A Method for Disguising Malformed SIP Messages to Evade SIP IDS

Yulong Wang, Lei Wang
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China
Email: wyl@bupt.edu.cn, xinghunwl@gmail.com

Abstract— **Malformed SIP attacks are threatening the security of VoIP system, such as IP Multimedia Subsystem, which uses SIP (Session Initiation Protocol) as its core protocol. Though IDSs (Intrusion Detection System) supporting malformed SIP detection had been produced, it was not clear to what extent they can detect disguised malformed SIP messages. This paper analyzes the condition of SIP IDS evasion and proposes a method for disguising malformed SIP messages. Based on the disguising method, a testing system is built for evaluation the capability of SIP IDS on evasion defending. The result of the experiments show that the proposed method can improve the evasion rate of malformed SIP messages considerably, which means the defending capability of SIP IDSs should be improved to prevent them from evasion.**

Index Terms— **Malformed SIP, Intrusion Detection System, IDS Evasion**

I. Introduction

SIP(Session Initiation Protocol) is an application-layer signaling protocol for creating, modifying, and terminating multimedia sessions between one or more participants [1]. As the core signaling protocol of IP multimedia network, especially VoIP system such IMS (IP Multimedia Subsystem), its security is becoming a focus of attention.

Since SIP is a text-based protocol, it's very easy to construct malformed SIP messages, which are threatening the security of IP Multimedia Network [2]. Nowadays a variety of attacking counter measures are being researched [3]–[5]. Especially, some IDSs(Intrusion Detection System) are designed for defending IP multimedia network against malformed SIP attacks [6]–[10].

However, these IDSs can be evaded by some SIP messages. According to our investigation, the reasons are:

1) Some IDSs support old version SIP protocol, while SIP servers protected by them do not support that version. So the differences between the SIP protocol versions may result in malicious SIP messages reaching the servers without being filtered out by the IDSs.
2) The SIP detection modules in different SIP IDSs utilize different methods to parse and detect malicious SIP messages. Some of them may have flaws. For example, some implementations split a SIP message using LF, which would cause servers splitting SIP messages by CRLF to obtain an unexpected set of headers.
3) Some SIP detection modules in SIP IDSs ignore some details of SIP protocols. For example, a header field may contain several lines and the header may have a short name besides its full name.

To evaluate the defending capability of the SIP IDSs, we propose a disguising method for malformed SIP messages. Our contributions can be summarized as follows:

1) For the first time, we proposed a SIP disguising method for IDS evasion.
2) We designed a testing system based on the method, which is able to evaluate the defending capability of SIP IDSs on evasion.
3) We carried out experiments using the testing system to show the severe situation the existing SIP IDSs stays.

The rest of the paper is organized as follows. Section II shows the motivating example. Section III describes related works and its relationship to our work. Section IV analyzes current SIP attack detection methods, proposes the SIP disguising method for evading the detection and presents the design of a testing system based on the proposed method. Section V shows the experiments for verifying the effectiveness of the proposed method and showing the defending capability of the SIP IDSs. Finally, Section VI concludes our work.

II. Motivating Example

We implemented a testing system which can be used to generate malformed SIP messages and evaluate the security of an IMS network. When performing a test, we observed that if we changed the sending messages in some way, the messages, which would originally be blocked by IDS, could pass through the IDS of the IMS without triggering an alert (Figure 1). We are curious about the reason and want to develop a systematic method for generating such messages which would be helpful for testing the defending capability of SIP IDS.

Manuscript received April 9, 2013; revised May 16, 2013; accepted May 18, 2013. © 2005 IEEE.
This work was supported in part by the Youth Scientific Research and Innovation Plan of Beijing University of Posts and Telecommunications(2013RC1101), the Innovative Research Groups of the National Natural Science Foundation of China (61121061), the Disciplinary Joint Construction Project of the Beijing Municipal Commission of Education, and the Important National Science & Technology Specific Projects:Next-Generation Broadband Wireless Mobile Communications Network (2010ZX03004-001-01).

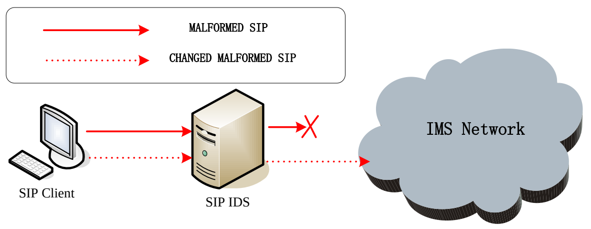

Figure 1. Motivating Example

III. Background and Related Work

IMS, the VoIP in the telecommunication field, is regarded as the core technologies of the communication network after 3G and it's also a network architecture to achieve the integration of fixed, mobile and Internet network [11]. Since it and other VoIP system heavily depends on SIP protocol, it's also very important to defend them against malformed SIP messages.

University of Oulu has developed a test suit, which is actually a malformed SIP message generator [12]. This test suit defines twenty malformation types for evaluating the effectiveness of IDSs.

Many IDSs have been designed adopting a variety of detection methods. S. Niccolini et al. proposed an intrusion detection and prevention architecture using snort [13], a very popular open source IDS [6]. It adopts a rule-based method and works very well when dealing with the SIP messages whose malformation types is known. The rule-based method has two stages for detecting malformed SIP messages:

1) Syntax Checking Stage: to check whether the received SIP message can be parsed correctly, usually by feeding it to a SIP stack.
2) Rule Checking Stage: to check whether the received SIP message conforms to the rules indicating malformation(e.g. the rule checking whether some mandatory header fields exist).

