Telecommun Syst (2010) 45: 61–76
DOI 10.1007/s11235-009-9234-1

Blocking attacks on SIP VoIP proxies caused by external processing

Ge Zhang · Simone Fischer-Hübner · Sven Ehlert

Published online: 19 November 2009

Abstract As Voice over IP (VoIP) applications become increasingly popular, they are more and more facing security challenges that have not been present in the traditional Public Switched Telephone Network (PSTN). One of the reasons is that VoIP applications rely heavily on external Internet-based infrastructures (e.g., DNS server, web server), so that vulnerabilities of these external infrastructures have an impact on the security of VoIP systems as well. This article presents a Denial of Service (DoS) attack on VoIP systems by exploiting long response times of external infrastructures. This attack can lead the whole VoIP system in a blocked state thus reducing the availability of its provided signalling services. The results of our experiments prove the feasibility of blocking attacks. Finally, we also discuss several defending methods and present an improved protection mechanism against blocking attacks.

Keywords Session Initiation Protocol (SIP) · Voice over IP (VoIP) · Denial of Service (DoS) · Security · DNS · Protection mechanism

1 Introduction

The emergence of VoIP has offered numerous features for both end users and providers alike, such as low cost, flexible services, simplified configuration, etc. These advantages are difficult to realize in traditional PSTN. However, as a closed network environment, PSTN provides high security, and particularly availability and reliability. That is why an extremely low number of attacks on PSTN have been reported until now. In contrast to PSTN, VoIP is more vulnerable due to two main reasons: Firstly, contrary to PSTN, VoIP is *based on an open network environment*, the Internet. Thus, a VoIP infrastructure can be easily accessed by attackers with automation tools. Secondly, current *VoIP applications rely heavily on external Internet-based infrastructures* (e.g., DNS server, web server). As a result, vulnerability of these external infrastructures also affect VoIP systems. The main security threats against VoIP systems are summarized in [1, 2].

The Session Initiation Protocol (SIP) [3] is designed as a signalling protocol standard for VoIP services running on the Internet and 3G Realms. In this article, we present a kind of DoS attack targeting SIP-based VoIP infrastructures using crafted SIP messages, which we call a **blocking attack**. Since VoIP systems require support from external hosts on the Internet (e.g., recursive domain-name resolving, downloading certificates, etc.), communication with external hosts may increase the latency of SIP services. The latency varies primarily according to hardware and network conditions of external hosts. Better hardware and network conditions of external hosts yield to lower latency. Unfortunately, this is beyond the control of VoIP systems. Actually, there are many hosts on the Internet with poor hardware or network conditions. In this article, we call them **high-latency hosts**. An attacker can craft special messages for a blocking attack in order to make a victim SIP proxy to interact with these high-latency hosts. Then, the victim proxy has to spend additional time on these attacking messages

G. Zhang (✉) · S. Fischer-Hübner
Karlstad University, Karlstad, Sweden
e-mail: ge.zhang@kau.se

S. Fischer-Hübner
e-mail: simone.fischer-huebner@kau.se

S. Ehlert
Fraunhofer FOKUS, Berlin, Germany
e-mail: sven.ehlert@fokus.fraunhofer.de

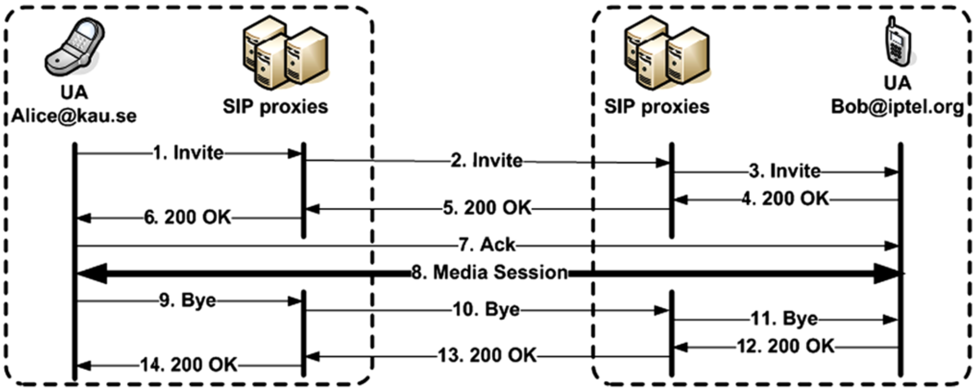


Fig. 1 A general overview of SIP components and the working procedure

and thus cannot handle messages from legal users. The results of our experiments show that blocking attacks can be easily launched against SIP proxies by using external DNS servers or web servers. We also discuss several defending solutions and propose an improved solution based on a priority mechanism.

