# SIP_LEX

# SIP Malformed Message detection

Raihana Ferdous

University of Trento, Italy

January, 2012

# Abstract

*Detection and prevention of SIP malformed messages has become an important indicator of high availability VoIP systems. This paper describes possible SIP malformed messages attacks and builds a malformed message detection system. It focuses on achieving high detecting accuracy and at the same time low processing overhead. SIP_LEX, lexical analyzer for SIP messages, is implemented for parsing incoming SIP messages in the system to identify the malformed messages. Experiments over synthetic traces demonstrate the efficiency of SIP_LEX in SIP malformed messages detection.*

# Chapter 1

# Introduction

Voice over IP is one of the commercially most important emerging trends in multimedia communications over IP. Network Operators, Consumers and Enterprises are rapidly adopting voice-over-IP (VoIP) technologies for multimedia communication as it provides higher flexibility and more features than traditional telephony infrastructures. VoIP services are generally based on standard protocols for signaling (i.e., SIP, H.323, and MGCP) and transporting media albeit proprietary solutions like Skinny by Cisco and the Open Source IAX adopted by Asterisk project are very popular in enterprise solutions.

SIP (Session Initiation Protocol) is the application layer signaling protocol used for managing multimedia sessions by the most major telecommunication operators. SIP based IP telephony network opens new perspective in telephony arena through its flexibility in call generation, possibility of multiple calls from same URL and low cost. SIP [11] is a text-based application layer signaling protocol standardized by the Internet Engineering Task Force (IETF), and it is designed to support the establishment, maintenance, modification, and termination of multimedia sessions including, but not limited to, IP telephony, video/voice-conference and instant messaging.

As the popularity of VoIP and its deployment grows, it also becomes the target of hackers and crackers. Although SIP received a big interest from the telecommunication community, it threats administrators with significant security challenges. Basically the open architecture of the Internet, heterogeneous environment of VoIP network, use of multiple protocols and the lack of a separated, secure signaling and control plane make the VoIP applications more vulnerable to attacks than traditional architectures.

Though secure transport-layer protocols such as Transport Layer Security (TLS) or Secure RTP (SRTP) have been standardized, they have not been fully implemented and deployed in current VoIP applications because of the overheads of implementation and performance. Moreover, both SIP protocol implementations and network applications are often not fully comply with underlying standards (e.g. RFCs) or they contain development errors in the source implementation code. As a result there are implementation errors that may pollute a network with incorrectly formed packets and lead to unstable conditions. Furthermore, standard protocol implementations usually focus on well-formed messages without considering any defense tactic against malformed messages. 'Malformed message' is referred to any type of invalid or non-standard message, skillfully formed by an attacker in order to exploit and eventually take advantage of any implementation gap or dysfunction might exist in the target system. The victim of malformed message attack is unable to process those message resulting to various undesired situations like crashing the VoIP server and creating Denial of Service (DoS) phenomena. The security threats introduced by malformed message attacks are should be properly understood and development of an efficient malformed message detection mechanism is necessary in order to protect VoIP infrastructures.

Existing work on SIP malformed message detection includes signature-based detection algorithm and machine learning based approaches. Signature-based anomalous message detection system [5] is not suitable for fast processing of large data set. On the other hand, existing machine learning based malformed message detection system [9, 7] suffers the problem of 'curse of dimentionality' in representing SIP message in feature space. Moreover, they also fail to detect shopsticated SIP malformed messages.

In this proposal, we focus on reducing the time complexity and increasing the accuracy of malformed message detection system. We have used machine learning techniques for categorizing SIP messages. In order to increase the accuary of classification, SIP messages are represented into structured data structure, such as syntactic parse tree. This structured representation is significant in identifying shopsticated SIP malformed messages as it contains syntactic information of a SIP message. For classification, tree kernel methods are used on structured SIP messages along with Support Vector Machine [13, 3] , one of the standard tools for machine learning and data mining, which delivers excellent performance in real-world applications such as text categorisation, hand-written character recognition, image classification, biosequences analysis, etc. The advantage of using kernel methods is that we can avoid feature representation problem which is known as 'curse of dimentionality' problem. This is because kernel methods avoid explicit representation of data into feature space and measure the similarity between two structued objects based on the common sub-parts. Experimental results illustrate the efficiency of using tree kernel for SIP malformed message detection on a large set of SIP messages.

The rest of the document is organized as following: Section 2 presents the State of the Art of the research, Section 3 describes SIP malformed message attack, Section 4 represents the proposed malformed message detection system, Section 5 presents our efforts and preliminary results of the proposed system and Section **??** indicates our future works and concludes this paper.

# Chapter 2

# State of the Art

Classic IDS system like SNORT [10] usually attempts to describe all possible malicious messages by storing their signatures in a static signature database. Afterwards, every network stream is examined against all stored signatures or rules for possible matches. Such an approach lacks flexibility and scalability.

On the contrary, this [5] malformed message detection mechanism adopts the opposite approach. They define a specific 'signature' considering the syntax of well-formed SIP message defined in he standard [11] and, thus, any message that does not comply with that 'signature' must be discarded as 'malformed' message. The authors show that this approach is far more efficient in recognizing malformed messages. The computational complexity of signature-based malformed message detection system depends on the number of rule/signature in the signature database. Thus signature-based systems may become inefficient for large set of SIP messages.

SIP being a text-based protocol, its content can be described in terms of textual tokens and words, thus, can be characterized by frequencies of contained substrings. This opens the possibility of considering the problem of SIP malformed message identification as a text classification problem and applying Machine Learning methods for text classification to solve this problem. [9] considered SIP malformed message detection problem as a text categorization problem and to solve this problem each SIP messages is mapped to a feature vector and identifies anomalous content by determining deviation from a model of normality. Thus, representation of SIP messages using a set of features may have two shortcomings, (i) identification of important features to characterize SIP message is complicated, and (ii) it may require too many feature which introduces the problem of 'curse of dimensionality'.

Kernel methods can be used to avoid major feature representation problems as kernel methods allow us to express the similarity between two objects without explicitly defining their feature space. Moreover, kernel methods are widely used to extend the applicability of traditional classification methods (e.g., Support Vector Machine, Principal component analysis, etc). The idea of using kernel methods for SIP malformed message detection introduces two different, but related lines of research.

First, incoming SIP messages are required to be transformed into data-structure that is applicable for learning/classification algorithms. As we intend to detect syntactic errors of SIP messages, so messages should be represented in suitable data structure that includes the syntactic information of SIP messages.

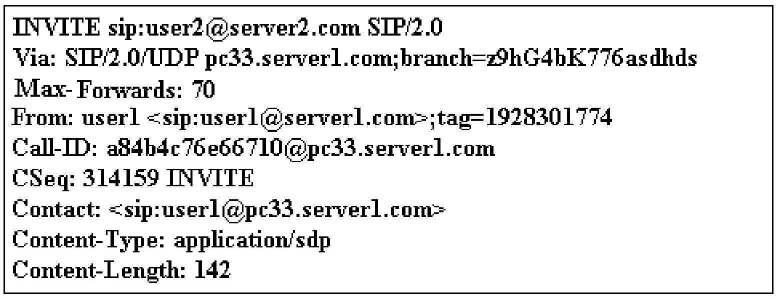
Second, the computational complexity of kernel evaluation does not depend on the size of the feature space but on the complexity of the kernel function. So identification of suitable kernel function for this problem is complicated. A fast and efficient kernel method should be used for this problem to fulfill the requirement of on-line malformed message identification.

# Chapter 3

# SIP Malformed Message Attack

This section discusses possible SIP malformed message [12].

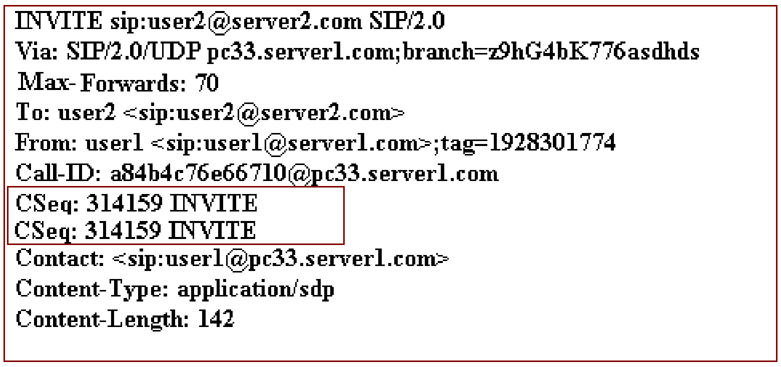
- **Missing of Mandatory field** - For each SIP message there are some mandatory fields. A SIP message can be considered as 'malfored' due to lack of any of those fields. Figure 3.1 shows an example of malformed SIP INVITE message where mandatory field 'To' is missing.



```
INVITE sip:user2@server2.com SIP/2.0
Via: SIP/2.0/UDP pc33.server1.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
From: user1 <sip:user1@server1.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.server1.com
CSeq: 314159 INVITE
Contact: <sip:user1@pc33.server1.com>
Content-Type: application/sdp
Content-Length: 142
```

Figure 3.1: SIP INVITE malfored message where mandatory field 'To' is missing.

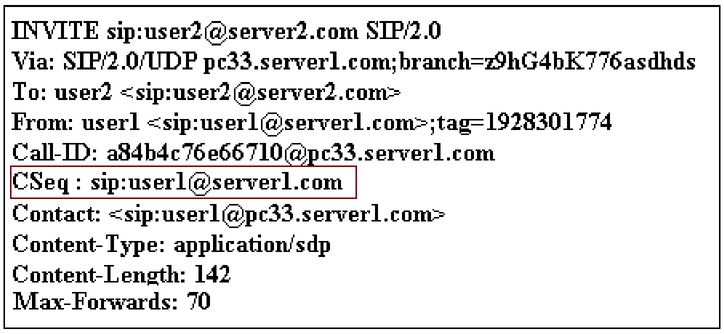
- **Duplicate entry of unique field** - SIP message headers must have contain some fields which are both marked mandatory and unique. Consequently, if any of these headers appears more than once at the incoming SIP message, then this message must be considered as malicious. Figure 3.2 shows an example of malformed SIP INVITE message where mandatory field 'To' is missing.



```
INVITE sip:user2@server2.com SIP/2.0
Via: SIP/2.0/UDP pc33.server1.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: user2 <sip:user2@server2.com>
From: user1 <sip:user1@server1.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.server1.com
CSeq: 314159 INVITE
CSeq: 314159 INVITE
Contact: <sip:user1@pc33.server1.com>
Content-Type: application/sdp
Content-Length: 142
```

Figure 3.2: SIP INVITE malfored message where mandatory and unique field 'CSeq' appears twice.