The main drawback of this method is that its effectiveness depends on SIP protocol stacks, which can be evaded due to implementation differences of the stacks. Moreover, it is unable to detect unknown type of malformation in SIP messages.

Dimitris Geneiatakis et al. adopted another method to detect malformed SIP messages, which is based on Regex Match [14] [7] [8]. The Regex-Match-based method provides some regular expressions for checking whether some parts of a received SIP message is benign. It checks the syntax of the received SIP message before parsing it. Usually the checking of this method is stricter than that of the rule-based method because it may check the charset of each field while SIP stacks utilized by rule-based method just partition a message to a set of fields without checking the charset. This method is able to handle unknown type of SIP malformation. The main weakness of this method is that the efficient of the detection would decrease when a more accurate regular expression is set. So the major challenge of this method is how to construct appropriate regular expressions. A straightforward way would be using the BNF(Backus-Naur Form) of header fields defined in the RFC 3261(SIP: Session Initiation Protocol) [15]. However, BNF is not strict enough. Fox example, BNF of SIP defines that a port is composed of several digit, but we know that the value of a port should be in the range from 0 to 65535.

Sohail Aziz et al. and Konrad Rieck, et al. all proposed a self learning model for detecting SIP malformed message attacks [9] [10]. However, the self-learning-based methods are not practical enough due to high rate of false positive and false negative. And they are unable to provide the accurate reasons why the found malformation is indeed a malformation.

According to the three reasons described in section I, these IDSs can be evaded using evasion technologies. Evasion technologies mainly utilize parsing flaws of protocol stacks to disguise the attacking messages so that the feature of the malicious content of the messages(e.g. malformation) would not be identified by IDSs. Evasion technology is usually applied to two protocol layers. The network layer evasion technologies usually take advantages of the differences between IDSs and destination host on packets handling. Tsung-Huan Cheng et al. summarized five common evasion technologies including denial of service, packet splitting, duplicate insertion, payload mutation and shellcode mutation [16]. The application layer evasion technologies mostly applied to HTTP. For example, Daniel J. Roelker reviewed two types of evasion technologies for HTTP protocol: invalid protocol parsing and invalid protocol field decoding [17]. Invalid protocol parsing means that the results of parsing are not correct(e.g. HTTP URL is not correct). Invalid protocol field decoding aims to test an IDS's capability for dealing with various types of encoding and normalization that should be supported in a specific protocol field. Recently, advanced evasion technology is proposed [18], which is a mix of known evasion technology [19]. The rapid development of evasion technology require researchers to find more effective methods for testing the defending capability of the existing IDSs.

Though evasion technologies is developing quickly, to our best knowledge, no work has been done on the evasion technology for SIP protocol. According to our investigation, the existing evasion technologies cannot be applied to SIP directly. So, in this paper, we focus on the application layer to research evasion technology for SIP protocol.

IV. SIP IDS evasion

A. *SIP IDS flaws on evasion*

In order to analyze how to evade a SIP IDS, we divides the procedure of malformed SIP detection methods into three stages.

1) Field Partition Stage. In this stage a SIP message will be partitioned to many strings and each string stands for a header field.
2) Field Parsing Stage. Based on previous results, in this stage the SIP IDS will check whether the string

```
REGISTER sip:ims.com SIP/2.0\r\n
Call-ID: b12057c9d426f2e82ff13cbfab5d056e@192.168.1.203\r\n
CSeq: 1 REGISTER\r\n
From: ”user1” <sip:user1@ims.com>;tag=1000\r\n
To: ”user1” <sip:user1@ims.com>\r\n
Expires: 3600\r\n
Authorization: Digest username=”user1@ims.
com”,realm=”ims.com”
,nonce=””,response=””,uri=”sip:ims.com”\r\n
Supported: path\r\n
Contact: <sip:192.168.1.203:5064>\r\n
P-Preferred-Identity: ”user1” <sip:user1@ims.com>\r\n
P-Access-Network-Info: 3GPP-UTRAN-TDD; utran-cell-id-
3gpp=00000000\r\n
Privacy: none\r\n
User-Agent: Fraunhofer FOKUS/NGNI Java IMS UserEndpoint
FoJIE 0.1 (jdk1.3)
Allow: INVITE,ACK,CANCEL,BYE,MESSAGE,NOTIFY\r\n
Content-Length: 0\r\n\r\n
```

Figure 2. A Normal SIP Message

representing the header field conforms to the syntax defined by its BNF in the SIP RFC.

3) Field Verification Stage. This stage is same as the rule-checking stage of the rule-based method.

For example, the normal SIP message in figure 2 consists of a Request Line, a *From* header field, a *To* header field and some other header fields. Request Line contains a *Method* element, a *Request URI* element and a *SIP Version* element. These elements are separated by a space element. The augmented BNF grammar of Request Line is ”Request-Line = Method SP Request-URI SP SIP-Version CRLF”. A Request Line can be further partitioned into more elements.

To partition a SIP message, the following rules should be paid attention according to RFC 3261:

1) RFC 3261 allows the value of a header field to be put into more than one line and requires a new line to start with a space or a tab.
2) RFC 3161 requires that a header field must be ended with a CRLF.

