Department of
Information Engineering
and Computer Science
DISI

UNIVERSITY
OF TRENTO - Italy

DISI - Via Sommarive, 5 - 38123 POVO, Trento - Italy
http://disi.unitn.it

CLASSIFICATION OF SIP MESSAGES BY A SYNTAX FILTER AND SVMs (Extended Version)

Raihana Ferdous, Renato Lo Cigno, Alessandro Zorat

July 2012

Technical Report # DISI-12-027

A short version of this Technical Report appears in the proceedings of the IEEE Global Communications Conference (GLOBECOM 2012), Anaheim, CA, USA, December 3-7, 2012.

Classification of SIP Messages by a Syntax Filter and SVMs* (Extended Version)

Raihana Ferdous, Renato Lo Cigno, Alessandro Zorat
Department of Information Engineering and Computer Science – DISI
University of Trento – Italy
Email: {ferdous, locigno, zorat}@disi.unitn.it

***Abstract*—The Session Initiation Protocol (SIP) is at the root of many sessions-based applications such as VoIP and media streaming that are used by a growing number of users and organizations. The increase of the availability and use of such applications calls for careful attention to the possibility of transferring malformed, incorrect, or malicious SIP messages as they can cause problems ranging from relatively innocuous disturbances to full blown attacks and frauds. To this end, SIP messages are analyzed to be classified as "good" or "bad" depending on whether this structure and content are deemed acceptable or not. This paper presents a classifier of SIP messages based on a two stage filter. The first stage uses a straightforward lexical analyzer to detect and remove all messages that are lexically incorrect with reference to the grammar that is defined by the protocol standard. The second stage uses a machine learning approach based on a Support Vector Machine (SVM) to analyze the structure of the remaining syntactically correct messages in order to detect semantic anomalies which are deemed a strong indication of a possibly malicious message. The SVM "learns" the structure of the "good" and "bad" SIP messages through an initial training phase and the SVM thus configured correctly classifies messages produced by a synthetic generator and also "real" SIP messages that have been collected from the communication network at our institution. The preliminary results of such classification look very promising and are presented in the final section of this paper.**

I. Introduction

Thanks to its flexibility and descriptive power, the Session Initiation Protocol (SIP) is becoming the support not only for Voice over IP (VoIP) and Internet telephony, but also for many other so called session-based applications, such as most multimedia streaming, chats, and many others integrating web-services with telephone and voice. While VoIP and Internet telephony remain the core SIP applications, all of these mentioned above have added value and thus have enjoyed rapid acceptance and use, which - in turn - has attracted the attention of the protocol's resilience and security in the presence of incorrect or malformed messages. This is of particular importance for the session-based applications since they appear to be much more sensitive than web services or e-mails to intrusion and mis-functions.

SIP analysis and anomaly detection has thus become an active area of research, as discussed below in the Related Work part (I-A). SIP is a plain, text-based protocol defined by an extensible formal grammar (see RFC 3261 [2] and related documents, *i.e.*, all the RFCs that are cited by and cite it) for the definition and scope of SIP). While SIP might be plain, it is not a simple protocol. Its extensibility, the fact that it must maintain an often complex status of the session, the structure of proxy agents and servers that define the global organization of services like telephony, conferencing, and so forth, make it very sensitive not only to malicious attacks, but also to errors, malformed messages, and incorrect interpretation of the standard. All these factors give raise to an enormous number of possible states which makes conformance testing not feasible (as for most Internet protocols).

The first step in any analysis and anomaly detection process is the control of single messages: are they "good" or "bad"? and most of all ... how do we define a "good" message? Fig. 1 shows a simple classification tree that introduces the terminology used in this paper. A *good* message is simply a valid SIP message that can be correctly interpreted by its recipient. This means the message is syntactically correct, semantically meaningful, and comes at the right time to trigger a correct and useful application decision. From a theoretical point of view, *bad* messages could be identified as the complement of the set of good messages, but this will not help much in the classification process, especially when considering that in realistic cases one needs a fast classification that can be performed in real time over a stream of SIP messages.

In this paper, the set of *bad* messages comprises *malformed*, *crooked*, and *malicious* messages. *Malformed* messages are those that simply are syntactically wrong. *Crooked* messages are those that, while syntactically correct, have no meaning, cannot be interpreted, are ambiguous, or lead to a deadlock, etc. Finally, *malicious* messages, are those that are correct and meaningful, but will harm the system: normally these are forged on purpose, but they can also be the outcome of malfunctioning devices, badly implemented instances of the protocol or, more likely, of its extensions.

Anomaly detection of a stateful protocol would also require to correlate different messages, e.g., to identify unsolicited calls in telephony, or to identify messages attempting to impersonate another user during a call. However, the first step in traffic analysis is always a message filtering subsystem able to separate bad from good messages. The contribution

*A short version of this Technical Report appears in [1]

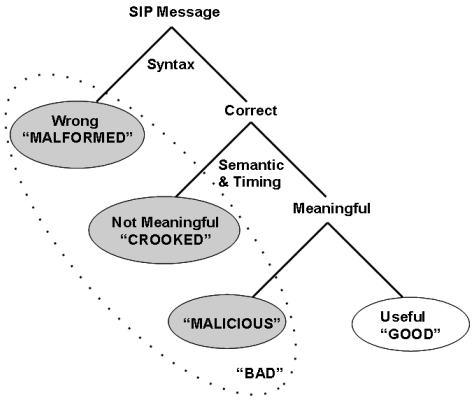

Fig. 1. Simple binary classification of SIP messages highlighting the terminology used in the paper.

of this paper is in this direction and to this end a two stage filtering methodology is proposed. The first stage is a lexical analyzer derived directly from the SIP formal grammar specification, while the second stage is based on machine learning techniques, specifically on Support Vector Machines (SVM) [3].