- **Error in header format** - Each header of SIP message has specific structure which must be followed. Header fields that do not match with SIP grammar must be considered as malicious. Figure 3.3 and 3.4 shows examples of malformed SIP INVITE message where the message format of mandatory field 'CSeq' is wrong.

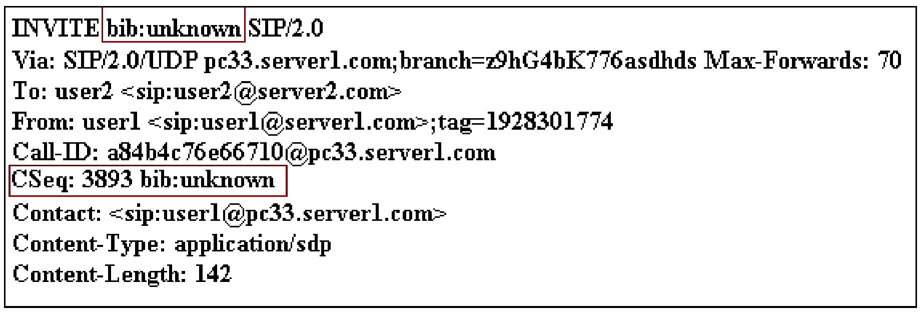


```
INVITE sip:user2@server2.com SIP/2.0
Via: SIP/2.0/UDP pc33.server1.com;branch=z9hG4bK776asdhds
To: user2 <sip:user2@server2.com>
From: user1 <sip:user1@server1.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.server1.com
CSeq : sip:user1@server1.com
Contact: <sip:user1@pc33.server1.com>
Content-Type: application/sdp
Content-Length: 142
Max-Forwards: 70
```

Figure 3.3: Syntax error in 'From' header field in SIP INVITE message.

- **Error in hierarchical structure of SIP message** - For all messages, the general format is:

  A start line
  One or more header fields

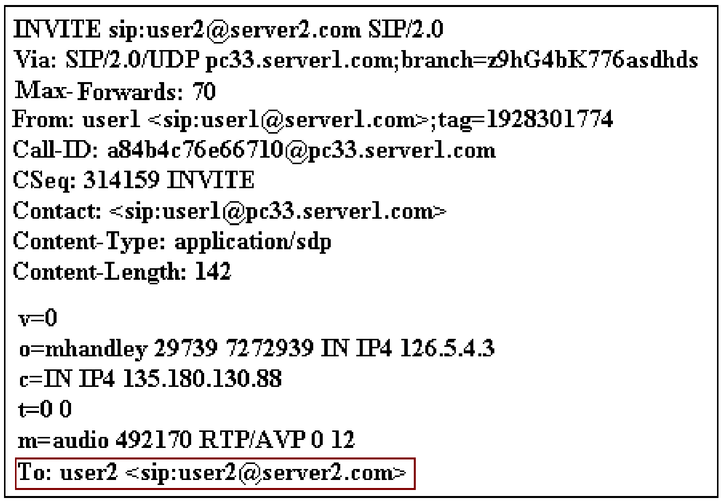


```
INVITE bib:unknown SIP/2.0
Via: SIP/2.0/UDP pc33.server1.com;branch=z9hG4bK776asdhds Max-Forwards: 70
To: user2 <sip:user2@server2.com>
From: user1 <sip:user1@server1.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.server1.com
CSeq: 3893 bib:unknown
Contact: <sip:user1@pc33.server1.com>
Content-Type: application/sdp
Content-Length: 142
```

Figure 3.4: Syntax error in ’From’ header field in SIP INVITE message.

An empty line
A message body (optional)

Any SIP message that does not follow above general structure can be considerd as malformed. Figure 3.5 shows an example of such malformed SIP INVITE message. Hierarchical structure of well-formed SIP message is not



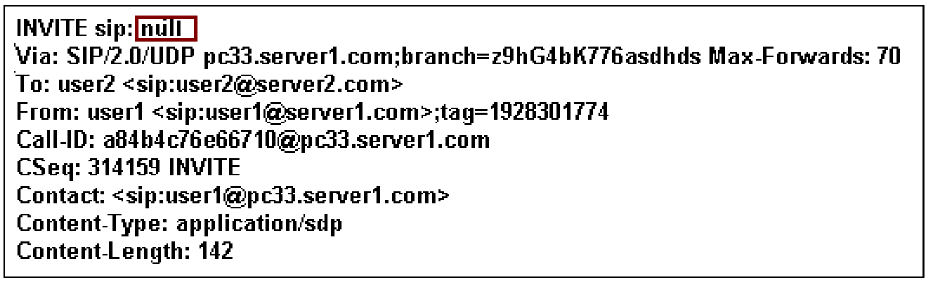
```
INVITE sip:user2@server2.com SIP/2.0
Via: SIP/2.0/UDP pc33.server1.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
From: user1 <sip:user1@server1.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.server1.com
CSeq: 314159 INVITE
Contact: <sip:user1@pc33.server1.com>
Content-Type: application/sdp
Content-Length: 142

v=0
o=mhandley 29739 7272939 IN IP4 126.5.4.3
c=IN IP4 135.180.130.88
t=0 0
m=audio 492170 RTP/AVP 0 12
To: user2 <sip:user2@server2.com>
```

Figure 3.5: Error in hierarchical structure of SIP message.

followed in SIP message of figure 3.5 as mandatory header field ’To’ comes after message body.

- **Null entry for any mandatory field** - A SIP message can be considered as malfored having ’Null’ entry in any of the mandatory fields. Figure 3.6

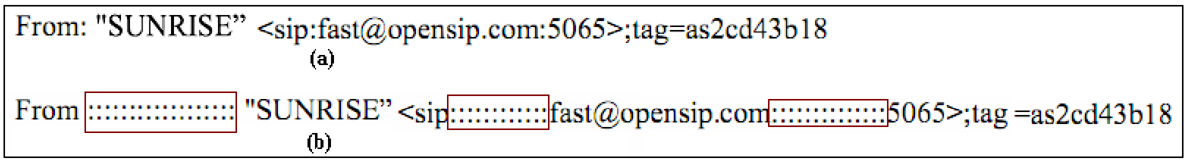
shows an example of malformed SIP INVITE message where mandatory field 'To' is missing.



```
INVITE sip:null
Via: SIP/2.0/UDP pc33.server1.com;branch=z9hG4bK776asdhds Max-Forwards: 70
To: user2 <sip:user2@server2.com>
From: user1 <sip:user1@server1.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.server1.com
CSeq: 314159 INVITE
Contact: <sip:user1@pc33.server1.com>
Content-Type: application/sdp
Content-Length: 142
```

Figure 3.6: SIP INVITE malfored message where valude of mandatory field 'To' is 'null'.

- **Presence of large formatted string/ansi-escape character/invalid UTF-8 sequences** - Large sequences of formatted strings/ansi-escape character/invalid UTF-8 sequences can cause buffer overflow in the SIP server. Figure 3.7 and 3.8 shows an example of SIP messages with large sequence of formatted string/ansi-escape character/invalid UTF-8 sequences.

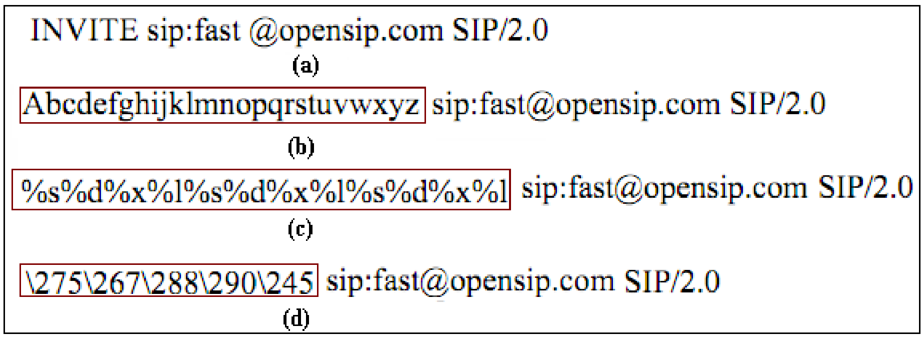


```
From: "SUNRISE" <sip:fast@opensip.com:5065>;tag=as2cd43b18
(a)

From :::::::::::::::: "SUNRISE" <sip:::::::::::fast@opensip.com::::::::::::5065>;tag =as2cd43b18
(b)
```

Figure 3.7: SIP malfored message where large sequence of unnecessary character has appeared.

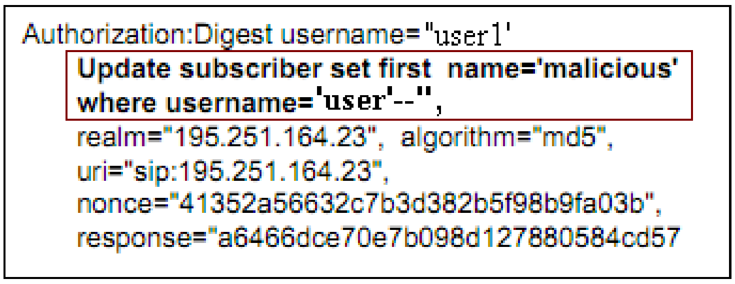
Figure 3.6 shows an example of malformed SIP INVITE message where mandatory field 'To' is missing.