As the previous version of RFC 3261, RFC 2543 [20] allows a field header to end with a CR, a LF or a CRLF. This may lead to different implementations of SIP stacks. Moreover, different operating systems may define different line feeds, which may also lead to different handling to line feeds. We take three SIP stacks as examples. oSIP [21] allows a CR, a LF or a CRLF to be a header field delimiter. SER [22] regards a LF or a CRLF as a header field delimiter. However, Open-SIP [23] only allows CRLF to be the delimiter. This difference may cause a SIP message be partitioned into different sets of strings when it is parsed by different SIP stacks. If the difference can result in a SIP IDS not parsing a SIP header field(i.e. the SIP stack in the SIP IDS doesn't recognize the SIP header field), the malformation contained in the field can be concealed and the SIP message can evade the SIP IDS.

Another difference of these three SIP stacks lies in how they deal with the more-than-one-line problem. oSIP will replace all tabs, CRs and LFs with spaces except for line feeds. This results in a changed message to be parsed, which may be a malformed SIP message before. For example, RFC 3261 requires that only one space should exist between the *method* and the *request-uri* in a request message. SER adopts two ways to deal with this problem. One way is inspecting the next character of a CRLF or a LF and it is used for handling complicated header fields. If the next character is a tab or a space, then the CRLF or LF indicates a new line for this header field. The other way, designed for simple header fields, is finding out all parts of a simple header field while ignoring all blank characters before the end of the field.

Before processing header fields, a SIP stack needs to parse the first line to determine whether the SIP message is a request message or a response one because different message types have different first line syntaxes and contains different header fields. oSIP will compare the top four characters of the first line with the string ”SIP/” and the comparison is case sensitive. SER will compare the top seven characters of the first line with the string ”SIP/2.0” but this comparison is case insensitive. Open-SIP behaves in the same way as SER. According to RFC 3261, this comparison must be case insensitive, but it also requires that the implementation of SIP stacks to adopt upper case ”SIP/2.0” . This definition may result in different implementations and may lead to SIP IDS evasion.

This stage can be further divided into two sub-stages: header field name parsing and header field value parsing. The importance of the header field name parsing must be emphasized because the name of the header field defines the syntax of the whole header field. Only if the name is parsed properly, can it be ensured that the next step could be right. Also noted that SIP is easy to expand, thus nearly all SIP stacks allow unknown header fields and may not deal with them. So malformed parts can be concealed in the unknown headers to evade the IDS.

SIP evasion may occur by exploiting the header field name because the name of a header field is not unique. It has two types of transformation:

1) The header field name is case insensitive, so ”From” is same with ”from”.
2) Many Header fields have shorthand forms. For example, ”f” is short for ”from”.

oSIP, SER and Open-SIP all support shorthand forms and case-insensitive checking for common header fields. But for application specific header fields such as *User-Agent*, *Require* and *Subject*, they will rendering them directly to applications in SIP servers(e.g P-CSCF in IMS networks).

The way to parse the value part of a header field depends on the implementations of SIP stacks. Different implementations may adopt different ways and some ways may have drawbacks. oSIP and Open-SIP both adopt the method of string partition, which partitions a header field according to delimiters until all needed parts are obtained. Generally, the method of string partition will not check the charset of the header field and it usually assumes that delimiters will only come up in special parts(e.g. The left angle bracket will only appear in the name-addr part of the *From* header field). So if this assumption can not be satisfied, it can not be guaranteed that the parsing result is correct. For example, when oSIP starts to parse *From* header field, it will firstly search for the left angle bracket. If it is found, oSIP will believe that the from-spec part is in the name-addr form. However, this may not be the case since the left angle bracket can also appear in other parts (e.g generic-params) and the from-spec part is actually in the addr-spec form. SER adopts two methods to parse the value part of a header field. One is based on state machines and parse the value character by character. The other is the method of string partition. The former method is designed for complicated header fields since it can avoid problems produced by the method of string partition. The latter one is designed for simple header fields which mostly are in the key-value form.

Usually the problems of partition are caused by quoted strings in generic-param or display-name. According to RFC 3261, the content of a quoted string can contain many kinds of characters, especially delimiters, and it also allows escape characters with backlashes. Meanwhile, SIP allows the appearance of unknown parameters which enables users to insert any parameter into a SIP message.

1) Escape quotes in quoted-string
   Quoted-string mainly appears in three parts: display-name, generic-param and comment. As comments seldom used, we just take the other two parts into consideration.
   a) Quoted-string appears in display-name
      When oSIP finds a quote sign in display-name, it will check whether the quote is an escape quote so as to make sure that no escape quote will be adopted. SER will regard the second quote as the end of display-name and doesn't carry out escape quote checking. Open-SIP does nothing on display-name. It just extracts the part before the left angle bracket as display-name.
   b) Quoted-string appears in generic-param
      oSIP will not carry out escape quote checking. It just looks for semicolons and equal signs. SER's procedure is the same as the procedure it handles display-name. Open-SIP will regard all of semicolons in this part as delimiters of parameters and then find the parameter name and the parameter value using equal signs.
2) Delimiters in quoted-string
   oSIP only carries out escape quote checking in display-name and will guarantee that delimiters in display-name will not affect the string partition. However, when delimiters appears in generic-param, oSIP may deal with it in wrong ways. For example, if the *From* field is "From:SIP:abnormal@abnormal.com; tag=-1000; param1="user0"; param2="<SIP:user0@ims.com>; tag=1000;\r\n", oSIP will firstly look for the left angle bracket, then it would think that the header content is "<SIP:user0@ims.com>; tag=1000;\r\n". SER will guarantee that quotes must appear in pairs. So if no escape quote appears, delimiters in generic-param will be ignored. Open-SIP just partitions parameters by delimiters and may produce wrong results.
3) Others
   There are other ways to evade SIP IDSs. For example, illegal characters can be encoded into other forms to delay its handling, *NULL* can be brought in to affect string handling procedures. If SIP URI contains incorrect transformation of characters, SIP stacks usually doesn't recognize them and let them pass through SIP IDSs. But when applications decode the characters they may crash because these are illegal characters.
   In RFC 3261, there are four rules on transformation.
   a) SIP URI supports the transformation of percent signs. For example, "SIP:%61lice@atlanta.com;transport=TCP" is equal to "SIP:alice@atlanta.com;transport=tcp".
   b) In quoted-string, escape characters can be used. For example, "\n" is regarded as a CR.
   c) SIP adopts the UTF-8 encoding, so it can represents wide characters.
   d) Content-Encoding header field can specify the encoding method of the body. For example, the content can be encoded in UTF-16.

