 192.168.1.2 - REGISTER sip:example.com
sip:alice@example.com;tag=iu8u76 sip:alice@example.com
8719u@example.com - hgt678h -
</allOneLine>

1230756550 hgt678h - 401 REGISTER sip:alice@example.com;tag=8hy -
```

In this example, Alice has successfully authenticated herself with her registrar. The information logged contains her authorized identity as well as the list of Contact URIs that were registered. Note that the last two fields are not populated because there is no need to maintain further state about this REGISTER transaction:

```
<allOneLine>
1230756560 192.168.1.2 alice REGISTER sip:example.com
sip:alice@example.com;tag=iu8u76 sip:alice@example.com;tag=yh78
8719u@example.com
"<sip:alice@lab.example.com>;q=0.7;expires=7200,
<sip:alice@home.example.net>;q=0.5;expires=3600"
hgt679h -
</allOneLine>

<allOneLine>
1230756550 hgt679h - 200 REGISTER +
"<sip:alice@lab.example.com>;q=0.7;expires=7200,
<sip:alice@home.example.net>;q=0.5;expires=3600"
</allOneLine>
```

The next example shows a log file entry from Alice's UAS when it received a MESSAGE request from Bob and responded to it using a 2xx-class response:

```
<allOneLine>
1230756560 192.168.1.10 - MESSAGE sip:alice@example.com
sip:alice@example.com;tag=jki7 sip:bob@example.net
7y16@example.net - 76gr56 -
</allOneLine>

1230756560 76gr56 - 200 MESSAGE sip:bob@example.net;tag=8uy -
```

This example shows a log file entry from Bob's UAS that responded to a 3xx-class response to Alice's session invitation request. Of interest here is the third line that contains an ACK request received by Bob's UAS for the INVITE transaction. See that the %x field of both the log entries match, thereby confirming correlation. Since this is an ACK request corresponding to an existing INVITE transaction, critical information like To, From, Call-ID, etc. remain the same. Thus to save I/O and log file space, the implementation chose to abbreviate the repeated fields with the "+" entries.

```
<allOneLine>
1230756560 192.168.1.10 - INVITE sip:bob@example.net
sip:alice@example.com;tag=iu8u76 sip:bob@example.net
i98ju@example.com - y6y78u -
</allOneLine>

<allOneLine>
1230756560 y6y78u - 302 INVITE sip:bob@example.net;tag=yh78
"<sip:bob@home.example.net>"
</allOneLine>

1230756560 192.168.1.10 - ACK + + + + + y6y78u -
```

The next few examples demonstrate the more complex scenarios corresponding to handling CANCELs and sending delayed responses upstream.
In this example, Bob contacts Alice; Alice's UAS has sent a 180 upstream but has not generated a final response yet. Before Alice has a chance to pick up the phone, Bob hangs up causing a CANCEL to arrive at Alice's UAS. Alice's UAS processes the CANCEL, sending a 200 OK (CANCEL), followed by sending a 487 (INVITE) and receiving an ACK:

```
<allOneLine>
1230756560 192.168.1.10 - INVITE sip:bob@example.net
sip:alice@example.com;tag=iu8u76 sip:bob@example.net
i98ju@example.com "<sip:bob@home.example.net>" y6y78u -
</allOneLine>

1230756560 y6y78u - 100 INVITE sip:bob@example.net;tag=yh78 -
1230756560 y6y78u - 180 INVITE + -
1230756561 192.168.1.10 - CANCEL + + + + - y6y78u -
1230756560 y6y78u - 200 CANCEL + -
1230756561 y6y78u - 487 INVITE sip:bob@example.net;tag=yh78 -
1230756561 192.168.1.10 - ACK + + + + + y6y78u -
```

The following example demonstrates a session being queued and finally answered by the UAS:

```
<allOneLine>
1230756560 192.168.1.10 - INVITE sip:agent@acd.example.net
sip:alice@example.com;tag=iu8u76 sip:agent@acd.example.net
i98ju@example.com - z9hG4bk7yt6 -
</allOneLine>
<allOneLine>
1230756560 z9hG4bk7yt6 - 100 INVITE
sip:agent@acd.example.net;tag=oi8 -
</allOneLine>
1230756560 z9hG4bk7yt6 - 180 INVITE + -
1230756561 z9hG4bk7yt6 - 182 INVITE + -
1230756564 z9hG4bk7yt6 - 182 INVITE + -
1230756565 z9hG4bk7yt6 - 183 INVITE + -
1230756566 z9hG4bk7yt6 - 200 INVITE + -
1230756566 192.168.1.10 - ACK + + + + - z9hG4bk7yt6 -
```

Note that the CLF format is designed such that using the server transaction in a regular expression search will yield a filtered result containing all pertinent entries to that server transaction. This allows the human observer to sift through the file ex post facto to recreate the transaction state or to train anomaly detection automata with a pertinent data set.

---

6. Proxy Servers and B2BUA Correlation Directives

TOC

SIP Proxies may fork, creating several client transactions that correlate to a single server transaction. Responses arriving on these client transactions, or new requests (CANCEL, ACK) sent on the client transaction need log file entries that correlate with a server transaction. Similarly, a B2BUA may create one or more client transactions in response to an incoming request. These transactions will require correlation as well.

We present the correlation directives below. They are best expressed through an example call flow described next.

Let us assume that Bob is running a call-stateful proxy. Alice decides to establish a session with Bob through his proxy. Bob's proxy does a location lookup and decides to fork the request downstream to two destinations. Of these two destinations, one generates a 500-class response while the other generates a 200-class response. Bob's proxy sends an ACK on the branch corresponding to the 500-class response and sends the 200-class one upstream. Since it is call stateful, it will receive an ACK and proxy it downstream.

First, Bob's proxy receives a request from Alice. Bob's proxy creates a server transaction in the log file and sends a 100 upstream:

```
<allOneLine>
1230756560 192.168.1.10 - INVITE sip:bob@example.net
sip:alice@example.com;tag=hy7 sip:bob@example.net
7yhgt1@example.com - uyt67h FORK/-
</allOneLine>

1230756560 uyt67h - 100 INVITE + -
```

Note the last column in the request CLF line: "FORK/-". This entry is the client transaction association code (%y) that the SIP CLF uses to track client transactions. The format of this field is "directive/ client-transaction-id". Directive is either "FORK" or "CLIENT". The server transaction of a proxy or a B2BUA uses "FORK". The client transaction uses "CLIENT". When a server transaction forks, and in this document we consider it forking even if the proxy forwards the request to a single downstream destination, the value of %y MUST be "FORK/-". The client-transaction-id portion of the %y field is an identification string unique to each client transaction sent downstream.
Implementations SHOULD reuse the branch-id value created for the client transaction, with our without the magic cookie. The implementation MAY generate a unique identification string to serve as a client transaction identifier. Such an identification string MUST be unique at that server.
As each client transaction is created and activated, it will be reflected in the log file. There is another idiosyncrasy that needs to be accommodated, however. Because it may take some time for a downstream UAS to elicit a response once contacted, the proxy or B2BUA MUST insert a log file entry when the client transaction reaches the "Proceeding" state upon the receipt of a provisional response. The next two lines demonstrate that a response each was elicited from two forked branches:

```
<allOneLine>
1230756563 - - INVITE sip:bob@home.example.net
sip:alice@example.com;tag=hy7 sip:bob@example.net
7yhgt1@example.com - uyt67h CLIENT/hb76
</allOneLine>
<allOneLine>
1230756564 - - INVITE sip:bob@carphone.example.net
sip:alice@example.com;tag=hy7 sip:bob@example.net
7yhgt1@example.com - uyt67h CLIENT/hb77
</allOneLine>
```

Note that each log file entry for a client transaction contains the CLIENT code and the corresponding client-transaction-id value ("hb76" and "hb77" in the above example.)
The forked branches return multiple provisional responses followed by a final response on each branch.

```
1230756565 uyt67h hb76 100 INVITE sip:bob@example.net;tag=876v -
1230756565 uyt67h hb77 100 INVITE sip:bob@example.net;tag=561t -
1230756565 uyt67h hb76 180 INVITE sip:bob@example.net;tag=876v -
1230756565 uyt67h hb77 180 INVITE sip:bob@example.net;tag=561t -
1230756567 uyt67h hb77 182 INVITE sip:bob@example.net;tag=561t -
1230756568 uyt67h hb76 500 INVITE sip:bob@example.net;tag=876v -
<allOneLine>
1230756568 uyt67h hb77 200 INVITE sip:bob@example.net;tag=561t
"sip:bob@home.example.net"
</allOneLine>
```

Bob's proxy will aggregate these responses and sends the best response (200) upstream and sends an ACK to the branch that returned a 5xx-class response:

```
 <allOneLine>
 1230756569 uyt67h - 200 INVITE sip:bob@example.net;tag=561t
 "sip:bob@home.example.net"
 </allOneLine>
 <allOneLine>
 1230756569 + - ACK sip:bob@home.example.net + + + - uyt67h
 CLIENT/hb76
 </allOneLine>
```

Because it is a stateful proxy, it receives an ACK from the upstream client. It now has to create a client transaction to send this ACK downstream:

```
<allOneLine>
1230756570 192.168.1.10 - ACK sip:bob@home.example.net
sip:alice@example.com;tag=hy7 sip:bob@example.net;tag=76y
7yhgt1@example.com - t6y5 FORK/-
</allOneLine>
<allOneLine>
1230756570 - - ACK sip:bob@home.example.net
sip:alice@example.com;tag=hy7 sip:bob@example.net;tag=76y
7yhgt1@example.com - t6y5 CLIENT/hb89
</allOneLine>
```

The SIP CLF format string includes the minimum set of headers that, we believe, lend themselves to trend analysis and serve as information that may be deemed useful. The inclusion of the To and From tags and the Call-ID as part of the format string allows automata to correlate later transactions to earlier ones, while the addition of the correlation directives similarly allows automata to associate an outgoing transaction with an existing one.

---

7. Security Considerations

TOC

A log file by its nature reveals the both the state of the entity producing it and the nature of the information being logged. To the extent that this state should not be publicly accessible and that the information is to be considered private, appropriate file and directory permissions attached to the log file SHOULD be used. In the worst case, public access to the SIP log file provides the same information that an adversary can gain using network sniffing tools (assuming that the SIP traffic is in clear text.) If all SIP traffic on a network segment is encrypted, then special attention MUST be directed to the file and directory permissions associated with the log file to preserve privacy such that only a privileged user can access the contents of the log file.

The SIP CLF format string includes the minimum set of headers that, we believe, lend themselves to trend analysis and serve as information that may be deemed useful. Other formats can be defined that include more headers (and the body) from Section 4.2 (Data Elements). However, where to draw a judicial line regarding the inclusion of non-mandatory headers can be challenging. Clearly, the more information a SIP server logs, the longer time the logging process will take, the more disk space the log entry will consume, and the more potentially sensitive information could be breached. Therefore, adequate tradeoffs should be taken in account when creating a format string that logs more header fields than the ones recommended by the CLF format string.

We believe that a SIP CLF format will aid in network and situational security. Such a format could be integrated into MITRE's Common Event Expression (CEE (Mitre Corporation, “Common Event Expression,” .) [8]) system, which could monitor the SIP CLF log file to produce CEE events that are fed into a detection system using CEE's common log transport and syntax. It is also possible to imagine a wrapper that takes data from a standard SIP CLF and turns it into an IDMEF-expressible syntax. RFC 4765 (Debar, H., Curry, D., and B. Feinstein, “The Intrusion Detection Message Exchange Format (IDMEF),” March 2007.) [9] defines IDMEF, which are the data formats and exchange procedures for sharing information of interest to intrusion detection and response systems and to the management systems that may need to interact with them.

Implementers need to pay particular attention to buffer handling when reading or writing log files. SIP CLF entries can be unbounded in length. It would be reasonable for a full body dump to be thousands of octets long. This is of particular importance to CLF log parsers, as conforming SIP CLF log writers are free to add fields to the mandatory fields described in this document.

TOC

8. IANA Considerations

This document does not require any considerations from IANA.

---

9. Acknowledgments

TOC

A big debt of gratitude to Dale Worley for a very close read of the draft and many excellent suggestions to the text. Hadriel Kaplan and Robert Sparks provided additional comments.

---

10. References

TOC

10.1. Normative References

TOC

[1]	Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, “SIP: Session Initiation Protocol,” RFC 3261, June 2002 (TXT).
[2]	Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

10.2. Informative References

TOC

[3]	Rieck, K., Wahl, S., Laskov, P., Domschitz, P., and K-R. Muller, “A Self-learning System for Detection of Anomalous SIP Messages,” Principles, Systems and Applications of IP Telecommunications Services and Security for Next Generation Networks (IPTComm), LNCS 5310, pp. 90-106, 2008.
[4]	St. Johns, M., “Authentication server,” RFC 931, January 1985 (TXT).
[5]	St. Johns, M., “Identification Protocol,” RFC 1413, February 1993 (TXT).
[6]	Peterson, J. and C. Jennings, “Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP),” RFC 4474, August 2006 (TXT).
[7]	Sparks, R., Hawrylyshen, A., Johnston, A., Rosenberg, J., and H. Schulzrinne, “Session Initiation Protocol (SIP) Torture Test Messages,” RFC 4475, May 2006 (TXT).
[8]	Mitre Corporation, “Common Event Expression.”
[9]	

Debar, H., Curry, D., and B. Feinstein, "The Intrusion Detection Message Exchange Format (IDMEF)," RFC 4765, March 2007 (TXT).

Authors' Addresses

TOC

	Vijay K. Gurbani
	Bell Laboratories, Alcatel-Lucent
	1960 Lucent Lane
	Naperville, IL 60566
	USA
Email:	vkg@alcatel-lucent.com
	Eric W. Burger
	This Space for Sale
	USA
Email:	eburger@standardstrack.com
URI:	http://www.standardstrack.com
	Tricha Anjali
	Illinois Institute of Technology
	316 Siegel Hall
	Chicago, IL 60616
	USA
Email:	tricha@ece.iit.edu
	Humberto Abdelnur
	INRIA
	INRIA - Nancy Grant Est
	Campus Scientifique
	54506, Vandoeuvre-lès-Nancy Cedex
	France
Email:	Humberto.Abdelnur@loria.fr
	Olivier Festor
	INRIA
	INRIA - Nancy Grant Est
	Campus Scientifique
	54506, Vandoeuvre-lès-Nancy Cedex
	France
Email:	Olivier.Festor@loria.fr