The first stage identifies all mistakes and malformations that violate the grammar; this is a deterministic and efficient process that is quite straightforward. For the second stage, an SVM has been previously "trained" to classify SIP messages by statistically learning from examples of good an bad messages. Once configured, the SVM can operate at run time by examining the SIP messages that have passed the first stage to identify the majority of crooked and malicious messages.

In addition to the initial description of this simple two-stage architecture, the paper provides insights on the identification and selection of the kernels of the SVM for its efficient implementation, as well as on the its training to achieve a high accuracy in the classification process.

After the initial presentation of related work, the paper is organized as follows. Sect. II discusses the proposed filtering methodology, explaining the reasons of our choices; Sect. III presents the performance of the system and Sect. IV ends the paper.

A. Related Work

Works on traffic analysis and intrusion detection in general are too numerous for a comprehensive presentation here, so this section focuses on papers that consider those issues in the context of SIP, highlighting the different approaches adopted by various researchers.

Niccolini *et al.* in [4] extended the basic functionalities of the very popular "signature-based" network intrusion detection system Snort[1] to SIP protocol, showing that it can perform well also for this protocol.

Similarly, Geneiatakis *et al.* [5] and Li *et al.* [6] defined a specific "signature" considering the syntax of well-formed SIP message defined in the IETF standard of SIP protocol RFC 3261 [2]. Any message that does not comply with that "signature" is considered as malformed and discarded. Seo *et al.* [7] proposed an intrusion detection system for SIP-based VoIP system utilizing rule matching algorithm and state transition models. Authors of [5] and [7] point out that the rules of SIP messages defined in RFC 3261 [2] cannot cover all kinds of malformed SIP messages (for example, RFC 3261 [2] does not define any range for scalar fields) and hence [7] has proposed an extension to RFC 3261 by introducing additional rules to make it (more) secure. Generally, such signature-based intrusion/anomaly detection systems work well when no entirely new, uncataloged attacks occur and the "attack signatures" database is not huge. However, handling of intrusion/anomalous detection problem in the context of SIP with a table-drive approach (rule/signature database) is destined to run up against the combinatorial explosion, as there are endless ways of forming a malformed or malicious message. Again, there are multiple ways of structuring a correct SIP message. Sengar *et al.* [8] proposed an intrusion detection system for SIP protocol using a finite state machine, which try to identify violations of the protocol behavior. Menna *et al.* in [9] concentrate specifically on unsolicited calls trying to isolate users that fall outside expected behaviors.

Other researchers have proposed machine learning techniques for SIP messages analysis, like in [10], [11], [12], [13] where the anomalous content is identified by parsing SIP messages. A self-learning anomaly detection system is proposed by Rieck *et al.* [10] which emphasizes the detection of unknown and novel attacks. In a manner similar to our own proposal, incoming SIP messages are mapped into feature spaces and the anomaly detection model is trained using normal/well-formed SIP traffic. Anomalous messages are identified as those whose Euclidean distance from those in the model of normality is higher than a given threshold. [12], however, reports that classifiers based on Euclidean distance computation do not produce adequate results for well-crafted malicious messages that differ very slightly from normal messages and hence its authors suggest a classifier based on Levenshtein distance [14] to measure the similarity between good and bad SIP messages.

II. Filtering Methodology

As mentioned in the introduction, "bad" SIP messages are all those messages that do not belong to a valid, correct and legitimate SIP session. They can be generated when SIP protocol implementations or applications do not fully comply with the standards or they contain errors in the implementation code. In addition, attackers can manipulate SIP messages to take advantage of existing security problems in the target system, or to exploit SIP weak points.

In this paper we have considered various kinds of *bad* messages, as shown in Fig. 1, comprising *malformed*, *crooked*, and *malicious* messages. While malformed messages can be detected by a straightforward check on their syntax, crooked and malicious messages need different techniques to be properly classified.

[1] Snort home page, http://www.snort.org/

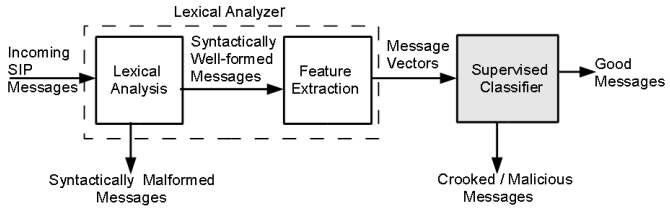

Fig. 2. Architecture of LEX_SIP.

Based on the considerations above, we have developed a 2-stage SIP message filter called LEX_SIP to detect anomalous messages within a stream of SIP messages. The main goal here was to obtain an efficient classifier that could operate in real time, without disrupting underlying services such as VoIP. LEX_SIP can be run on the same machine of the agent, or, if performance requires it, as a kind of firewall in front of it.

Fig. 2 shows the architecture of LEX_SIP where the first block is the lexical/syntactic[2] analyzer and the second block is the structure and content analyzer.

The work-flow of the two filtering stages are described in following sub-sections.