This stage's duty is to check whether the semantics of the received SIP message is correct. For example, whether the required header fields are appeared or not, whether the value of *Content Length* header field is equal to the length of the body.

Previous evasion measures can be utilized to evade the checking in this stage.

B. SIP IDS evasion rules

Based on the analysis on the flaws on SIP IDS, we presented SIP IDS evasion rules, which can be classified into two categories:

1) Target systems have some flaws, and normal SIP messages will have some attack effects.
   a) Insert some carriage returns, spaces, tabs, line breaks or combination of them into white spaces. This rule may cause the target systems to regard a multi-line header as two or more headers.
   b) Change the case of SIP version in the status line. This rule may cause the target systems to recognize it as a request message.
   c) Change the case of header field names. This rule may cause the target systems unable to recognize them.
   d) Change some header field names into abbreviated forms. This rule may cause the target systems unable to recognize them.

   e) Add some colons, semicolons or commas into quoted-string parts. This rule may cause the target systems to parse them in a wrong way.
   f) Change the case of the scheme of SIP URIs. This rule may cause the target systems unable to recognize the scheme.
2) SIP IDSs have some flaws and some malformed SIP messages can not be identified.
   a) Replace a CRLF with a carriage return or a line break.
   b) Add some characters to the end of the SIP version in the status line. For example, ”SIP/2.0ddd 200 OK” may be recognized as a normal status line by IDSs.
   c) Change the SIP version in the status line. For example, ”SIP/3.0” indicates a wrong version.
   d) Use the name-addr form in the request URI where only the addr-spec form should be used. For example,”REGISTER bob<SIP:bob@ims.com> SIP/2.0”.
   e) Insert some carriage returns, line breaks or combination of them into the quoted-string.

C. SIP disguising algorithm

To apply the SIP IDS evasion rules, we designed an algorithm SDA(SIP Disguising Algorithm), as showned in figure 1.

```
Algorithm 1 SIP Disguising Algorithm - SDA.
Input: The rule set, $rs$;
    A normal message, $nm$;
    A selection strategy, $s$.
Output: A malformed SIP message, $mm$.
 1: select a rule, $r$, from $rs$ according to $s$.
 2: partition $nm$ into a collection of headers, $hc$.
 3: for all $h \in hc$ do
 4:   partition $h$ into a collection of elements, $ec$.
 5:   for all $e \in ec$ do
 6:     if $e.type \in r.type\_set$ then
 7:       change $e$ to a new element $ne$ according to $r$.
 8:     else
 9:       copy $e$ to a new element, $ne$.
10:     end if
11:     put $ne$ into a new collection of elements, $nec$.
12:   end for
13:   assemble $nec$ to a new header, $nh$.
14:   put $nh$ into a new collection of headers, $nhc$.
15: end for
16: assemble $nhc$ to a malformed SIP message, $mm$.
17: return $mm$.
```

Firstly, partition a SIP message into a collection of header fields. Secondly, for each header filed in the collection, partition it into a collection of elements. Thirdly, according to a selection strategy, select one or more rules and change the related elements. Finally, assemble a new SIP message.

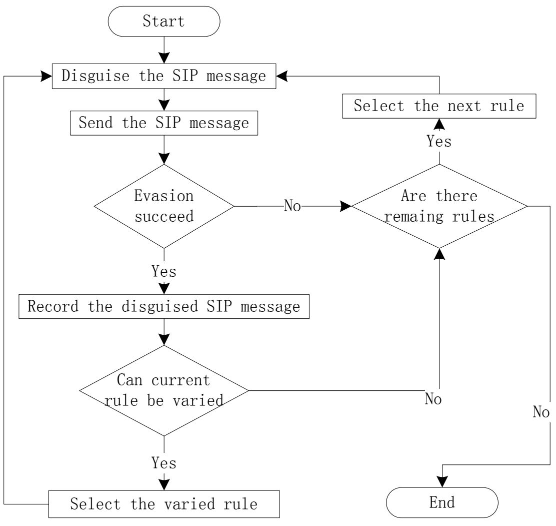

Figure 3. Selection Strategy

Suppose there are m rules and a message can be divided into n elements, the time complexity of the algorithm is O(m+n). To improve the possibility of SIP IDS evasion, we bring up a selection strategy, as shown in figure 3.

The number of rules is always limited. To improve the possibility of evading signature-based SIP IDSs, some variability should be added to these rules. This can be achieved in two ways:

1) The combination of multiple rules. In other words, more than one rules are applied to one SIP message.
2) The implementation of rules. A rule will change the content of one SIP message using three methods:
   a) Conversion Method. For example, case conversion can be done to parts or all elements.
   b) Insertion Method. Different characters or different number of characters can be inserted into some elements.
   c) Replacement Method. Based on some templates, the whole element can be replaced with another one.