- **SQL message injection** - Another case that can be seen as a malformed message attack is that of SIP messages embedding SQL code in their authorization header as illustrated in Figure 3.9.



```
INVITE sip:fast @opensip.com SIP/2.0
(a)
Abcdefghijklmnopqrstuvwxyz sip:fast@opensip.com SIP/2.0
(b)
%s%d%x%l%s%d%x%l%s%d%x%l sip:fast@opensip.com SIP/2.0
(c)
\275\267\288\290\245 sip:fast@opensip.com SIP/2.0
(d)
```

Figure 3.8: SIP malfored message where large sequence of unnecessary character has appeared.



```
Authorization:Digest username="user1'
     Update subscriber set first  name='malicious'
     where username='user'--'',
     realm="195.251.164.23",  algorithm="md5",
     uri="sip:195.251.164.23",
     nonce="41352a56632c7b3d382b5f98b9fa03b",
     response="a6466dce70e7b098d127880584cd57
```

Figure 3.9: Example of a malformed message that contains SQL code.

# Chapter 4

# Kernel methods for SIP malformed message detection

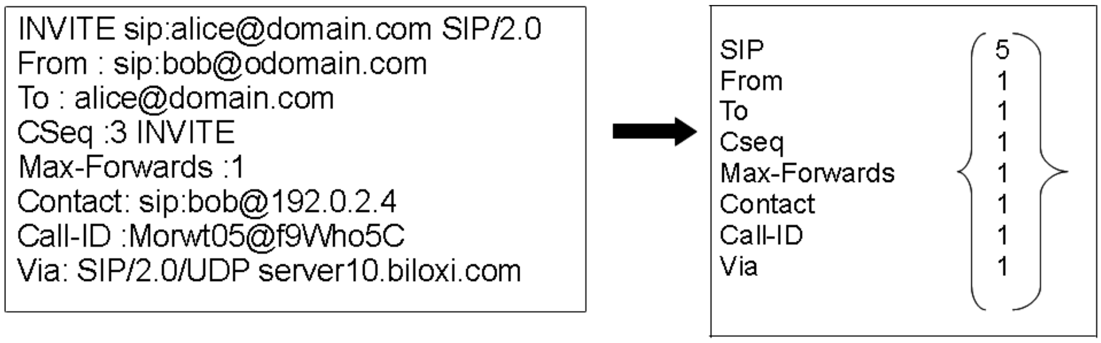
The goal of this work is to develop a fast and efficient SIP malformed messages detection system. Description of the proposed system is illustrated in the following sub-sections.

## 4.1 Structured representation of SIP messages

SIP being a text-based protocol, malformed message classification problem can be considered as a text categorization problem. In text categorization (TC) systems machine learning techniques are applied to documents represented by term vectors (i.e.bag-of-words representation), which is still the most popular choice in text mining tasks. According to these models, the input documents are encoded as vectors whose dimensions correspond to the terms in the overall training corpus. The inner product (or the cosine) between two such vectors is used as kernel hence making the similarity of two documents dependant only on the amount of terms they share.

[9] applied this method of text classification to solve the problem of SIP malformed message identification. The authors mapped SIP messages to a feature vector and identifies anomalous content by determining deviation from a model of normality. A set of sub-strings/terms S is defined to model the content of SIP messages and this set S is called the "feature-set". Given a feature string s∈S and a SIP message x, the number of occurrences of the feature s in message x

can be determined. SIP messages are embedded into feature space where messages are expresses by feature s∈S and its frequencies. For instance, figure 4.1(a) shows an example of well formed SIP INVITE message. In the INVITE mes-



```
INVITE sip:alice@domain.com SIP/2.0
From : sip:bob@odomain.com
To : alice@domain.com
CSeq :3 INVITE
Max-Forwards :1
Contact: sip:bob@192.0.2.4
Call-ID :Morwt05@f9Who5C
Via: SIP/2.0/UDP server10.biloxi.com
```

| | |
|---|---|
| SIP | 5 |
| From | 1 |
| To | 1 |
| Cseq | 1 |
| Max-Forwards | 1 |
| Contact | 1 |
| Call-ID | 1 |
| Via | 1 |

Figure 4.1: SIP INVITE message.

sage in figure 4.1(a), a set of features can be defined where "To", "From", "SIP", "Via", "Max-Forwards", "Call-ID", "CSeq", and "Contact" can be considered the most significant terms/features. After embedding to feature space, SIP INVITE message of figure 4.1(a) can be represented as in figure 4.1(b).

However, this flat-representation (e.g., bag-of-words) of SIP messages is not suitable for classifying SIP malformed messages. One of the main shortcomings of this kind of representation is that it fails to encode the syntactic structure of the input text and generally too many terms/features are required to represent the input text. For instance, detection of malformed message where the hierarchical structure of SIP message structure does not follow (e.g., figure 3.5 in section 3) will not possible with malformed message detection system proposed in [9].

For identification of sophisticated SIP malformed message, we need more complex and non-linear data structure that can model syntactic information of incoming SIP messages. Syntactic parse tree, an ordered, rooted tree that represents the syntactic structure of a string according to some formal grammar, can be considered the most significant structured representation of SIP messages. SIP messages structure is defined in RFC 3261 [11] using Augmented BackusNaur Form (ABNF) [4] which is capable of describing the syntax of languages used in computing, such as communication protocols. Here, each node with its children is associated with a SIP message rule defined in SIP standard [11]. Leaves of parse tree are the terms/words from the incoming SIP messages. For example, figure 4.2 illustrates the syntactic parse tree of a well-formed SIP INVITE message defined in SIP standard [11].

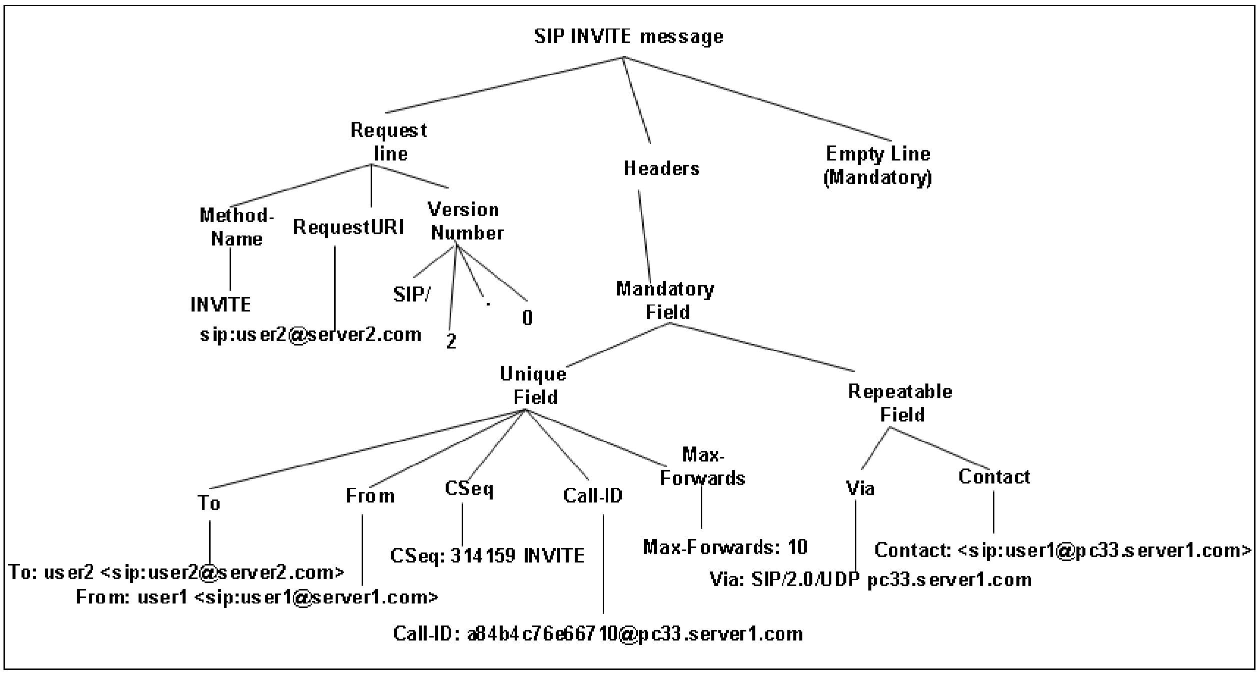


Figure 4.2: Parse tree of a well-formed SIP INVITE message according to SIP grammar [11].

## 4.2 Kernel method for SIP malformed message detection

After representation of incoming SIP messages into structured data-structure like parse trees, efficient classification algorithm should be applied for malformed message classification. Classifiers are not generally designed in order to apply to structured data, thus, classical machine learning approaches attempt to represent structural objects (e.g., tree, graph) by using a flat feature representation, i.e. attribute-value vectors. However, this raises two problems,

- Identifying proper features to represent structural properties is complecated and is prone to loss of relevant information.

- It may be computational demanding as it may require too many features to represent structured data.

To solve this problem, kernel method for stuctured data can be used which is able to express the similarity between two structured objects without explicitly defining their feature space. As a result we do not have major feature representation problems. Tree Kernels have been proposed as a powerful framework to exploit structured data such as parse trees of the input texts and are widely used for natureal language processing (NLP). Particularly, convolution kernel framework proposed in [6] is the most popular methodologies for designing kernels for structured data such as trees. Convolution kernels are based on the idea that a complex object can be described in terms of its constituent parts, for example a tree can be described in terms of its subtrees, thus, a convolution kernel measures the similarity of two objects in terms of the similarities of their subparts. For instance, measuring the similarity between the parse trees of two noun phrases: “a dog” and “a cat” can be computed using convolution kernel methods. Syntactic parse trees for these two phrases are found in figure 4.3. Figure 4.3 represents all the substructures of the two parse “a dog” and “a cat”. It is found that only 3 structures (out of 5) are completely identical between two parse trees, and, thus, the similarity is equal to 3.