A. First Stage: Lexical Analyzer

The first stage filtering is performed by the lexical analyzer which investigates each SIP message, extracts from it a sequence of tokens/words that are then parsed to determine if they are part of the language generated by the formal grammar which specifies the SIP protocol [2]. The formal syntax of SIP protocol is defined by a context-free grammar specified in Augmented Backus-Naur Form (ABNF)[3], a metalanguage based on Backus-Naur Form (BNF) which is one of the main notation techniques for context-free grammars. Given the formal grammar definition of the SIP protocol (and thus, the syntax of all SIP messages), the implementation of the lexical analyzer is a well defined task that can be realized by using any of the standard tools, for example *lex* available under *Unix*, used to parse a language. Unlike in parsing of programming languages, here the processing can be stopped immediately upon detection of the first syntax error rather than continue parsing to the end of that message since there is no need to extract a list of errors present in the whole message.

Before passing them to the next level filtering, the lexical analyzer further processes the syntactically well-formed messages. This process can be thought of as a specialized kind of tokenization, where the token are significant features extracted from the SIP messages which are now represented as vectors in features space.

B. Second Stage: Structure and Content Analyzer

SIP messages that have passed the lexical analyzer filter may still be "bad". For example, Fig. 3 shows a SIP message with an unknown request method, and an unknown URI scheme in the Request-URI. The message is syntactically valid but a server receiving this message will fail to process this type of message and may be perform time-consuming analyses to determine the request message type and the information necessary to route this kind of request.

```
NEWMETHOD unknown:unknown uri SIP/2.
To: sip:j.user@example.com
From: sip:caller@example.net;tag=34525
Max-Forwards: 6
Call-ID: mismatch02.dj0234sxdfl3
CSeq: 8 NEWMETHOD
Contact: <sip:caller@host.example.net>
Via: SIP/2.0/UDP host.example.net;branch=z9hG4bKkdjuw
Content-Type: application/sdp
l: 138

v=0
o=mhandley 29739 7272939 IN IP4 192.0.2.1
c=IN IP4 192.0.2.1
m=audio 49217 RTP/AVP 0 12
m=video 3227 RTP/AVP 31
a=rtpmap:31 LPC
```

Fig. 3. Syntactically well-formed *malicious* SIP message.

To cull out these *bad* messages from those syntactically well-formed SIP messages that have been accepted by the lexical analyzer, the second stage filter analyzes the structure and the contents of the messages to flag those that are either *crooked* or *malicious*. Any hope of tackling this problem with an algorithmic or table-drive approach is destined to run up against the combinatorial explosion of the cases that need to be considered, as there are endless ways of forming a crooked or malicious message. ([2] there are 14 request message types, 6 response message kinds, there are neither maximum lengths of the headers, nor upper limits for some values, some fields are mandatory while other are optional and —finally— there are multiple ways of structuring a correct SIP message. This results in the practical impossibility of systematically examining all possible cases of corruption — casual or voluntary— of a message).

An attractive alternative is that of correctly classifying those messages that have a structure and content that *appear frequently* in a stream of messages. This can be done by applying one of the "supervised machine learning" techniques that recently have found many useful applications in numerous problem areas. The basic idea is to provide a sufficiently rich set of examples with their correct classification and "train" a machine automaton to carry out such classification, even on messages that were never seen before. An added bonus of this approach is in its flexibility to fit new kinds of bad messages that might become common at a later time, perhaps as a new breed of malicious messages is introduced when some weakness of the protocol is uncovered. The adaptation to the changed operating scenario can be obtained simply by retraining the machine automaton while including the new messages, duly identified as *bad*.

Various approaches to supervised machine learning have been proposed. Recently the so called Artificial Neural Networks (ANN) and the Support Vector Machines (SVM) have

[2]We use the terms lexical and syntactical as synonyms in this paper although in other contexts they may acquire slightly different meanings.

[3]David H. Crocker and Paul Overell. Augmented BNF for Syntax Specifications: ABNF. Internet RFC 5234, January 2008.

received much attention since they have performed quite well in a variety of problem contexts. For our second stage filter of LEX_SIP we have selected to use an SVM for the reasons that will be discussed below.

Support Vector Machines have been introduced by Vapnik in [15] and have been successfully applied to many fields such as Bio-informatics, Natural Language Processing, Handwritten Character Recognition and many others. Unlike Artificial Neural Networks, SVMs do not have the problem of getting stuck in a local minimum while searching for an optimal configuration of its operating parameters (see for example [16], [3], [17]). In addition, SVMs are scalable since the computational complexity does not depend on the dimensionality of the input space. Finally, SVMs optimization by training (or re-training) is quite fast problem which can represent an important point if online-intrusion (re)configuration is of paramount importance.

The basic idea of SVM classification is to interpret the d-dimensional feature vectors derived from SIP messages as points in an d-dimensional space. Some of these points correspond to "good" messages (label them as -1) and the others correspond to "bad" messages (label them as +1). The classification problem can then be seen as finding an hyperplane that separates the space in two sub-spaces: one containing all the -1 points, the other all the +1 points. If the set of points is linearly separable into two classes, there are infinite planes that will work. Notice, however, that there is only one "best" hyperplane that maximizes the distance between it and the nearest data points of each class. Unfortunately, it is often the case that no such hyperplane exists (the set of points is not linearly separable) and hence some points would be misclassified, as they would lay on the "wrong" side of the best hyperplane. However, while not linearly separable, the points could be separable if some other, more complex, surface were used instead of the (simple) hyperplane. Informally, the SVM computation does this by projecting all points into a higher-dimensional space and in that space the complex separating surface becomes a hyperplane, thus linearly separating the set of projected points.