D. SIP IDS evasion testing system

To evaluate the defending capability of SIP IDS on SIP evasion, we designed a system, ETS(Evasion Testing Sytem), as shown in figure 4.

The generator will generate SIP messages which are processed by the evasion rules. Then the sender will decide where to send these messages. Target emulator will wait for arrival of these SIP messages, then analyze the content to decide whether they contain attacking payload. In order to calculate the percentage of evasion, the sender would send two SIP messages to the target emulator, one is through the SIP IDS under test, one is not. Recorder has the duty of recording successful evasion SIP messages. Counter is responsible for performing some statistics about evasion such as the number of effective evasion rules. Based on the statistics we can find out that for the SIP IDS under test which rules are effective. We

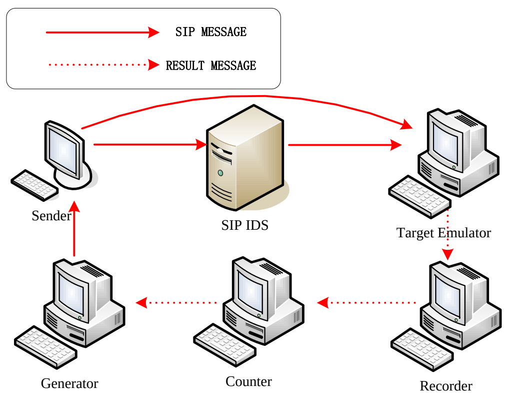

Figure 4. Evasion Testing System - ETS

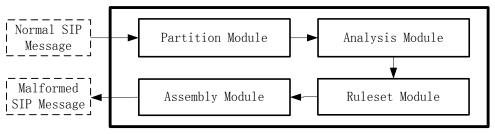

Figure 5. Generator Components

also can adjust the selection strategy to generate more SIP messages to obtain more effective rules.

The generator is the core of this system. It consists of four components(Figure 5):

The duty of the partition module is to partition a normal SIP message into a collection of header fields. For example, a normal SIP message contains a *from* header field, a *to* header field , one or more *via* header fields, and so on. The analysis module will analyze all header fields in the collection produced by previous module, and divide a header field into a collection of elements. The encoding of SIP is specified using an augmented BNF, so a header field can be divided to many parts according to these rules. For example, a *call-id* header field contains a header field name that is case insensitive, a colon that helps obtain the header field name and a number named *call id*.The ruleset module is the core of the generator, and any rule can be added into this part to change any element of any header field, so most disguising technologies are applied here. For example, if there is a *user-agent* header field, a malformed *from* header field can be inserted into it. The assembly module is responsible for constructing the disguised malformed SIP message in the end.

V. Experiments

To verify the effectiveness of the proposed method, we have carried out two groups of experiments based on Snort:

1) Snort version 2.9 has implemented a preprocessor for SIP to detect malformed SIP messages. It also provided a rule set. We use the version 2.8 rule set in this experiment.
   The preprocessor for SIP in Snort checks for the existence of required header fields such as the *Via* header field. ETS utilizes *User-Agent* header field to generate a malformed SIP message and evades it successfully.

```
REGISTER sip:ims.com SIP/2.0\r\n
Call-ID: b12057c9d426f2e82ff13cbfab5d056e@192.168.1.101\r\n
CSeq: 1 REGISTER\r\n
From: "user1" <sip:user1@ims.com>;tag=1000\r\n
To: "user1" <sip:user1@ims.com>\r\n
Expires: 3600\r\n
Authorization: Digest username="user1@ims.
com",realm="ims.com"
,nonce="",response="",uri="sip:ims.com"\r\n
Supported: path\r\n
Contact: <sip:192.168.1.101:5060>\r\n
P-Preferred-Identity: "user1" <sip:user1@ims.com>\r\n
P-Access-Network-Info: 3GPP-UTRAN-TDD; utran-cell-id-
3gpp=00000000\r\n
Privacy: none\r\n
User-Agent: Fraunhofer FOKUS/NGNI Java IMS UserEndpoint
FoJIE 0.1 (jdk1.3), "\nVia: SIP/2.0/UDP 192.168.1.101:5060;
branch=z9hG4bK6a34ee2e9bdd86c6
63abd2bb5ff0302c \n" \r\n
Allow: INVITE,ACK,CANCEL,BYE,MESSAGE,NOTIFY\r\n
Content-Length: 0\r\n\r\n
```