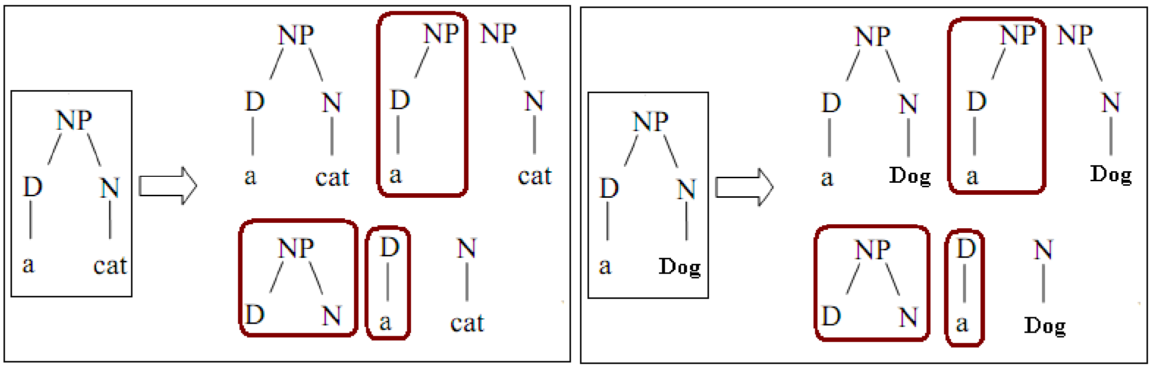


Figure 4.3: Syntactic parse trees for natural languages (a)’a cat’, (b)’a dog’.

### 4.2.1 Tree Kernel

The tree kernel proposed in [2] have been designed based on the idea of representing trees in terms of all their substructures (fragments). The job of the kernel function is then to efficiently count the number of tree substructures that are common to both argument trees.

Let's consider a $T_1$ and $T_2$ are two trees in which m different sub-trees (fragments) are present. Here, the sub-trees are identified as features and each feature, i.e. subset tree[1], can be indexed by an integer between 1 and m. So, the feature space can be defined as: $F = \{f_1, f_2, .........., f_m\}$.

Each tree is represented by a m dimensional vector where the $i^{th}$ component counts the number of occurrences of the $i^{th}$ sub-tree. To count the frequency of sub-trees in the tree the function $h\left(T\right)$ can be defined which represents the number of occurrences of the $i^{th}$ tree fragment in tree a tree T. Now, trees $T_1$ and $T_2$ can be represented as feature vectors $\phi\left(T_1\right)$ and $\phi\left(T_2\right)$, where,
$\phi\left(T_1\right) = \left[h_1\left(T1\right), h_2\left(T1\right), h_3\left(T1\right), ......, h_m\left(T1\right)\right]$
$\phi\left(T_2\right) = \left[h_1\left(T2\right), h_2\left(T2\right), h_3\left(T2\right), ......, h_m\left(T2\right)\right]$

The inner product between two trees $T_1$ and $T_2$ under the representation $\phi\left(T_1\right)$ and $\phi\left(T_2\right)$ can be defined as:

$$K\left(T_1, T_2\right) = \phi\left(T_1\right) \cdot \phi\left(T_2\right) = \sum_{m}^{i=1} \left[h_i\left(T_1\right) \cdot h_i\left(T_2\right)\right] \tag{4.1}$$

Thus the sub-tree kernel (ST) defines a similarity measure between trees which is proportional to the number of shared subset trees. [2] proposed a dynamic programming algorithm that computes the inner product between two trees in polynomial (in the size of the trees involved) time.

We first define the set of nodes in trees $T_1$ and $T_2$ as $N_{T_1}$ and $N_{T_2}$ respectively. The Indicator function is $I_i\left(n\right)$ and can be defined as:

$$I_i\left(n\right) = \begin{cases} 1 & \text{; if sub-tree i is seen rooted at node n} \\ 0 & \text{; otherwise} \end{cases}$$

[1]Sub-Tree: A sub-tree (ST) is any node of a tree along with all its descendants.

It follows that:

$$h_i\left(T_1\right) = \sum_{n_1 \in N_{T_1}} I_i\left(n_1\right)$$

and

$$h_i\left(T_2\right) = \sum_{n_2 \in N_{T_2}} I_i\left(n_2\right)$$

Substituting $h_i\left(T_1\right)$ and $h_i\left(T_2\right)$ the kernel function of equation 4.1 can be written as below:

$$\begin{aligned} K\left(T_1, T_2\right) &= \sum_{m}^{i=1} \left[h_i\left(T_1\right) \cdot h_i\left(T_2\right)\right] & (4.2)\\ &= \sum_{m}^{i=1} \sum_{n_1 \in N_{T_1}} I_i\left(n_1\right) \sum_{n_2 \in N_{T_2}} I_i\left(n_2\right) & (4.3)\\ &= \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \sum_{m}^{i=1} I_i\left(n_1\right) I_i\left(n_2\right) & (4.4)\\ &= \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta\left(n_1, n_2\right) & (4.5) \end{aligned}$$

Author of [2, 8] provide a recursive deifinition for computing $\Delta\left(n_1, n_2\right)$ in natural language processing. For efficiently computing $\Delta\left(n_1, n_2\right)$ we modified the recursive algorithm proposed in [2, 8], as here tree kernel are used for pattern analysis of SIP message where leaf nodes are also included. The definition for computing $Delta\left(n_1, n_2\right)$ is given below:

- If the productions(number of children) at $n_1$ and $n_2$ are different, then $\Delta\left(n_1, n_2\right) = 0$.
- If $n_1$ and $n_2$ are leaves and their associated symbols are equal, then $\Delta\left(n_1, n_2\right)$ $= 1$.
- Else if the productions at $n_1$ and $n_2$ are the same and $n_1$ and $n_2$ are not leaves, then

$$\Delta\left(n_1, n_2\right) = \prod_{nc(n_1)}^{j} \Delta\left(c_j^{n_1}, c_j^{n_2}\right)$$

Here, $nc(n_1)$ is the number of children of node $n_1$ in the tree $T_1$; because the productions at $n_1/n_2$ are the same, so we have, $nc(n_1) = nc(n_2)$. Again, $c_j^{n_1}$ is the $j_{th}$ child-node of $n_1$.

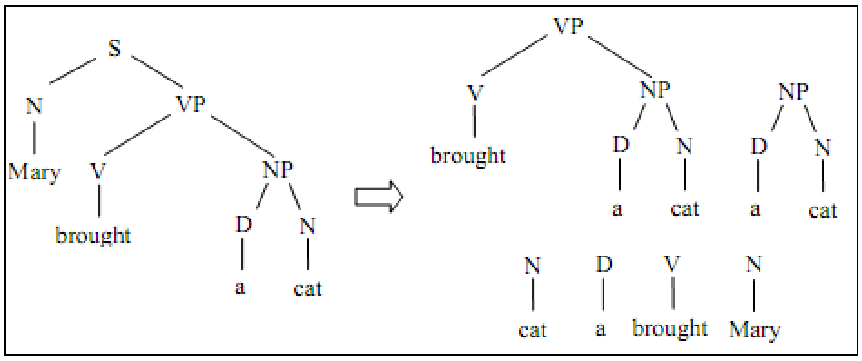
By recursively applying these above rules, it can be concluded whether the sub-tree rooted at $n_1$ and $n_2$ are identical or not.

### 4.2.2 Fast Tree Kernel

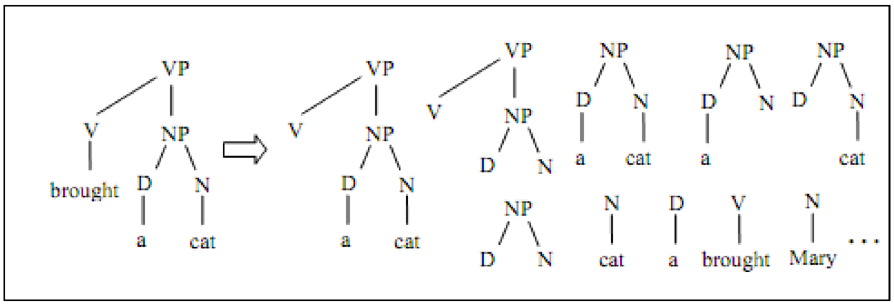
Tree kernel proposed in [2] compute the kernel by summing the $\Delta(n_1, n_2)$ function for each pair $\langle n_1, n_2 \rangle \in N_{T_1} \times N_{T_2}$ of equation 4.5. The computational complexity of this [2] tree kernel is $O(|N_{T_1}| \times |N_{T_2}|)$ and it is really high for large trees.

To reduce the computational complexity of kernel evalutation, [8] slightly modified the kernel function proposed in [2] and proposes an algorithm for the evaluation of the tree kernel which runs in linear average time. [8] proposes to avoid the evaluation of $\Delta(n_1, n_2)$ function (equation 4.5) when the productions associated with $n_1$ and $n_2$ are different, since it is 0. This fast algorithm computes the kernels between two syntactic parse trees in $O(m + n)$ average time, where m and n are the number of nodes in the two trees. The fast algorithm proposed in [8] proves effective for natural language processing. To make tree kernels suitable for SIP messages classification and to reduce the time complexity of kernel evaluation following methodologies can be applied:

- The basic idea of tree kernels is to measure the similarity of fragments of trees where fragments can be subtrees or subset-trees. A sub-tree (ST) is any node of a tree along with all its descendants. A subset-tree (SST) is a more general structure since its leaves can be non-terminal symbols. Figure 4.4 shows the parse tree of the sentence “Mary brought a cat” together with its 6 sub-trees (ST) figure 4.4(a) and 10 subset trees/SST (out of 17) figure 4.4(b). For using tree kernels in natural language processing [2, 8], either sub-trees or subset-trees can be considered as fragments of a tree. In case of using tree kernels for SIP malformed message identification, sub-trees would be suitable fragments to be considered. Though subset-trees provide learning algorithms with richer information but too many irrelevant subset-trees also may occur overfitting and may decrease the classification accuracy.



(a) Sub-trees of a tree.



(b) Subset trees of a tree.

Figure 4.4: (a)Sub-trees and (b)Subset-trees of a parse tree.

- Every incoming SIP message is represented into syntactic tree. In our problem domain, all we need to identify syntactic trees that does not match SIP grammar. So, we do not need to consider all possible sub-trees for every two trees in the dataset. Instead, a set of most significant sub-trees from syntactic tree of a well-formed SIP message can be defined and kernel evaluation between syntactic trees (incoming SIP messages) can be acomplished based on these pre-defined sub-trees. Consideration of only m sub-trees ($m \in N$, where N = all possible subtrees of tree $T_1$ and $T_2$), reduces the time complexity of kernel evaluation.