More precisely, given a set of n SIP messages, let $\vec{x_i}$ be the i-th message which is transferred into a d dimensional feature vector and let $y_i \in \{-1, +1\}$ be and indicator function where -1 indicates that the i-th message belongs to the class of *good* message class while $+1$ indicates that the i-th message is in the *bad* message class. The equation of the hyperplane separating the training set $\left\{(x_i, y_i)\,|x_i \in \Re^d, y_i \in \{-1, 1\}\right\}_n^{i=1}$ can be defined as:

$$\vec{w} \cdot \vec{x} + b = 0 \tag{1}$$

where $\vec{w}$ is the vector normal to the hyperplane and $\frac{b}{\|\vec{w}\|}$ is the perpendicular distance from the hyperplane to the origin.

For the linearly separable data, SVM finds the optimum separating hyperplane with the largest margin (Fig. 4) by solving the Quadratic Programming (QP) optimization problem

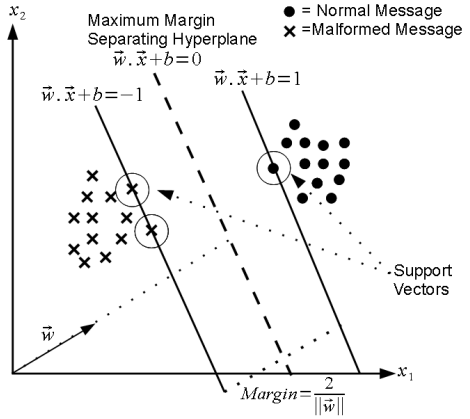

Fig. 4. Linear Support Vector Machine: optimum separation hyperplane.

described by eq. (2).

$$min\left\{\frac{\|\vec{w}\|^2}{2}\right\},\ \text{subject to}\ y_i\left(\vec{w}\cdot\vec{x_i}+b\right) \geq 1\ ,\ \forall_i \tag{2}$$

If the set of points is *not* linearly separable, then instead of trying to fit a non-linear model, the set of points can be mapped to a higher-dimensional space by a non-linear mapping function ϕ, $\vec{x} \rightarrow \phi(\vec{x})$ so that the points become linearly separable in this higher dimensional space.

The classification function in dual space becomes:

$$h\left(x\right) = sgn\left(\phi\left(\vec{w}\right)\cdot\phi\left(\vec{x}\right)+b\right) \tag{3}$$

$$= sgn\left(\sum_n^{i=1}\alpha_i y_i \phi\left(\vec{x_i}\right)\cdot\phi\left(\vec{x}\right)+b\right) \tag{4}$$

In the quadratic optimization problem for non-linearly separable data SVM, the training vectors appear only in the form of dot products, $\left(\phi\left(\vec{x_i}\right),\phi\left(\vec{x_j}\right)\right)$ which imply that computationally expensive dot product calculation would have to be needed. However, by using the so called "kernel functions" one can apply a "kernel trick" that avoids the expensive dot products. Kernel functions that have been favored in the recent literature include:

- *Linear kernels*: $k\left(\vec{x_i},\vec{x}\right) = \vec{x_i}\cdot\vec{x}$;
- *Polynomial kernels* of degree d: $k\left(\vec{x_i},\vec{x}\right) = \left(\vec{x_i}\cdot\vec{x}\right)^d$;
- *Radial Basis Functions (RBF) kernels*: $k\left(\vec{x_i},\vec{x}\right) = exp\left(-||\vec{x_i}-\vec{x}||^2/2\sigma^2\right)$.

Thus, the classification function of eq. (3) becomes:

$$h\left(x\right) = sgn\left(\sum_n^{i=1}\alpha_i y_i k\left(\vec{x_i}\cdot\vec{x}\right)+b\right) \tag{5}$$

Fig. 5 represents the polynomial projection of non-linear data into high-dimentional feature space where they are linearly separable.

For our classifier of well-formed SIP messages we have used LibSVM [18], a freely available library for Support Vector Machines. First, the SVM was trained with a set of 500 pre-classified examples of 40 features (details information about these features are described in Appendix) derived from a

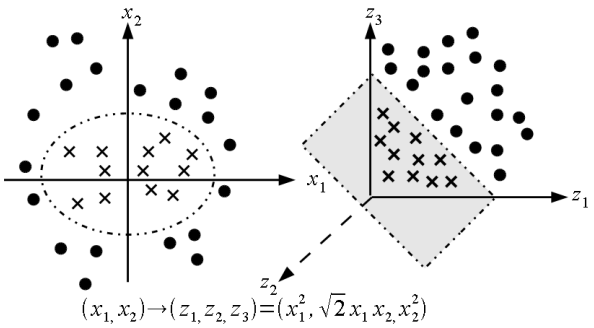

Fig. 5. Non-linear SVM: polynomial mapping

balanced mix of good and bad messages. This training phase determines a configuration of the SVM that will result in a good classification of subsequent messages contained in a test set. The test set contains unlabeled SIP message vectors that are passed from lexical analyzer. Our experimental results shows that SIP message vectors are not linearly separable, so we need to choose a kernel and relevant parameters in the hope that in higher dimension the data would become linearly separable. To this end, experiments were carried out with various kernels and parameters. It was very obvious that the linear kernel did not yield any interesting result and hence the experimentations focused on polynomial kernels of degree 2, 3, and 4 and on RBF kernels. The latter are reported in the literature as leading to good results and so are deemed preferable despite their higher computational cost. However, in the support documents available with LibSVM it is mentioned that in the case of data with a large number of features (as in our case), sometime a low degree polynomial kernel is preferable. Our experiments confirmed the validity of this suggestion —at least for our specific application. The detailed performance results of the various experiments are reported in the next section.