Figure 6. Example 1

Example 1:
User-Agent: Fraunhofer FOKUS/NGNI Java IMS UserEndpoint FoJIE 0.1 (jdk1.3), ”\nVia: SIP/2.0/UDP 10.108.114.246:5064; branch=z9hG4bK6a34ee2e9bdd86c6 63abd2bb5ff0302c \n” \r\n
Analysis:
The corresponding snort rule is ”alert udp any any -> any any (msg:”VOIP-SIP Via header missing SIP field”; content:”Via|3A|”; nocase; pcre:”/^Via\x3A\s+(?!SIP\x2F2\x2E0)/smi”; reference:url,www.ietf.org/rfc/rfc3261.txt; sid:11975; rev:2;)”. The rule checks whether a *Via* header field is existent using a simple regular expression. We found that the preprocessor of snort partitions a message by CRLF or LF and it has no rules on the checking of *User-Agent* header field. So we construct a *Via* header field string and put it into the *User-Agent* header field. This message can evade snort while SIP servers behind it would not obtain a *Via* header field. The complete message is shown in Figure 6.
Snort will check whether the *From* header field contains format strings using the Regex Match method. However, ETS can evade it using the shorthand form of the *From* header field name.
Example 2:
F: ”user1%d” <SIP:user1@ims.com>;tag=1000\r\n
Analysis:
The corresponding snort rule is ”alert udp any any -> any any (msg:”VOIP-SIP From header format string attempt”; content:”From|3A|”; nocase; content:”%”; distance:0; pcre:”/^From\x3A\s*[^\r\n%]*%/smi”; reference:url,www.ee.oulu.fi/research/ouspg/ protos/testing/c07/SIP/; reference:url,www.ietf.org/rfc/rfc3261.txt; sid:11988; rev:2;)”. This rule tries to find a format string in

```
REGISTER sip:ims.com SIP/2.0\r\n
Call-ID: b12057c9d426f2e82ff13cbfab5d056e@192.168.1.101\r\n
CSeq: 1 REGISTER\r\n
F: "user1%d" <sip:user1@ims.com>;tag=1000\r\n
To: "user1" <sip:user1@ims.com>\r\n
Via: SIP/2.0/UDP 192.168.1.101:5060;
branch=z9hG4bK6a34ee2e9bdd86c663abd2bb5ff0302c \r\n
Expires: 3600\r\n
Authorization: Digest username="user1@ims.
com",realm="ims.com"
,nonce="",response="",uri="sip:ims.com"\r\n
Supported: path\r\n
Contact: <sip:192.168.1.101:5060>\r\n
P-Preferred-Identity: "user1" <sip:user1@ims.com>\r\n
P-Access-Network-Info: 3GPP-UTRAN-TDD; utran-cell-id-
3gpp=00000000\r\n
Privacy: none\r\n
User-Agent: Fraunhofer FOKUS/NGNI Java IMS UserEndpoint
FoJIE 0.1 (jdk1.3)\r\n
Allow: INVITE,ACK,CANCEL,BYE,MESSAGE,NOTIFY\r\n
Content-Length: 0\r\n\r\n
```

Figure 7. Example 2

From header field. However it neglects that *From* header field has a shorthand name *f*. So using this shorthand form, some format strings can be inserted into *From* header field and evade the detection of snort. The complete message is shown in Figure 7.

Snort will check whether overflow exists in the *From* header field using the Regex Match method. The checking can be evaded by ETS using CR and LF.

Example 3:

From: ”user\r ...\r ...\r” <SIP:user1@ims.com>; tag=1000\r\n

Analysis:

The corresponding snort rule is ”alert udp any any -> any any (msg:”VOIP-SIP from header field buffer overflow attempt”; content:”From|3A|”; nocase; pcre:”/^From\x3A\s+[^\r\n]256/smi”; reference:url,www.cert.org/advisories/CA-2003-06.html; reference:url,www.ietf.org/rfc/rfc3261.txt; sid:11978; rev:3;)”. This rule checks whether the *From* header field contains more than 256 characters. However we find out that when a CR or LF is encountered, this check will stop. So we insert some LFs to the SIP message to evade the detection. If SIP servers partition a message with CRLF, then the overflow payload can still work. The complete message is shown in Figure 8.

2) Based on oSIP and Snort, we have implemented a SIP IDS. This system will firstly put a SIP message into oSIP to check whether the syntax of it is correct, and then we will set a rule to check whether the *From* header field exists.

Using the *User-Agent* header field ETS makes this system to believe that the *From* header field does exist.

Example 4:

User-Agent: Fraunhofer FOKUS/NGNI Java IMS

```
REGISTER sip:ims.com SIP/2.0\r\n
Call-ID: b12057c9d426f2e82ff13cbfab5d056e@192.168.1.101\r\n
CSeq: 1 REGISTER\r\n
From: "user\r ...\r ...\r" <sip:user1@ims.com>;
tag=1000\r\n
To: "user1" <sip:user1@ims.com>\r\n
Via: SIP/2.0/UDP 192.168.1.101:5060;
branch=z9hG4bK6a34ee2e9bdd86c663abd2bb5ff0302c \r\n
Expires: 3600\r\n
Authorization: Digest username="user1@ims.
com",realm="ims.com"
,nonce="",response="",uri="sip:ims.com"\r\n
Supported: path\r\n
Contact: <sip:192.168.1.101:5060>\r\n
P-Preferred-Identity: "user1" <sip:user1@ims.com>\r\n
P-Access-Network-Info: 3GPP-UTRAN-TDD; utran-cell-id-
3gpp=00000000\r\n
Privacy: none\r\n
User-Agent: Fraunhofer FOKUS/NGNI Java IMS UserEndpoint
FoJIE 0.1 (jdk1.3)\r\n
Allow: INVITE,ACK,CANCEL,BYE,MESSAGE,NOTIFY\r\n
Content-Length: 0\r\n\r\n
```