This article builds on previous work published in [4]. Here, we innovate in three new directions: Firstly, we show a new method of blocking attacks by using certificate downloading besides domain name resolving. Secondly, a formalized model is given to analyse the blocking attacks in detail. Finally, we present an improved solution, called priority mechanism, for attack mitigation.

The rest of this article is organized as follows. In Sect. 2 we give a short introduction to VoIP using SIP. Section 3 presents related work. Then, we analyze the attacking method in Sect. 4. We propose our testbed and experiments in Sect. 5. Several countermeasure solutions are discussed in Sect. 6. Finally, Sect. 7 provides conclusions for this article.

2 SIP-based VoIP

SIP works as a signaling protocol at the application layer of the TCP/IP model aiming for establishing or tearing down media sessions between two parties. A basic SIP infrastructure consists of User-Agents (UAs) and several SIP servers, such as registrar servers, proxy servers (called proxy or proxies in the remainder of this article), etc. UAs are the users' equipments which generate, receive and response SIP messages. Registrar servers enable users to login to corresponding domains. Proxies forward SIP messages within SIP networks.

The general SIP-based telephony calling setup is shown in Fig. 1. There are two SIP users located in different domains in this scenario: Alice, a caller, is located in the domain kau.se and Bob, a callee, is located in iptel.org. Initially, Alice sends an INVITE message to one of the local proxies. This INVITE message indicates that she wants to talk to Bob at iptel.org. Then, the local proxy forwards this

SIP headers
```
INVITE sip:bob@iptel.org SIP/2.0
From: sip:alice@kau.se; tag=1b34283
To: sip:bob@iptel.org
Call-Id: 1-15673@193.11.155.22
Contact: <sip:alice@193.11.155.22:5069>
Content-Type: application/sdp
```
Message Body
```
V=0
o=alice 2891234526 2891234526 IN IP4 alice.kau.se
s=Let us talk for a while
c=IN IP4 193.11.155.22
t=0 0
m=audio 20002 RTP/AVP 0
```

Fig. 2 An example of SIP INVITE request

INVITE message to the remote proxy at iptel.org. The request is finally delivered to the UA of Bob. If Bob wants to accept the call, his UA will reply with a 200 OK message back through the proxies. After Alice has sent an ACK message to confirm the request, the signaling handshaking is accomplished. Thus, Alice and Bob will build a peer-to-peer media session in which they can exchange voice packets. When Alice wants to tear down the conversation, her UA will send a BYE message to Bob, and Bob's UA will reply with a 200 OK message. Then the call is terminated.

An example of a SIP INVITE message is shown in Fig. 2. The format of SIP messages is similar to HTTP [5], with message headers and corresponding values. The destination of a SIP message (SIP identity of the callee) is provided in the first line of the message. The remaining SIP message headers in Fig. 2 are explained as follows.

To: indicates the SIP identity of the callee.
From: indicates the SIP identity of the caller.
Call-ID: unique identifier for each call. It is used to indicate a SIP transaction.
Contact: indicates the actual location where the sender can be reached. The value can be different from the *from* field.
Content-Type: indicates which kind of payload is included in this message.

In this example, the INVITE message contains a message body using the Session Description Protocol (SDP) [6] as the payload. This SDP message body is used to negotiate the media type and format of the proposed session.

SIP applications are time-sensitive. Users probably will give up their calling requests if their calls take a long time to be built up. As defined in [3], after sending a message, a UA will wait for its response. However, if no response is received during a certain time interval, the UA will retransmit the message. The retransmission could be done up to 7 times until a timeout occurs and if still no response for the message is received, the UA will consider the message as failed and give up waiting for its response. The default timeout for each SIP message is 32 seconds, that is, after sending a SIP message, a UA should receive its response within 32 seconds. Thus, if the SIP proxy is busy and cannot generate the response to users in time, the users cannot successfully place calls. This real-time requirement poses a great challenges for SIP services and makes SIP services vulnerable to DoS attacks.

3 Related work

The objective of DoS attacks is to reduce the availability of services as long as possible. The motive of attackers could be fun or profit. Some recent research has focused on DoS attacks on SIP VoIP infrastructures.

Sisalem et al. [7] investigated the issues of DoS attacks targeting SIP infrastructures. They developed a taxonomy of attacks by different exploitable resources: CPU, memory and bandwidth. They also mentioned the blocking attack against SIP proxies using high-latency DNS servers, but no experiments were performed.