### 4.2.3 Example of Tree Kernel for SIP malformed message detection

This section describes an example of detecting SIP malformed message using our proposed methodology. For simplicity, all examples of SIP INVITE messages in this sub-section follow only the basic mandatory structure (optional header fields and SIP message body are ignored) in this example. Parse tree of a well-formed SIP INVITE message is shown in figure 4.5. In figure 4.5, leaves include well-formed

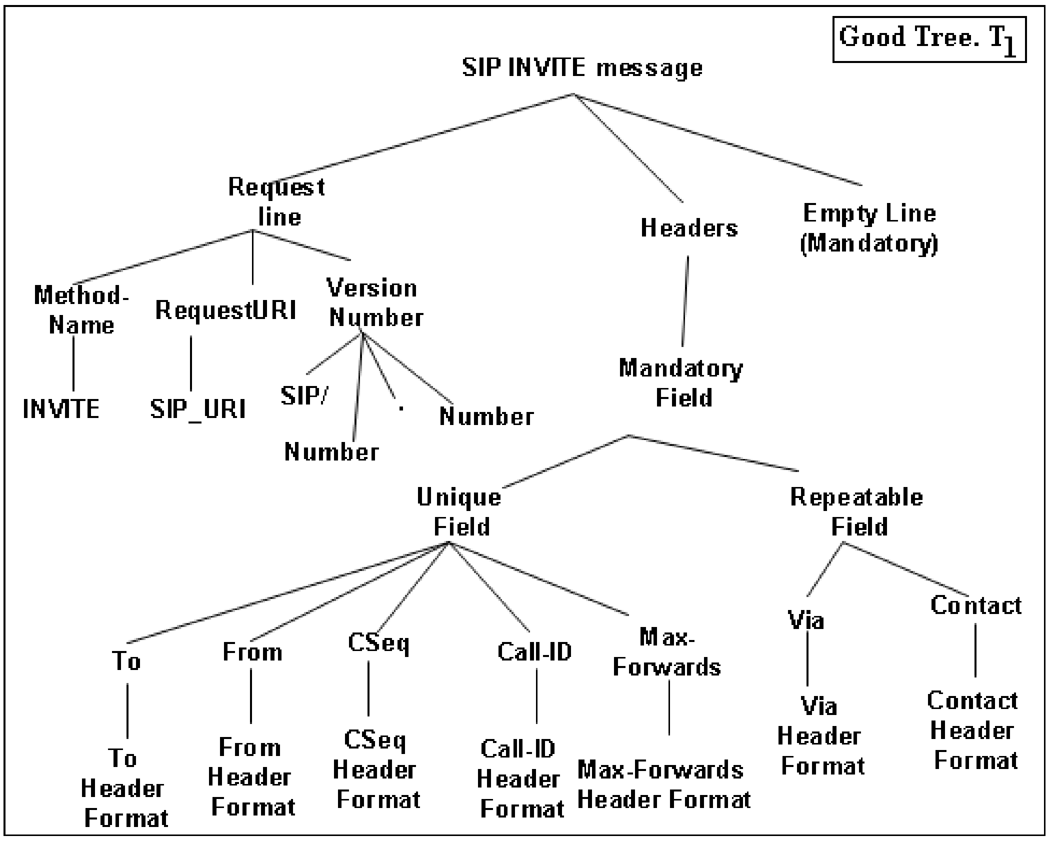


Figure 4.5: Parse tree of a well-formed SIP INVITE message according to SIP grammar in RFC 3261[11].

structures of pre-terminal (node associated with leave nodes) fields. For instance, leave "CSeq Header Format" which is associated with pre-terminal "CSeq", expresses structure of the well-formed "CSeq" header in a SIP INVITE message. The parse tree of "CSeq Header Format" in figure 4.6 is formed following SIP grammar defined using Augmented BackusNaur Form (ABNF) [4]. Grammar of all headers

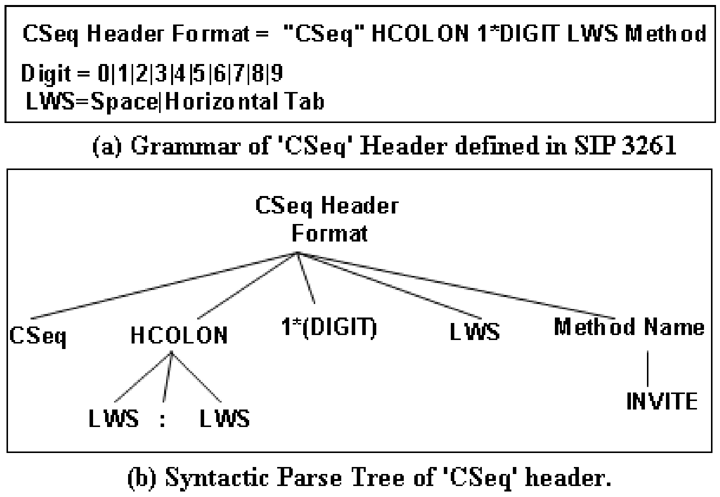

CSeq Header Format = "CSeq" HCOLON 1*DIGIT LWS Method
Digit = 0|1|2|3|4|5|6|7|8|9
LWS=Space|Horizontal Tab
(a) Grammar of 'CSeq' Header defined in SIP 3261
CSeq Header Format
CSeq
HCOLON
1*(DIGIT)
LWS
Method Name
LWS
:
LWS
INVITE
(b) Syntactic Parse Tree of 'CSeq' header.

Figure 4.6: Parse tree of a well-formed “CSeq Header Format”.

are found in [11]. Parse tree of an incoming SIP invite message is shown in figure 4.7 where leaves of the tree are the terms/words from the incoming SIP messages. After representation of SIP messages into syntactic trees, tree kernel can be applied on both trees for measuring the similarity between them. Here, parse tree of figure 4.5 can be defined as $T_1$ and parse tree of incoming SIP message of figure 4.7 is $T_2$. As described in previous section 4.2.2, instead of considering all sub-trees of tree $T_1$ and $T_2$, a set of defined sub-trees we will considered for computing the kernel function. Figure 4.8 shows the set of sub-trees which is defined considering the SIP message syntax defined in [11].

Tree kernel can measure the similarity between two trees $T_1$ and $T_2$ by counting the number of sub-trees (defined in figure 4.8) they share. It is found that, trees $T_1$ and $T_2$ are not identical as they do not share all the sub-trees (sub-tree $t_8$ and sub-tree $t_{10}$ does not match). As tree $T_1$ is formed according to SIP message structure [11] and tree $T_2$ does not match with tree $T_1$, thus, we can conclude tree $T_2$ is a malformed SIP INVITE message. SIP messages of tree $T_2$ contains error in mandatory header fields 'To' and 'CSeq'. Figure 4.9 represents trees $T_1$ and $T_2$ into feature vectors where features are the sub-trees defined in figure 4.8.

In this way, our proposed malformed detection system transforms a huge amout of incoming SIP messages into feature vectors and then using Support Vector

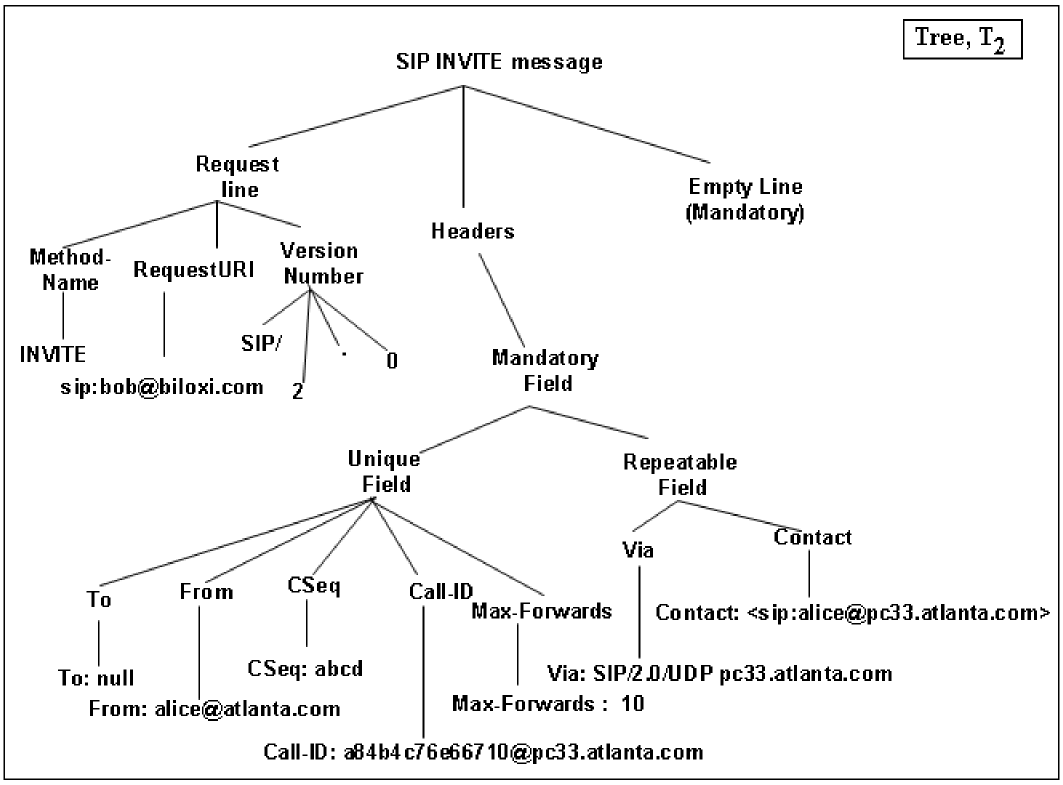


Figure 4.7: An incoming SIP message is transformed into a parse tree.

Machine classifies into two categories, (i) well-formed, and (ii) malformed.

## 4.3 Architecture and Implementation of SIP malformed message detection system

The implemented SIP malformed message detection system is defined as SIP_LEX. It includes a lexical analyzer followed by support vector machine. The lexical analyzer examines the incoming SIP traces and discard syntactically incorrect SIP messages. The rest of the syntactically valid SIP messages are passed to a another classifier which used support vector machine and kernel methods. LibSVM [1], a Library for Support Vector Machines, is used for classification. This second classifier identifies messages that are syntactically valid but are malformed. Figure ?? represents the working block of SIP_LEX.