Figure 8. Example 3

```
REGISTER sip:ims.com SIP/2.0\r\n
Call-ID: b12057c9d426f2e82ff13cbfab5d056e@192.168.1.101\r\n
CSeq: 1 REGISTER\r\n
To: "user1" <sip:user1@ims.com>\r\n
Via: SIP/2.0/UDP 192.168.1.101:5060;
branch=z9hG4bK6a34ee2e9bdd86c663abd2bb5ff0302c \r\n
Expires: 3600\r\n
Authorization: Digest username="user1@ims.
com",realm="ims.com"
,nonce="",response="",uri="sip:ims.com"\r\n
Supported: path\r\n
Contact: <sip:192.168.1.101:5060>\r\n
P-Preferred-Identity: "user1" <sip:user1@ims.com>\r\n
P-Access-Network-Info: 3GPP-UTRAN-TDD; utran-cell-id-
3gpp=00000000\r\n
Privacy: none\r\n
User-Agent: Fraunhofer FOKUS/NGNI Java IMS UserEndpoint
FoJIE 0.1 (jdk1.3), "\rFrom: "user0" <sip:"user0"@ims.
com>; tag=1000\r"\r\n
Allow: INVITE,ACK,CANCEL,BYE,MESSAGE,NOTIFY\r\n
Content-Length: 0\r\n\r\n
```

Figure 9. Example 4

UserEndpoint FoJIE 0.1 (jdk1.3), ”\rFrom: ”user0” <SIP:”user0”@ims.com>; tag=1000\r”\r\n

Analysis:

oSIP will partition a message using CR, LF or CRLF into a collection of headers. So the SIP IDS will obtain the *From* header field, which is ”From: ”user0” <SIP:”user0”@ims.com>; tag=1000”. However, SIP servers may not obtain it because this message does not have one in fact. The complete message is shown in Figure 9.

If the *From* header field is malformed, ETS will insert a normal SIP uri in the generic-param part to make oSIP think that SIP uri is SIP:user0@ims.com not SIP:abnormal@abnorma.com.

Example 5:

From:SIP:abnormal@abnormal.com; tag=-1000; param1=”user0”; param2=”<SIP:user0@ims.com>; tag=1000;”\r\n

```
REGISTER sip:ims.com SIP/2.0\r\n
Call-ID: b12057c9d426f2e82ff13cbfab5d056e@192.168.1.101\r\n
CSeq: 1 REGISTER\r\n
From:sip:abnormal@abnormal.com; tag=-1000;
param1="user0";
param2="<sip:user0@ims.com>; tag=1000;"\r\n
To: "user1" <sip:user1@ims.com>\r\n
Via: SIP/2.0/UDP 192.168.1.101:5060;
branch=z9hG4bK6a34ee2e9bdd86c663abd2bb5ff0302c \r\n
Expires: 3600\r\n
Authorization: Digest username="user1@ims.
com",realm="ims.com"
,nonce="",response="",uri="sip:ims.com"\r\n
Supported: path\r\n
Contact: <sip:192.168.1.101:5060>\r\n
P-Preferred-Identity: "user1" <sip:user1@ims.com>\r\n
P-Access-Network-Info: 3GPP-UTRAN-TDD; utran-cell-id-
3gpp=00000000\r\n
Privacy: none\r\n
User-Agent: Fraunhofer FOKUS/NGNI Java IMS UserEndpoint
FoJIE 0.1 (jdk1.3)
Allow: INVITE,ACK,CANCEL,BYE,MESSAGE,NOTIFY\r\n
Content-Length: 0\r\n\r\n
```

Figure 10. Example 5

TABLE I.
SIP MALFORMATION DETECTION OF SNORT

Message Type	**Header**	**Malformation Type**	**Detection Rate**
All	All but User-Agent	All	100%
All	User-Agent	All	0%

Analysis:
When oSIP parses the *From* header field, it will firstly try to find a left angle bracket. If it is found, oSIP will think that the from-spec part is in the name-addr form without checking whether the left angle bracket belongs to other parts(e.g. generic-params)or not. So attackers can utilize this bug to insert malicious elements in the *From* header field to evade the detection. The complete message is shown in Figure 10.

In the end, we carried out an experiment to calculate the rate of successful evasion to Snort. Firstly, we generated sixty normal SIP messages including ten *ACK* messages, ten *BYE* messages, ten *CANCEL* messages, ten *INVITE* messages, ten *OPTIONS* messages and ten *REGISTER* messages. Secondly, we inserted three types of typical malformation elements(*Invalid Characters*, *Format String* and *Overflow* into six SIP headers including *CSeq* header, *From* header, *To* header, *Call-ID* header, *Via* header and *User-Agent* header which can appear in every SIP message. Finally, we generated one thousand and eighty malformed SIP messages. When these messages went through Snort, nearly 83.33 percents of them are detected, as shown in Table I.

In order to evade Snort, we adopted five evasion rules:

1) Adopt the shorthand form of the name of the header containing malformation elements.
2) Randomly change the characters of the name of the header to their upper or lower cases.
3) Rearrange the value of the header to multiply lines but do not place the malformation elements on the first line.
4) Use backlashes.
5) Utilize the UTF-8 encoding diversity.

TABLE II.
EVASION RELATION

Element / Evasion Rule	**Invalid Characters**	**Format String**	**Overflow**
Rule 1	Evaded	Evaded	Evaded
Rule 2	Evaded	Evaded	Evaded
Rule 3	Evaded	Evaded	Evaded
Rule 4	Evaded	Evaded	Not Evaded
Rule 5	Evaded	Evaded	Not Evaded

TABLE III.
EVASION RATE

Rule	**Unused Detection Rate**	**Detection Rate**	**Evasion Rate**
Rule 1	83.33%	0%	100%
Rule 2	83.33%	83.33%	0%
Rule 3	83.33%	0%	100%
Rule 4	83.33%	0%	100%
Rule 5	83.33%	83.33%	0%

The relations between evasion rules and the malformation elements is shown in table II.