A stateful SIP proxy has to consume memory resources to keep the transaction states of unfinished SIP transactions. Therefore, stateful SIP proxies are especially vulnerable to INVITE flooding attack, in which an attacker floods SIP proxies with only INVITE messages to create a large number of broken transactions. Sengar et al. [8] proposed a machine learning method to distinguish legal against INVITE flooding attacks. Based on this method, they firstly characterize legal SIP traffic behavior by summarizing the normalize frequency of INVITE, 200 OK, ACK and BYE messages. Compared with the normalized samples, the attacking traffic behavior can be easily detected by an algorithm called Hellinger distance. Moreover, [9–11] propose specification-based detection methods using SIP state machine models to counter INVITE flooding attacks.

Conner et al. [12] proposed a ringing-based DoS attack to consume memory resources of stateful SIP proxies. Different to INVITE flooding, this attack exploits the potential long time 180 Ringing state. They also designed an algorithm to optimize the system.

Geneiatakis et al. [13] introduced a mechanism to detect malformed SIP messages. Malformed messages are non-standard SIP messages, which are skillfully generated by attackers in order to exploit the implementation flaw of SIP infrastructures. Malformed message may drive the services to various unstable states and consequently cause DoS. The detection of malformed messages is based on a message signature, which is defined by using regular expressions according to the SIP grammar. Their tests prove that the overall processing overhead caused by this detection is insignificant.

The blocking attack we present in this article is a kind of Denial of Service attacks. However, blocking attacks are not targeted to create unfinished transaction states as discussed in related works. The blocking attacks that we discuss are effective on both stateful and stateless proxies. Furthermore, the attacking messages are well-formed. Thus, current protection mechanisms are not effective as a countermeasure against such blocking attacks as described in this article.

In our previous work [4], we represented a blocking attack on SIP VoIP infrastructures by exploiting high-latency DNS servers. We also implemented an unblocking cache mechanism to defend against such an attack. Continuing, this article addresses aspects of the blocking attacks. We formalize and analyze why the vulnerability can cause a Denial-of-Service in detail and propose a new type of blocking attack using high latency web servers. Besides, we perform our experiments with new parameters. Finally, we discuss the problems of the previous designed solutions and propose the priority mechanism as an improved protection mechanism.

4 Blocking attacks

Blocking attacks introduced in this article is a kind of DoS attack targeting SIP proxies. As explained above and illustrated in Fig. 1, SIP proxies play a very important role in SIP networks. Thus, the VoIP services cannot be provided any longer if SIP proxies stop operating due to attacks.

To analyze how such a blocking attack works, we illustrate a basic working flow of SIP proxies in Fig. 3.[1] When a SIP proxy receives a SIP message, it will insert this message into a message buffer. Next, the buffered messages will be distributed to the parallel working processes. (In practice, a SIP proxy can be configured with parallel working processes to enhance its performance. Each working process is then responsible for one messages synchronously.) We classify the steps of each processing into two categories, named **internal processing** and **external processing**, respectively.

[1]However, the actual order of the internal and external processing depends on the concrete implementation of the proxy. The order of the internal and external processing is irrelevant for our topic.

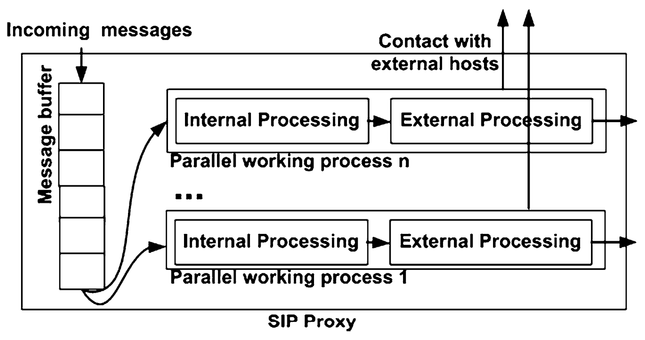


Fig. 3 The procedure of proxy working: the proxy has to contact with other hosts for external processing

Internal processing: Internal processing consists of the processing steps that are only performed directly within the local proxy (e.g., message parsing, computing for verification, etc.). The time consumption for internal processing mostly depends on the computational capability of the proxy itself. Thus, the time consumption for internal processing is predictable.

External processing: External processing consists of the processing steps that do not take place locally, but also remotely (e.g., contact with a DNS server to issue a domain name request, or download a certificate from a web server to verify a SIP message). The purpose of external processing is to get **Related External Information (REI)** (e.g., DNS mapping of the destination domain or a certificate of the sender's domain) for message processing. Since the SIP proxy has to communicate with external hosts (DNS server, web server) over the Internet, the time consumption for external processing depends not only on the capability of the local proxy, but also on the hardware and network conditions of the remote hosts. As a consequence, the time consumption for external processing is unpredictable. Furthermore, the SIP proxy will contact the external hosts according to the information contained in SIP messages. Therefore, the SIP users who generate SIP messages can specify which external hosts a SIP proxy should contact.