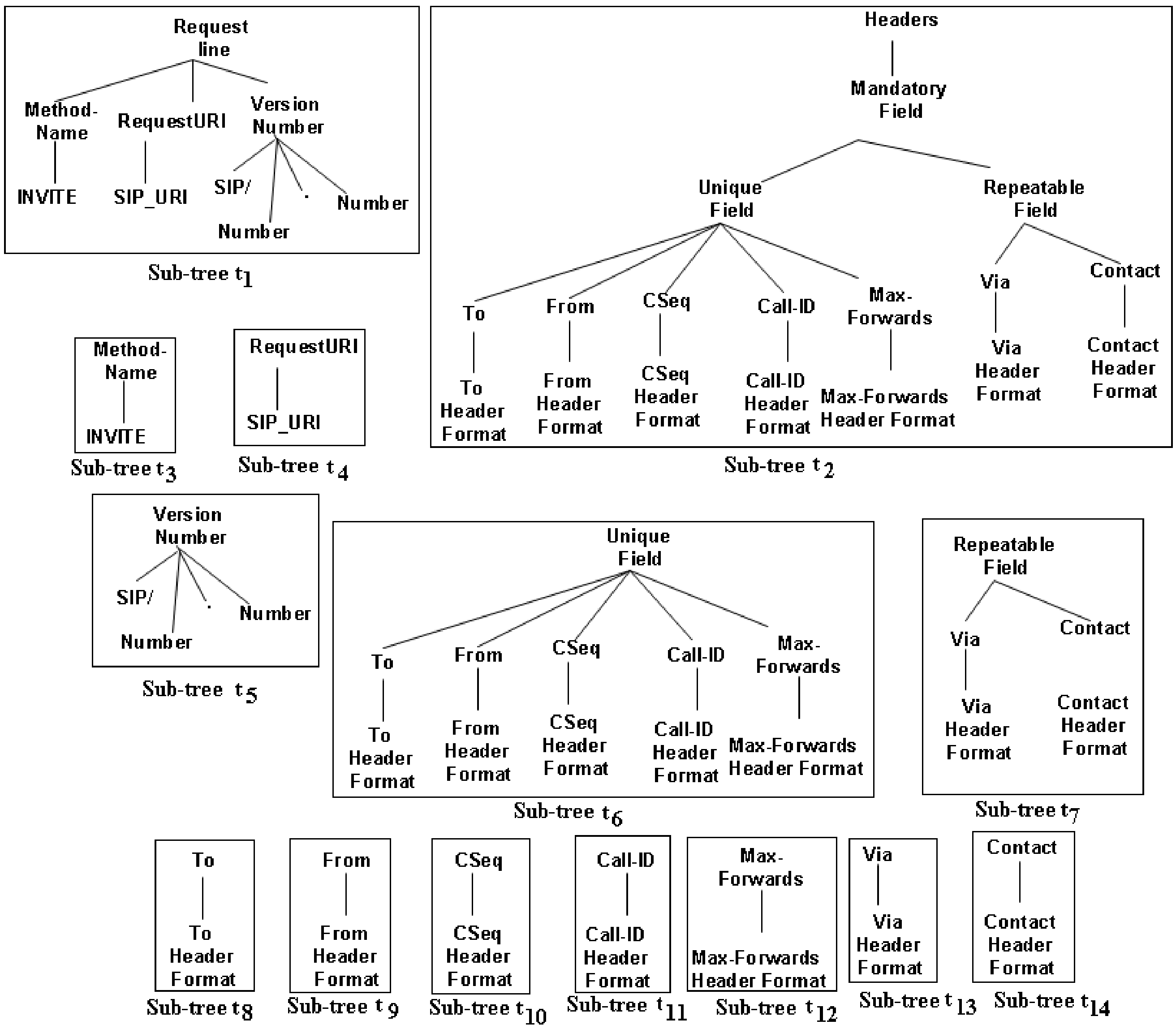


Figure 4.8: Set of sub-trees for evaluation of kernel function over structured SIP messages.

$$\boxed{h_{t_i}(Tj) = \text{Occurance of } i^{th} \text{ sub-tree in tree } Tj}$$

$$T1 = \{ h_{t_1}(T1), h_{t_2}(T1), h_{t_3}(T1), h_{t_4}(T1), h_{t_5}(T1), h_{t_6}(T1), h_{t_7}(T1), h_{t_8}(T1), h_{t_9}(T1), h_{t_{10}}(T1), h_{t_{11}}(T1), h_{t_{12}}(T1), h_{t_{13}}(T1), h_{t_{14}}(T1) \}$$
$$= \{1,1,1,1,1,1,1,1,1,1,1,1,1,1\}$$

$$T2 = \{ h_{t_1}(T2), h_{t_2}(T2), h_{t_3}(T2), h_{t_4}(T2), h_{t_5}(T2), h_{t_6}(T2), h_{t_7}(T2), h_{t_8}(T2), h_{t_9}(T2), h_{t_{10}}(T2), h_{t_{11}}(T2), h_{t_{12}}(T2), h_{t_{13}}(T2), h_{t_{14}}(T2) \}$$
$$= \{1,1,1,1,1,1,1,0,1,0,1,1,1,1\}$$

Figure 4.9: Tree $T_1$ and $T_2$ represented in feature space.

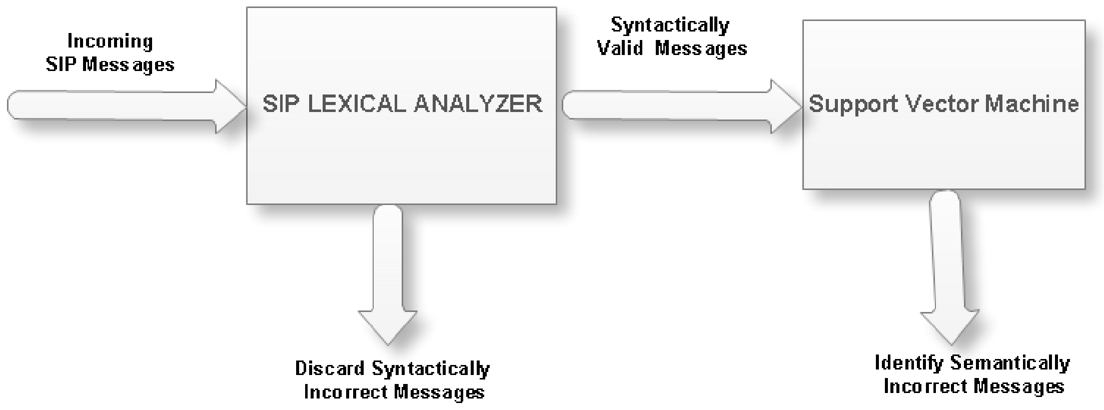

Incoming SIP Messages
SIP LEXICAL ANALYZER
Syntactically Valid Messages
Support Vector Machine
Discard Syntactically Incorrect Messages
Identify Semantically Incorrect Messages

Figure 4.10: Architecture of SIP_LEX.

# Chapter 5

# Experiments and Results

This section describes the performance of LEX_SIP. Our goal is to maximize detection accuracy of SIP malformed message and at the same time to reduce messsage processing time. Infact, time complexity plays a vital role in measuring the effectiveness of an intrusion detection system in a real-time environment like VoIP.

## 5.1 DataSet

Performance evaluation of LEX_SIP relies on large scale of SIP traces (individual SIP request/response messages engaged in a session). But reliable real world VoIP traces are not always available as VoIP providers are not willing to distribute their data due to user privacy agreements. Moreover, VoIP traces with attack information are not so frequent. Considering this situation, we developed a Synthetic generator 'SIP-Msg-Gen' for generating SIP traces. 'SIP-Msg-Gen' is capable of generating SIP malformed messages following the SIP torture test messages defined in RFC 4475 [12]. Scenarios for malformed messages generated by synthetic generator for performance evaluation are found in table 5.1. Possitive examples are generated following well-formed SIP message structure defined in RFC 3261 [11].

Table 5.1: Malformed messages generated by synthetic generator

| | |
|---|---|
| Scenario 1 : | Error in Request line |
| Scenario 2 : | Syntax error in one mandatory header field |
| Scenario 3 : | Syntax error in multiple mandatory header field |
| Scenario 4 : | Syntax error in optional header field |
| Scenario 5 : | Syntax error in message body |
| Scenario 6 : | Missing mandatory header field |
| Scenario 7 : | Duplicate entry for unique header field |
| Scenario 8 : | Missing empty line after header fields |
| Scenario 9 : | Presence of garbage string/invalid character in message |
| Scenario 10 : | Hierarchical disorder of message structure |
| Scenario 11 : | Negative value for mandatory field 'CSeq' or 'Max-Forward' |
| Scenario 12 : | Invalid method name |
| Scenario 13 : | Message Body with unknown Content-Type |
| Scenario 14 : | Unknown Authorization/Accept scheme |
| Scenario 15 : | Message with outrange values for scaler fields |
| Scenario 16 : | Message with multiple request method |

## 5.2 Performance Evaluation - Efficiency of LEX_SIP

Efficiency of LEX_SIP is evaluated by examining the results achieved from LEX_SIP for SIP messages defined in RFC 4475 [12]. RFC 4475 [12] does not attempt to catalog every way to make an invalid message, instead it tries to focus on areas that have caused interoperability problems or that have particularly unfavorable characteristics if they are handled improperly. Following subsections summarize experimental results of LEX_SIP for messages defined in RFC 4475.

### 5.2.1 Valid Message - SIP torture messages

RFC 4475 [12] defines various SIP test messages which are valid but only design to "torture" the malformed message detector. A sample SIP "torture" message is found in table 5.2 and also the results of LEX_SIP for this message. The sample message (table 5.2) contains :

- Escaped characters within quotes.
- An empty subject.