Then we applied these evasion rules to disguise the above one thousand and eighty malformed SIP messages and obtain four thousand six hundred and eighty SIP messages. When all these SIP messages were sending to snort, only one thousand and five hundred malformed SIP messages are detected. So the evasion rate is about 61.54% on average(table III).

The experiment shows that snort will not look for these three types of malformation elements in the *User-Agent* header. Therefore, snort should be enhance to defend the evasion through this header field.

VI. CONCLUSION

In this paper, we analyzed the flaws of SIP IDSs on their capability of evasion detection. Then we proposed the SIP evasion rules as well as the related SIP disguising algorithm. Also, we designed a evasion testing system using the proposed evasion rules and disguising algorithm to evaluate the evasion rate of SIP IDSs. We conducted evasion experiments on snort and oSIP-based IDS. The results show that the proposed method can improve the evasion rate of SIP messages considerably, thus can help enhance the defending capability of SIP IDS.

ACKNOWLEDGMENT

We thank professors and colleagues for numerous discussions concerning this work, State Key Laboratory of Networking and Switching Technology for assistance, and the reviewers for their detailed comments.

REFERENCES

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, and R. Sparks, "Sip: Session initiation protocol," RFC 3261, Internet Engineering Task Force, June 2002.

[2] D. Geneiatakis, T. Dagiuklas, G. Kambourakis, C. Lambrinoudakis, S. Gritzalis, S. Ehlert, and D. Sisalem, "Survey of security vulnerabilities in session initiation protocol," *IEEE Communications Surveys and Tutorials - COMSUR*, vol. 8, no. 3, pp. 68 – 81, Dec. 2006.

[3] L. Xie, F. Yu, and C. Xu, "Distributed firewall with intrusion detection system," *Journal of Computers*, vol. 7, no. 12, 2012. [Online]. Available: http://ojs.academypublisher.com/index.php/jcp/article/view/jcp071231103115

[4] B. Meng, W. Wang, and W. Chen, "Verification of resistance of denial of service attacks in extended applied pi calculus with proverif," *Journal of Computers*, vol. 7, no. 4, 2012. [Online]. Available: http://ojs.academypublisher.com/index.php/jcp/article/view/jcp0704890899

[5] W. Jiang, X. Fan, D. Duanmu, and Y. Deng, "A new security risk assessment method of website based on generalized fuzzy numbers," *Journal of Computers*, vol. 8, no. 1, 2013. [Online]. Available: http://ojs.academypublisher.com/index.php/jcp/article/view/jcp0801136145

[6] S. Niccolini, R. G. 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*, Apr. 2006, pp. 47 – 52.

[7] D. Geneiatakis, G. Kambourakis, C. Lambrinoudakis, T. Dagiuklas, and S. Gritzalis, "A framework for protecting a sip-based infrastructure against malformed message attacks," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 51, no. 11, pp. 2580 – 2593, July 2007.

[8] D. Geneiatakis, G. Kambourakis, T. Dagiuklas, C. Lambrinoudakis, and S. Gritzalis, "A framework for detecting malformed messages in sip networks," in *IEEE Workshop on Local and Metropolitan Area Networks - LANMAN , 2005*, Sept. 2005, p. 5.

[9] S. Aziz and M. Gul, "A self learning model for detecting sip malformed message attacks," in *2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*, Oct. 2010, pp. 744– 749.

[10] K. Rieck, S. Wahl, P. Laskov, P. Domschitz, and K. robert 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: Second International Conference, IPTComm 2008, Heidelberg, Germany*, Oct. 2008, pp. 90 – 106.

[11] L. Meilian, W. Lei, and Z. Xing, "Research and implementation of ims simulation system based on ns2," in *International Conference on Wireless Communications, Networking and Mobile Computing - WiCom , 2008*, Oct. 2008, pp. 1 – 5.

[12] University of Oulu, "Protos test-suite: c07-sip," https://www.ee.oulu.fi/research/ouspg/PROTOS_Test−Suite_c07−sip. Accessed on October 5th, 2012.

[13] "Snort home page," http://www.snort.org. Accessed on October 5th, 2012.

[14] "Perl compatible regular expressions," http://www.pcre.org. Accessed on October 5th, 2012.

[15] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "Sip: Session initiation protocol," RFC 2543, Internet Engineering Task Force, June 2002.

[16] T. Cheng, Y. Lin, Y. Lai, and P. Lin, "Evasion techniques: Sneaking through your intrusion detection/prevention systems," *Communications Surveys & Tutorials, IEEE*, vol. 14, no. 4, pp. 1011– 1020, Oct. 2011.

[17] Daniel J. Roelker, "HTTP IDS Evasions Revisited," http://www.idsresearch.org/. Accessed on October 5th, 2012.

[18] M. Boltz, M. Jalava, and J. Walsh, "New methods and combinatorics for bypassing intrusion prevention technologies," Stonesoft, Tech. Rep., 2010.

[19] S. Gold, "Advanced evasion techniques," *Network Security*, vol. 2011, no. 1, pp. 16 – 19, Jan. 2011.

[20] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "Sip: Session initiation protocol," RFC 2543, Internet Engineering Task Force, Mar. 1999.

[21] "oSIP library home page," http://www.gnu.org/software/osip/osip.html. Accessed on October 5th, 2012.

[22] "SER - SIP Express Router home page," http://www.iptel.org/ser?from=0&comments_per_page=10. Accessed on October 5th, 2012.

[23] "Opensipstack library home page," http://www.opensipstack.org. Accessed on October 5th, 2012.