III. Results and Performance

The goal of LEX_SIP is to maximize detection accuracy of SIP malformed message, while reducing message processing time since the latter plays a vital role in applicability of an intrusion detection system in a real-time environment like VoIP.

A. Synthetic Traffic Generation

Performance evaluation of any classifier can be established by analyzing the results over a statistically relevant collection of data. For LEX_SIP this means that a large number of SIP traces would be needed. However, reliable real world VoIP traces are hard to get because of the user privacy agreement of VoIP providers. Furthermore, VoIP traces of malicious messages during an attack are quite infrequent. Considering this situation, we have developed "SIP-Msg-Gen"[4], a synthetic SIP message generator, capable of generating both "good" or "bad" SIP messages. "SIP-Msg-Gen" is available under GPL license terms. "Good" messages are generated by following the basic parsing constructs of SIP protocols defined in RFC 3261 [2], while generation of "bad" messages is influenced by SIP torture test messages defined in RFC 4475 [19] and PROTOS test suite[5]. Torture test messages contain all kinds of variations on the basic structure of a "normal" message, as, for example, numerous line foldings and white spaces all over the message, escaped characters within quotes, a mix of short and long form for the same header, unknown header fields, unusual header ordering, unknown parameters of a known header, *etc.*

TABLE I
Description of Dataset containing SIP messages.

Scenario	Number of Msg
"Good" Messages	984,000
Syntax Error in "First-Line" of a message	94,815
Syntax Error in header fields of a message	181,944
Null entry for mandatory header fields	94,723
Multiple "First-Line" in a message	47,672
Unknown/Invalid Protocol version	23,836
Missing mandatory header fields a message	95,245
Duplicate entry for unique header fields	95,560
Presence of garbage string after message body	47,554
Hierarchical disorder of message structure	23,935
Overlarge value of scalar field	46,789
Missing/multiple empty line in a message	11,918
Msg Length larger than "Content-Length" value	47,553
Unknown scheme for "Content-Type", "Authentication" and "Accept" header	59,788
Unknown Method Name	23,935
Unknown scheme for "Request-URI"	46,788
Unknown scheme for message body	36,073
Unknown Response status	36,072

B. Performance Evaluation

The performance of LEX_SIP is measured through *efficiency* which is defined by the message classification capability, and *effectiveness* which is the time/effort needed for classification.

The experimental dataset consists of two million SIP messages generated by "SIP-Msg-Gen" where 984,000 are "good" messages and 1,016,000 are "bad" messages. Included in the "good" messages there are 123,000 valid "torture" messages to measure the efficiency of lexical analyzer. Among the "bad" messages, 371,482 messages contain syntax error and 644,518 messages are syntactically well-formed but are not meaningful, those that we called "crooked" messages. A detailed description of the composition of the data set is found in Table I.

The lexical analyzer of LEX_SIP analyzed all messages in the data set and has successfully identified all the 371,482 malformed messages. The remaining 1,628,518 syntactically correct messages were processed to extract from them 40 significant features which were passed on to the second level filter to be classified by the SVM-based structure and content analyzer.

In our work we have used C-SVC, which is a modified maximum margin idea of SVM proposed by Cortes and

[4]http://disi.unitn.it/∼ferdous/SIP-Msg-Gen.html

[5]PROTOS Project page, https://www.ee.oulu.fi/research/ouspg/PROTOS_Test-Suite_c07-sip

Vapnik [20] that allows the decision margin to separate the data set with a minimal number of errors (outliers or noisy examples that are inside or on the wrong side of the margin). This method introduces *slack variables* ξ_i which measure the degree of misclassification of the data x_i. The corresponding formulation of the SVM optimization problem with slack variables becomes:

$$min \left\{ \frac{\|\vec{w}\|^2}{2} + C \sum \xi_i \right\} \quad (6)$$

$$\text{subject to } y_i \left(\vec{w} \cdot \vec{x_i} + b\right) \geq 1 - \xi_i \ , \forall_i \quad (7)$$

By definition, $\xi > 0$; $0 < \xi \leq 1$ indicates a data point that lies somewhere between the margin and the correct side of hyperplane, while $\xi > 1$ denotes a misclassified data point. Here C is a regularization parameter defined as "soft margin constant" that controls the trade-off between maximizing the margin and minimizing the training error and is usually thought of as a way to control overfitting.

Although SVM is a widely used classifier, effective use of SVM requires an understanding of its parameters and their influence over classification accuracy. In particular, the selection of kernel function and tuning of the kernel parameters (*e.g.*, degree of polynomial kernel, σ for RBF kernel) and soft margin constant C, are very important decisions for classification with SVM. Unfortunately, there is no specific algorithm to select a kernel function, since, much like in other machine learning techniques, this is also data dependent. The user guide of LibSVM [21] suggests to use *cross-validation* to obtain an estimate the performance of a predictive model with various parameters and then select a specific set of parameters based on these estimates. In search of suitable parameter setting, we have performed a k-fold cross-validation where the idea is to divide the training set into k subsets of equal size (we have used k=5). Each subset is then tested (validated) using the classifier trained on the remaining $k-1$ subsets using a specific set of parameters. The k results are then combined (*e.g.* averaged) to obtain a single index. The entire process is repeated for each set of parameters and the set of parameters with the best cross-validation index is finally selected. We have performed this test for three kernel functions (e.g., linear, radial basis function, and polynomial) and various values of the relevant parameters, as shown in Table:II.

For our analysis we have started with a simple linear kernel, since, having only the soft-margin constant C, it is easier to tune. The linear kernel with the best value of C provided us with a baseline against which we could assess the performance of the other kernels and parameters.