Table 5.2: INVITE with Invalid Via and Contact Headers

| **SIP Message** | **Type** | **LEX_SIP** |
|---|---|---|
| INVITE sip:vivekg@chair-dnrc.example.com;unknownparam SIP/2.0<br>TO : sip:vivekg@chair-dnrc.example.com ; tag = 1918181833n<br>from : “J. Rosenberg”” <sip:jdrosen@example.com> ; tag = 98asjd8<br>MaX-fOrWaRdS: 0068<br>Call-ID: 0ha0isndaksdj@192.0.2.1<br>Content-Length : 151<br>cseq: 0009 INVITE<br>Via : SIP / 2.0 /UDP 192.0.2.2;branch=390skdjuw<br>s :<br>NewFangledHeader: newfangled value continued newfangled value<br>UnknownHeaderWithUnusualValue: ;;,,;;,;<br>Content-Type: application/sdp<br>v: SIP / 2.0 /TCP spindle.example.com ; branch = z9hG4bK9ikj8 , SIP / 2.0 / UDP 192.168.255.11; branch=9hG4bK30239<br>m: “user%”” ¡sip:jdrosen@example.com¿ ; new-param = newvalue ; secondparam ; q = 0.33<br>v=0<br>o=mhandley 29739 7272939 IN IP4 192.0.2.3<br>s=-<br>m=audio 492170 RTP/AVP 0 12 | Valid | Valid |

- LWS (Linear White Space) between colons, semicolons, header field values, and other fields.
- Mix or short and long form for the same header field name.
- Unknown header fields.
- Unusual header field ordering.
- Unusual header field name character case/ unknown parameters of a known header field/ uri parameter with no value/ header parameter with no value.
- Integer fields (Max-Forwards and CSeq) with leading zeros

Table 5.3: INVITE with Invalid Via and Contact Headers

| **SIP Message** | **Type** | **LEX_SIP** |
|---|---|---|
| INVITE sip:user@example.com SIP/2.0<br>To: sip:j.user@example.com<br>From: sip:caller@example.net;tag=134161461246<br>Max-Forwards: 7<br>Call-ID: badinv01.0ha0isndaksdjasdf3234nas<br>CSeq: 8 INVITE<br>**Via: SIP/2.0/UDP 192.0.2.15;;,;,,**<br>**Contact: "Joe" <sip:joe@example.org>;;;;**<br>Content-Length: 153<br>Content-Type: application/sdp | Malformed | Malformed |

## 5.2.2 Invalid Message - Syntectically Invalid

RFC 4475 [12] mentions various examples and clearly calls out what makes any message invalid/incorrect.

### Extraneous header field separators

Presence of extraneous header field separators (e.g., comma & semicolon without any tag parameter) in any header field makes the message **syntactically invalid**. Any server receiving this request should respond with a 400 Bad Request error. An invalid message with extraneous header field separators is shown in table 5.3 where the Via and Contact header fields of this request contain contain additional semicolons and commas without parameters or values.

### Negative/non-numeric value for Content-Length

A SIP INVITE message with a negative/non-numeric value for Content-Length is a syntactically invalid message. An element receiving this message should respond with an error. Table 5.4 shows a SIP INVITE message with negative value for Content-Length and also the results that received from LEX_SIP for this message.

Table 5.4: INVITE with Incorrect Content-Length Header

| **SIP Message** | **Type** | **LEX_SIP** |
|---|---|---|
| INVITE sip:user@example.com SIP/2.0<br>Max-Forwards: 254<br>To: sip:j.user@example.com<br>From: sip:caller@example.net;tag=32394234<br>Call-ID: ncl.0ha0isndaksdj2193423r542w35<br>CSeq: 0 INVITE<br>Via: SIP/2.0/UDP 192.0.2.53 ;branch = z9hG4bKkdjuw<br>Contact: <sip:caller@example53.example.net><br>Content-Type: application/sdp<br>**Content-Length: -999** | Malformed | Malformed |

Table 5.5: INVITE with Error in Display Name in "To" Header

| **SIP Message** | **Type** | **LEX_SIP** |
|---|---|---|
| INVITE sip:user@example.com SIP/2.0<br>**To: "Mr.J.User <sip:j.user@example.com>**<br>From: sip:caller@example.net;tag=93334<br>Max-Forwards: 10<br>Call-ID: quotbal.aksdj<br>Contact: <sip:caller@host59.example.net><br>CSeq: 8 INVITE<br>Via: SIP/2.0/UDP 192.0.2.59:5050; branch = z9hG4bKkdjuw39234<br>Content-Type: application/sdp<br>Content-Length: 153 | Malformed | Malformed |

### Unterminated quoted string in display-name

SIP messages with unterminated quote in the display name are syntactically invalid messages. An element receiving this request should return an 400 Bad Request error. Table 5.5 shows a SIP INVITE message with an unterminated quote in the display name of the To field and result of LEX_SIP for this message.

### <> enclosing Request-URI

A SIP INVITE request message should be considered invalid if the Request-URI is enclosed within in "<>". This message should be rejected with error as a 400

Table 5.6: INVITE with Illegal Enclosing of Request-URI in <>

| SIP Message | Type | LEX_SIP |
|---|---|---|
| **INVITE <sip:user@example.com> SIP/2.0**<br>To: sip:user@example.com<br>From: sip:caller@example.net;tag=39291<br>Max-Forwards: 23<br>Call-ID: ltgtruri.1@192.0.2.5<br>CSeq: 1 INVITE<br>Via: SIP/2.0/UDP 192.0.2.5<br>Contact: <sip:caller@host5.example.net><br>Content-Type: application/sdp<br>Content-Length: 160 | Malformed | Malformed |

Table 5.7: INVITE with illegal space within Request-URI

| SIP Message | Type | LEX_SIP |
|---|---|---|
| **INVITE sip:user@example.com; lr SIP/2.0**<br>To: sip:user@example.com<br>From: sip:caller@example.net;tag=39291<br>Max-Forwards: 23<br>Call-ID: ltgtruri.1@192.0.2.5<br>CSeq: 1 INVITE<br>Via: SIP/2.0/UDP 192.0.2.5<br>Contact: <sip:caller@host5.example.net><br>Content-Type: application/sdp<br>Content-Length: 160 | Malformed | Malformed |

Bad Request. Table 5.6 shows a malformed INVITE request message where the Request-Uri is enclosed with <>.

### Malformed SIP Request-URI (embedded LWS)

An INVITE message with illegal LWS within the Request-URI is invalid. An element receiving this request should respond with a 400 Bad Request. Table 5.7 shows a sample INVITE message with illegal space withing Request-URI.

### Multiple SP separating Request-Line elements

An INVITE message becomes invalid if it contains illegal multiple SP characters between elements of the start line. This type of request message can be rejected

Table 5.8: INVITE with illegal (>1) SP between elements of Request URI

| SIP Message | Type | LEX_SIP |
|---|---|---|
| **INVITE sip:user@example.com SIP/2.0**<br>To: sip:user@example.com<br>From: sip:caller@example.net;tag=39291<br>Max-Forwards: 23<br>Call-ID: ltgtruri.1@192.0.2.5<br>CSeq: 1 INVITE<br>Via: SIP/2.0/UDP 192.0.2.5<br>Contact: <sip:caller@host5.example.net><br>Content-Type: application/sdp<br>Content-Length: 160 | Malformed | Malformed |

Table 5.9: INVITE with an illegal SIP Date format

| SIP Message | Type | LEX_SIP |
|---|---|---|
| INVITE sip:user@example.com SIP/2.0<br>To: sip:user@example.com<br>From: sip:caller@example.net;tag=39291<br>Max-Forwards: 23<br>Call-ID: ltgtruri.1@192.0.2.5<br>CSeq: 1 INVITE<br>Via: SIP/2.0/UDP 192.0.2.5<br>Contact: <sip:caller@host5.example.net><br>Content-Type: application/sdp<br>**Date: Fri, 01 Jan 2010 16:00:00 EST**<br>Content-Length: 160 | Malformed | Malformed |

as malformed. Table 5.8 shows a sample INVITE message where the Request-line contains extra space (>1).

### Invalid timezone in Date header field

An INVITE message is considered invalid if it contains a non GMT time zone in the SIP Date header field and can be rejected as malforme. RFC 3261 [11] explicitly defines the only acceptable timezone designation as "GMT". "UT", "UTC", and "UCT" are invalid. Table 5.9 shows a sample INVITE message with illegal SIP date format.

Table 5.10: INVITE with extra spaces within addr-spec "<>"

| **SIP Message** | **Type** | **LEX_SIP** |
|---|---|---|
| INVITE sip:user@example.com SIP/2.0<br>**To: "Mr.J.User" < sip:user@example.com >**<br>From: sip:caller@example.net;tag=39291<br>Max-Forwards: 23<br>Call-ID: ltgtruri.1@192.0.2.5<br>CSeq: 1 INVITE<br>Via: SIP/2.0/UDP 192.0.2.5<br>Contact: <sip:caller@host5.example.net><br>Content-Type: application/sdp<br>Content-Length: 160 | Malformed | Malformed |

### Extra spaces within addr-spec "<>"

A request message is malformed if the addr-spec between "<>" contains extra spaces. This message can be rejected with a 400 Bad Request response. Table 5.10 shows a sample SIP INVITE message where the SIP-URI in the To header field (enclosed with <>) contains extra spaces.

### Non-token characters in display-name

An request message is malformed if the display names in header fields contain unquoted non-token characters. It is reasonable to always reject this kind of error with a 400 Bad Request response. Table 5.11 shows an illegal SIP INVITE message where the display names in the "To" and "From" headers contain non-token characters but are unquoted.

### Unknown Method in "CSeq" header field

A SIP message is considered malformed if it contains an unknown method name in "CSeq" header field. Table 5.15 shows a malformed SIP INVITE message where the "CSeq" header field contains unknown method name.

Table 5.11: INVITE with an Unquoted Display Name Containg Non-Token Characters

| SIP Message | Type | LEX_SIP |
|---|---|---|
| INVITE sip:t.watson@ieee.org SIP/2.0<br>Via: SIP/2.0/UDP c.bell-tel.com:5060; branch=z9hG4bKkdjuw<br>Max-Forwards: 70<br>**From: Bell, Alexander <sip:a.g.bell@bell-tel.com>;tag=43**<br>**To: Watson, Thomas <sip:t.watson@ieee.org>**<br>Call-ID: 31415@c.bell-tel.com<br>CSeq: 1 INVITE<br>Contact: <sip:a.g.bell@bell-tel.com> | Malformed | Malformed |

Table 5.12: INVITE with unknown method in "CSeq" header field

| SIP Message | Type | LEX_SIP |
|---|---|---|
| INVITE sip:user@example.com SIP/2.0<br>To: "Mr.J.User" <sip:user@example.com><br>From: sip:caller@example.net;tag=39291<br>Max-Forwards: 23<br>Call-ID: ltgtruri.1@192.0.2.5<br>**CSeq: 1 NewMethod**<br>Via: SIP/2.0/UDP 192.0.2.5<br>Contact: <sip:caller@host5.example.net><br>Content-Type: application/sdp<br>Content-Length: 160 | Malformed | Malformed |

### 5.2.3 Invalid Message - Syntactically valid but malformed

#### Missing Required Header Fields

Any request message without mandatory header fields such as "Call-ID", "From", "To", "CSeq" is malformed. Table 5.13 shows a sample INVITE message with missing mandatory header fields. An element receiving this message will respond with a 400 Bad Request error.

Table 5.13: INVITE with missing mandatory header fields.

| **SIP Message** | **Type** | **LEX_SIP** |
|---|---|---|
| INVITE sip:user@example.com SIP/2.0 | | |
| To: “Mr.J.User” <sip:user@example.com> | | |
| From: sip:caller@example.net;tag=39291 | | |
| Max-Forwards: 23 | | |
| Call-ID: ltgtruri.1@192.0.2.5 | Malformed | Malformed |
| **CSeq: 1 OPTION** | | |
| Via: SIP/2.0/UDP 192.0.2.5 | | |
| Contact: <sip:caller@host5.example.net> | | |
| Content-Type: application/sdp | | |
| Content-Length: 160 | | |

Table 5.14: INVITE with Incorrect “Content-Length” Header

| **SIP Message** | **Type** | **LEX_SIP** |
|---|---|---|
| INVITE sip:user@example.com SIP/2.0 | | |
| To: “Mr.J.User” <sip:user@example.com> | | |
| From: sip:caller@example.net;tag=39291 | | |
| Max-Forwards: 23 | | |
| Call-ID: ltgtruri.1@192.0.2.5 | Malformed | Malformed |
| CSeq: 1 INVITE | | |
| Via: SIP/2.0/UDP 192.0.2.5 | | |
| Contact: <sip:caller@host5.example.net> | | |
| Content-Type: application/sdp | | |
| Content-Length: 9999 | | |
| | | |
| v=0 | | |
| o=mhandley 29739 7272939 IN IP4 192.0.2.155 | | |
| s=- | | |
| c=IN IP4 192.0.2.155 | | |
| t=0 0 | | |
| m=audio 492170 RTP/AVP 0 12 | | |
| a=rtpmap:31 LPC | | |

### Content length larger than message

A request message can be considered malformed if the Content Length is larger than the length of the body. The receiving element should respond with a 400 Bad Request error. Table **??** shows a sample SIP INVITE message.

Table 5.15: INVITE with overlarge values for scaler field

| **SIP Message** | **Type** | **LEX_SIP** |
|---|---|---|
| INVITE sip:user@example.com SIP/2.0<br>To: “Mr.J.User” <sip:user@example.com><br>From: sip:caller@example.net;tag=39291<br>**Max-Forwards: 300**<br>Call-ID: ltgtruri.1@192.0.2.5<br>**CSeq: 36893488147419103232 INVITE**<br>Via: SIP/2.0/UDP 192.0.2.5<br>**Contact: <sip:user@host129.example.com>**<br>**;expires=280297596632815**<br>Content-Type: application/sdp<br>Content-Length: 160 | Malformed | Malformed |

### Request scalar fields with overlarge values

A request message with overlarge values for various scaler fields is syntactically valid but it is a malformed one. An element receiving requests contains several scalar header field values outside their legal range should respond with a 400 Bad Request error. Some of the scaler fields are follows:

- CSeq sequence number is $> 2^{32-1}$.
- Max-Forwards value is >255.
- Expires value is $> 2^{32-1}$.
- Contact expires parameter value is $> 2^{32-1}$.

### Start line and CSeq method mismatch

A request message is invalid if it contains mismatching values for the method in the start line and the CSeq header field. Any element receiving this request will respond with a 400 Bad Request Table 5.16 shows a sample INVITE message which has a different method name in “CSeq” header field.

### Unknown authorization scheme

A REGISTER request should be considered invalid if it contains an Authorization header field with an unknown scheme.

Table 5.16: INVITE with different method in "CSeq" header field

| **SIP Message** | **Type** | **LEX_SIP** |
|---|---|---|
| INVITE sip:user@example.com SIP/2.0<br>To: "Mr.J.User" <sip:user@example.com><br>From: sip:caller@example.net;tag=39291<br>Max-Forwards: 23<br>Call-ID: ltgtruri.1@192.0.2.5<br>**CSeq: 1 OPTION**<br>Via: SIP/2.0/UDP 192.0.2.5<br>Contact: <sip:caller@host5.example.net><br>Content-Type: application/sdp<br>Content-Length: 160 | Malformed | Malformed |

Table 5.17: REGISTER with Unknown authorization scheme

| **SIP Message** | **Type** | **LEX_SIP** |
|---|---|---|
| REGISTER sip:example.com SIP/2.0<br>To: sip:j.user@example.com<br>From: sip:j.user@example.com;tag=87321hj23128<br>Max-Forwards: 8<br>Call-ID: 0ha0isndaksdj<br>CSeq: 9338 REGISTER<br>Via: SIP/2.0/TCP 192.0.2.253;branch=z9hG4bKkdjuw<br>Contact: <sip:caller@host5.example.net><br>Authorization: NoOneKnowsThisScheme opaque data here<br>Content-Length:0 | Malformed | Malformed |

### Multiple values in single value required fields

Any request message with multiple Call-ID, To, From, Max-Forwards and CSeq values should be considered malformed. Though this type of message is syntactically valid but an element receiving this request would respond with a 400 Bad Request error. Table 5.18 shows a sample INVITE message with duplicate "To" and "From" headers.

Table 5.18: INVITE with duplicate “To” and “From” header fields

| **SIP Message** | **Type** | **LEX_SIP** |
|---|---|---|
| INVITE sip:user@example.com SIP/2.0<br>To: “Mr.J.User” <sip:user@example.com><br>From: sip:caller@example.net;tag=39291<br>To: “Mr.J.User” <sip:user@example.com><br>Max-Forwards: 23<br>Call-ID: ltgtruri.1@192.0.2.5<br>From: sip:caller2@example.net;tag=39291<br>CSeq: 1 INVITE<br>Via: SIP/2.0/UDP 192.0.2.5<br>Contact: <sip:caller@host5.example.net><br>Content-Type: application/sdp<br>Content-Length: 160 | Malformed | Malformed |

Table 5.19: INVITE with multiple request-line

| **SIP Message** | **Type** | **LEX_SIP** |
|---|---|---|
| **INVITE sip:user@example.com SIP/2.0**<br>To: sip:user@example.com<br>From: sip:caller@example.net;tag=39291<br>Max-Forwards: 23<br>Call-ID: ltgtruri.1@192.0.2.5<br>CSeq: 1 INVITE<br>Via: SIP/2.0/UDP 192.0.2.5<br>Contact: <sip:caller@host5.example.net><br>**Content-Type: application/unknownformat**<br>Content-Length: 160<br><br>**INVITE sip:user@example.com SIP/2.0**<br>To: sip:user@example.com<br>From: sip:caller@example.net;tag=39291 | Malformed | Malformed |

### Multiple SIP Request values

Any message with multiple request-line is considered malformed. Table **??** shows a request message with multiple request line.

Table 5.20: INVITE with unknown content type

| SIP Message | Type | LEX_SIP |
|---|---|---|
| **INVITE sip:user@example.com SIP/2.0**<br>To: sip:user@example.com<br>From: sip:caller@example.net;tag=39291<br>Max-Forwards: 23<br>Call-ID: ltgtruri.1@192.0.2.5<br>CSeq: 1 INVITE<br>Via: SIP/2.0/UDP 192.0.2.5<br>Contact: <sip:caller@host5.example.net><br>**Content-Type: application/unknownformat**<br>Content-Length: 160<br><audio><br><pcmu port="443"/><br></audio> | Malformed | Malformed |

### Unknown Content-Type

Table 5.20 shows a sample INVITE request message which contains a body of unknown type. Though this request message is syntactically valid but the an endpoint receiving this request would reject it with a 415 Unsupported Media Type error.

# Bibliography

[1] C.-C. Chang and C.-J. Lin. Libsvm : A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[2] M. Collins and N. Duffy. New ranking algorithms for parsing and tagging: kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 263–270, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics (ACL).

[3] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines: and other kernel-based learning methods.* Cambridge University Press, 2000.

[4] D. Crocker and P. Overell. Augmented bnf for syntax specifications: Abnf. RFC 2234 (Proposed Standard), November 1997.

[5] T. D. C. L. Dimitris Geneiatakis, Georgios Kambourakis and S. Gritzalis. A framework for detecting malformed messages in sip networks. *Local and Metropolitan Area Networks (LANMAN), 2005.*

[6] D. Haussler. Convolution kernels on discrete structures. Technical report, University of California, Santa Cruz, 1999. Technical Report UCSC-CRL-99-10.

[7] H. Li, H. Lin, H. Hou, and X. Yang. An efficient intrusion detection and prevention system against sip malformed messages attacks. *Computational Aspects of Social Networks, International Conference on*, 0:69–73, 2010.

[8] A. Moschitti. Making tree kernels practical for natural language learning. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.

[9] K. Rieck, S. Wahl, P. Laskov, P. Domschitz, and K.-R. Mller. A self-learning system for detection of anomalous sip messages. In *Principles, Systems and Applications of IP Telecommunications. Services and Security for Next Generation Networks*, Lecture Notes in Computer Science.

[10] M. Roesch. Snort ids. an open source NIDS, August 2001. http://www.snort.org/.

[11] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Sip: Session initiation protocol. RFC 3261 (Proposed Standard), June 2002.

[12] R. Sparks, A. Hawrylyshen, A. Johnston, J. Rosenberg, and H. Schulzrinne. Session initiation protocol (sip) torture test messages. RFC 4475 (Informational, May 2006.

[13] V. Vapnik, S. E. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In *Advances in Neural Information Processing Systems 9*, pages 281–287. MIT Press, 1996.