Although in general RBF is often recommended despite its higher computational cost, for this particular dataset the RBF kernel was not the most suitable. Table II shows that the cross-validation accuracy using RBF kernel (with $0 < \sigma \leq 1$) is very low for small value of C (soft margin constant), though the accuracy increases as the value of C increases. In general, small values of C tend to emphasize the margin while ignoring the outliers in the training data. Conversely, large values of C increase the possibility to overfit the training data. Hence, a small value of soft margin constant C is usually preferable.

TABLE II
ACCURACY OF DIFFERENT KERNEL FUNCTIONS ON DATASET OF SIP MESSAGES.

Soft Margin Constant,C	Kernel Function	Accuracy
2^{-1}	Polynomial (degree 2)	99.62%
	Polynomial (degree 3)	99.56%
	Polynomial (degree 4)	99.27%
	Radial basis function (σ=0.4)	46.40%
	Linear	47.64%
2^{0}	Polynomial (degree 2)	99.62%
	Polynomial (degree 3)	99.56%
	Polynomial (degree 4)	99.27%
	Radial basis function (σ=0.4)	52.74%
	Linear	47.64%
2^{2}	Polynomial (degree 2)	99.62%
	Polynomial (degree 3)	99.56%
	Polynomial (degree 4)	99.27%
	Radial basis function (σ=0.4)	88.21%
	Linear kernel	47.64%
2^{3}	Polynomial (degree 2)	99.62%
	Polynomial (degree 3)	99.56%
	Polynomial (degree 4)	99.27%
	Radial basis function (σ=0.4)	91.03%
	Linear kernel	47.64%

TABLE III
RESULT OF SVM CLASSIFIER FOR 1628518 SYNTACTICALLY WELL-FORMED MESSAGES.

True positive	A "bad" messages is correctly identifies as "bad"	643,008
False positive	A "good" message is incorrectly identified as "bad"	119
True negative	A "good" message is correctly identified as "good"	983,881
False negative	A "bad" message is incorrectly identified as "good"	1,510

Finally, we turned our attention to polynomial kernels of degree d. To estimate the suitable degree d of polynomial kernel ($k\left(\vec{x_i}, \vec{x}\right) = \left(\vec{x_i} \cdot \vec{x}\right)^d$), we performed a "grid-search" on two parameters (soft margin constant C, and polynomial degree d) using cross-validation. The performance for various pairs of (C, d) values ($C = 2^{-1}, 2^0, 2^2, 2^3$... and $d = 2, 3, 4, ...$) were evaluated through cross-validation. The best combination was obtained for the pair, indicating that a second degree polynomial flexible enough to discriminate between the two classes with a reasonably small soft margin.

Results produced by SVM classifier are found in table III

We have represented the efficiency of classifier in table IV through a few metrics which are widely used in the area of Pattern Recognition and Information Retrieval to measure the performance of classification.

The average time for LEX_SIP to classify a SIP message is 0.45 millisec/msg. This time is the aggregation of processing time of individual filtering stage. It is found that about 0.35 millisec/msg time is requred for lexical analyzer to perform syntax checking of a SIP message and the remaining time is required for classification with SVM. All experiments are done in a machine of Intel Core i7 CPU, 2.0 GHz Quad-core and

TABLE IV
EXPERIMENTAL RESULTS:EFFICIENCY OF CLASSIFIER

Metrics	Metric Description	Percentage
Recall/ Sensitivity	Fraction of malformed messages that are identified	99.76%
Accuracy	Proportion of true results (both true positives and true negatives)	99.89%

TABLE V
RESULT OF LEX_SIP FOR REAL SIP TRACES.

Description	Number of Messages
Total Message	13,836
True positive	10
False positive	2
True negative	13824
False negative	0
Metric	**Percentage**
Recall/ Sensitivity	100%
Accuracy	99.9%

8 GB RAM memory.

C. Preliminary Test with Real Traffic

The final goal of a classifier is working with real traffic — possibly on-line. To this end, we established an agreement with our institution that allowed us to start collecting SIP traces by mirroring the port in front of the SIP Proxy server. Due to privacy issues, so far we could only collect very few messages (less than 20,000), but anonymized data collection is on its way for future use. The lexical analyzer successfully passed all the messages as syntactically well-formed messages towards the classifier for next level filtering. Performance of LEX_SIP on real SIP traces is summarized in Table V.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, a two-stage filtering approach is proposed for SIP anomalous message detection. The first stage is a straightforward lexical analyzer, whose goal, besides controlling the validity of the message syntax, is the extraction of relevant features to be used by the second stage, a supervised classifier.

As a supervised classifier, we have selected an SVM. The advantages of SVM is that very few parameters are required for tuning the learning machine and a small sample set can build the model to classify huge unlabeled dataset. But the efficiency of SVM depends on selection of kernel functions and its parameters. Though [21] suggests to use cross-validation procedure for model selection as it prevents the overfitting problem, for large training set it becomes time-consuming to use cross-validation. Similarly, due the exhaustive nature of search, selection of kernel parameters becomes tedious and cumbersome using "grid-search" techniques, specially for large dataset. Moreover, the use of cross-validation and exhaustive techniques does not fit well with the goal of on-line use of the classifier.

Instead of performing a complete "grid-search", we have reduced the searching space by identifying a set of possible value for parameter pair (C, d) (soft margin constant and degree of polynomial), then the parameter pair with best accuracy is selected. In this case, the "grid-search" can be accelerated by parallel search as each pair (C, d) is independent.

The selection of features to use for the classification,is fundamental for efficient and precise classification. The results obtained both with artificial traces, and with an initial set of real-traffic traces collected in the production network of our University, are very promising, leading to extremely accurate classification.

Future work includes extended analysis of real traces, as well as deeper insight on feature extraction. Moreover, the possibility of defining a methodology for on-line continuous training of the SVM will be explored in the attempt of realizing an autonomic, self-training system for SIP message classification.

REFERENCES

[1] R. Ferdous, R. Lo Cigno, and A. Zorat, "Classification of sip messages by a syntax filter and svms," in *IEEE Global Telecommunications Conference (GLOBECOM 2012)*, Anaheim, CA, USA, Dec. 3-7, 2012.

[2] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "Sip: Session initiation protocol," RFC 3261, jun 2002.

[3] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 121–167, 1998.

[4] S. Niccolini, R. Garroppo, S. Giordano, G. Risi, and S. Ventura, "Sip intrusion detection and prevention: recommendations and prototype implementation," in *1st IEEE Workshop on VoIP Management and Security*, april 2006, pp. 47 – 52.

[5] D. Geneiatakis, G. Kambourakis, C. Lambrinoudakis, T. Dagiuklas, and S. Gritzalis, "A framework for protecting a sip-based infrastructure against malformed message attacks," *Comput. Netw.*, vol. 51, no. 10, pp. 2580–2593, jul 2007.

[6] H. Li, H. Lin, X. Yang, and F. Liu, "A rules-based intrusion detection and prevention framework against sip malformed messages attacks," in *3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*, oct. 2010, pp. 700 –705.

[7] D. Seo, H. Lee, and E. Nuwere, "Detecting more sip attacks on voip services by combining rule matching and state transition models," in *Proceedings of The International Federation for Information Processing (IFIP)*, vol. 278. Springer Boston, 2008, pp. 397–411.

[8] H. Sengar, D. Wijesekera, H. Wang, and S. Jajodia, "Voip intrusion detection through interacting protocol state machines," in *International Conference on Dependable Systems and Networks, (DSN 2006)*, june 2006, pp. 393 –402.

[9] F. Menna, R. Lo Cigno, S. Niccolini, and S. Tartarelli, "Simulation of spit filtering: Quantitative evaluation of parameter tuning," in *IEEE International Conference on Communications (ICC '09)*, June 2009.

[10] K. Rieck, S. Wahl, P. Laskov, P. Domschitz, and K.-R. Müller, "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*. Springer Berlin / Heidelberg, 2008, vol. 5310, pp. 90–106.

[11] M. Rafique, Z. Khan, M. Khan, and K. Alghatbar, "Securing ip-multimedia subsystem (ims) against anomalous message exploits by using machine learning algorithms," in *Eighth International Conference on Information Technology: New Generations (ITNG)*, april 2011, pp. 559 –563.

[12] N. Hentehzadeh, A. Mehta, V. Gurbani, L. Gupta, T. K. Ho, and G. Wilathgamuwa, "Statistical analysis of self-similar session initiation protocol (sip) messages for anomaly detection," in *4th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, feb. 2011, pp. 1 –5.

[13] M. Nassar, R. State, and O. Festor, "Monitoring sip traffic using support vector machines," in *Recent Advances in Intrusion Detection (RAID '08)*. Springer Berlin / Heidelberg, 2008, vol. 5230, pp. 311–330.

[14] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[15] V. N. Vapnik, *The nature of statistical learning theory*. NY, USA: Springer-Verlag New York, Inc., 1995.
[16] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning (ICML '06)*, New York, USA, 2006, pp. 161–168.
[17] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge, UK: Cambridge University Press, 2004.
[18] C. Chang and C. Lin, "Libsvm: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 1–27, 2011, software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.
[19] R. Sparks, A. Hawrylyshen, A. Johnston, J. Rosenberg, and H. Schulzrinne, "Session initiation protocol (sip) torture test messages," RFC 4475, May 2006.
[20] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.
[21] C. Hsu, C. Chang, and C. Lin, "A practical guide to support vector classification," Department of Computer Science, National Taiwan University, Tech. Rep., 2003.

Table 1: List of Features for classification of SIP messages using SVMs

ID	Feature	Description	Value Range
1	Message Type	This feature indicates the type (Request/Response) of a SIP message	0= Unknown Method, 1-14 =Request Method, 15=Response Message
2	Request Line Status	This feature indicates the status and frequency (absence/multiple) of Request-Line in a Request Message.It is required to identify error in Request-Line	-1= In wrong position, 0 = No Request-Line, 1 = In perfect postion , >1 = Multiple Request-Line
3	Protocol Version	This feature contains information about SIP protocol version. It is required to detect error such as unknown or higher protocol version	SIP protocol version
4	Empty Line Status	This feature indicates the status and occurance of empty line in a message. It is required to hierarchical dirorder in a message due to multiple/incorrect presence of empty line in a messge	-1=in wrong position, 0= no empty line, >1= multiple empty line
5	Message order	This feature contains information about the hierarchical order of a SIP message. It holds information of any kind of disorder in a message	1=message in order, -1= message is not in order
6	Presence of garbage string	This features indicates the presence of garbage string in a message	-1=presence of garbage string, 1= no garbage string
7	Response Line Status	This feature indicates the status (absence/multiple) of Response-Line in a Response Message	Frequency of Request-line in a message
8	Scalar value of “CSeq” header	This feature contains the value of the scalar field in ’CSeq’ header	Value of scalar field in “CSeq” header
9	Scalar value of “Max-Forwards” header	This feature contains the value of the scalar field in “Max-Forwards” header	Value of scalar field in ’Max-Forwards’ header
10	Scalar value of “Content-Length” header	This feature contains the value of the scalar field in “Content-Length” header	Value of scalar field in ’Content-Length’ header
11	Missing Mandatory Header	This feature contains the information whether any specific mandatory header field is missing in a Response message.	0 = All mandatory fields are present, -1 = missing mandatory header fields in Response message
12	Method Name	This feature indicates the method name of a Request message	0= Unknown Method Name, 1 -14 = Valid Request Method
13	Request-URI status	This feature contains the status of Request-URI. This featue is used to identify unknown Request-URI scheme in a Request message	-1 = Unknown Request-URI scheme, 1= Correct Request-URI scheme
14	IP address in “Via” header	This feature contains the IP address of “Via” header field of a Response message. This feature is required to identify Response message which intends to broadcast due to IP addreess (“255.255.255.255”) in ’Via’ header	1= header ok, -1 = error in IP address
15	Size of Response code	This feature indicates the length of a Response Message. It is used to detect very large Response message	Length of the Response Message
16	Mandatory header field “Call-ID” status	This feature indicates the occurance of header field “Call-ID” in a message	Frequency of header field ’Call-ID’ in a message. 0= Missing, 1= Single appearance, 2= Duplicate appearance... of “Call-ID” in a message

Continued on next page

Table 1 – continued from previous page

ID	Feature	Description	Value Range
17	Mandatory header field "CSeq" status	This feature indicates the occurance of header field "CSeq" in a message	Frequency of header field "CSeq" in a message. 0= Missing, 1= Single appearance and ok, >1= Multiple appearances
18	Unknown Header Field	This feature indicates the presence of unknown header field in a message.	1= no unknown header field, -1= unknown header field
19	Mandatory header field "Contact" status	This feature indicates the occurance of header field "Contact" in a message. It is used to detect error such as missing or multiple occurance of "Contact" header field in a message.	Frequency of header field "Contact" in a message. 0= Missing, 1= Single appearance, >1= Multiple appearances
20	Mandatory header field "From" status	This feature indicates the occurance of header field "From" in a message. It is used to detect error such as missing or multiple occurance of "From" header field in a message.	Frequency of header field "From" in a message. 0= Missing, 1= Single appearance, >1 = Multiple appearances
21	Mandatory header field "Max-Forwards" status	This feature indicates the occurance of header field "Max-Forwards" in a message. It is used to detect error such as missing or multiple occurance of "Max-Forwards" header field in a message.	Frequency of header field "Max-Forwards" in a message. 0= Missing, 1= Single appearance, >1= Multiple appearances
22	Mandatory header field "To" status	This feature indicates the occurance of header field "To" in a message. It is used to detect error such as missing or multiple occurance of "To" header field in a message	Frequency of header field "To" in a message. 0= Missing, 1= Single appearance, >1 = Multiple appearances
23	Frequency header field "Via" status	This feature indicates the occurance of header field "Via" in a message. It is used to detect error such as missing or multiple occurance of "Via" header field in a message	Frequency of header field "Via" in a message. 0= Missing, 1= Single appearance, >1 = Multiple appearances
24	"Authentication_Info" header field status	This feature contains information about the "Authentication_Info" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contains Eerror, 1= No error in header field.
25	"Accept" header field status	This feature contains information about the "Accept" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contains Eerror, 1= No error in header field.
26	"Content_Type" header field status	This feature contains information about the "Content_Type" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contains Eerror, 1= No error in header field.
27	"Organization" header field status	This feature contains information about the "Organization" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contains Eerror, 1= No error in header field.
28	"Date" header field status	This feature contains information about the "Date" header fields. This feature is used to detect any error in this field, such as unauthorized date format	-1= Contains Eerror, 1= No error in header field
29	"Expires" header field status	This feature contains information about the "Expires" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contains Eerror, 1= No error in header field

Continued on next page

Table 1 – continued from previous page

ID	Feature	Description	Value Range
30	"Allow" header field status	This feature contains information about the "Allow" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contains Eerror, 1= No error in header field
31	"Authorization" header field status	This feature contains information about the "Authorization" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contains Eerror, 1= No error in header field
32	"Proxy_Authenticate" header field status	This feature contains information about the "Proxy_Authenticate" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contins Eerror, 1= No error in header field
33	"Timestamp" header field status	This feature contains information about the "Timestamp" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contains Eerror, 1= No error in header field
34	"Subject" header field status	This feature contains information about the "Subject" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contains Eerror, 1= No error in header field
35	"Alert_Info" header status	This feature contains information about the "Alert_Info" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contains Eerror, 1= No error in header field
36	"Accept_Language" header field status	This feature contains information about the "Accept_Language" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contains Eerror, 1= No error in header field
37	"Content_Language" header field status	Contains information about the "Content_Language" header fields. This feature is used to detect any error in this field, such as unauthorized scheme	-1= Contains Eerror, 1= No error in header field
38	Combination of *Request-line* & *Response-Line*	This feature contains information about the presence of both *Request-line* and *Response-line* in the same message	-1= Contains Eerror, 1= No error
39	Message Body status	This feature contains information about the scheme of message body. This feature is used to identify messages with unknown message scheme	-1= Unknown scheme of Message body , 1= SDP/xml scheme for message body
40	Message Body Size	Contains information about the length of message body. This feature is required to identify messages where message length indicated by "Content_Length" header is larger than the received message body	Length of received message body