Blocking attacks addressed in this article exploit external processing. The vulnerability is based on two aspects. Firstly, SIP users have the ability to select which external hosts that the SIP proxy should contact. Secondly, the time consumption for SIP proxies needed to communicate with external hosts is unpredictable. Thus, an attacker can perform a series of pretests to collect a list of external hosts which could be time consuming to communicate with, defined as **high-latency hosts**. The measurements in [14] show that the connection latency on the Internet strongly depends on the physical distances between the hosts. The attacker can easily find high-latency hosts which are in a remote location. Then, the attacker can craft attacking messages and send them to a victim SIP proxy. The attacking messages require a victim proxy to contact these high-latency hosts for external processing. As a result, the proxy spends a long time on the attacking messages so that it cannot handle the legal messages in due time.

To explain this problem better, we formalize the processing model as follows. The purpose of this model is twofold: Firstly, it illustrates the parameters which influence the throughput of a proxy and allows us to discuss how far these parameters can be controlled by an attacker. Secondly, this model also demonstrates how perspective countermeasures can be influenced by these parameters. We assume that a SIP proxy receives a batch of SIP messages during a period of time T with a constant rate R. The proxy is equipped with n parallel working processes. Each process can work independently. The average processing time needed for each message of this batch on each parallel process is assumed to be $\bar{t}$. Then, the processing capability of this SIP proxy during time T is at most C if all the parallel working processes are fully working during time T.

$$C = \frac{n}{\bar{t}} \tag{1}$$

Further, we separate $\bar{t}$ as $\bar{t} = \overline{t_e} + \overline{t_i}$. We define that $\overline{t_e}$ is the average time on external processing. $\overline{t_i}$ is the average time on internal processing. Thus, the processing capability (C) becomes

$$C = \frac{n}{\overline{t_e} + \overline{t_i}} \tag{2}$$

Then, the size of the message buffer changes at a rate S is

$$S = R - C \tag{3}$$

We know that if $S > 0$, more and more message will be kept waiting in the waiting queue. Therefore, the size of the message buffer increases as long as $S > 0$, which could lead to two results. Firstly, the memory resources of the proxy will be depleted if there is no maximum limitation on the size of message buffer. Also the messages in the buffer have to be waiting for a longer time to get processed. Secondly, if there is a maximum limitation on the size of buffer, then the proxy will refuse to accept new SIP messages as long as the limitation is reached. Thus, any of these two alternatives result in a Denial of Service on the proxy. Hence, for system stability, we assume that overall S should be less or equal than 0: $S = R - C \le 0$. Thus,

$$R - \frac{n}{\overline{t_e} + \overline{t_i}} \le 0 \tag{4}$$

From the model, the stability of SIP proxy depends on four parameters: R, $\overline{t_e}$, $\overline{t_i}$ and n. Attackers can easily control the first three parameters. As long as an attacker can increase one or more of the three parameters, the steady-state of the proxy might be broken and a Denial of Service might happen. For example, attackers can flood the victim proxy with thousands of messages to increase R. However, a flooding attack can be easily detected e.g., by an Intrusion Detection System (IDS). It is abnormal and suspicious if a user constantly tries to setup a large number of calls in a given time interval. As an alternative, attackers also can send malformed SIP messages to the victim proxy. Then, the proxy needs to perform additional grammar checking so that $\overline{t_i}$ increases. Nevertheless, the caused delay time on $\overline{t_i}$ is relatively small [13].

Compared with the other two parameters, $\overline{t_e}$ can be more easily controlled by attackers. As we introduced before, $\overline{t_e}$ partly depends on the conditions of external hosts and can be out of control of the SIP proxy. The attackers can find some high-latency hosts on the Internet and craft SIP messages (attacking messages) so that the victim SIP proxy will contact them. The consequences are as follows.

1. Firstly, the working processes have to spend much time on these attacking messages when they perform external processing to contact high-latency hosts. Hence, the increase of $\overline{t_e}$ reduces the capability of the proxy.
2. Secondly, as it takes long time for working processes to handle attacking messages, the messages in the message buffer have to be waiting for a long time to get processed. As we mentioned in Sect. 2, after sending a message, a UA will only wait for its response until the timeout (32 seconds in default). In this way, even if these messages can be processed eventually, they probably are already expired.
3. Finally, considering the retransmission mechanism in SIP, a UA will resend the message up to 7 times if its response is not received within a given time period. Therefore, R increases as well resulting in a heavy load on the proxy. However, these retransmitted messages are from legal users and formed in a legal way.

For blocking attacks there is no need to flood the victim proxy and all attacking messages are well-formed. Since messages originated from legal users can sometimes cause a large $\overline{t_e}$ as well, it is difficult to distinguish between legal and illegal messages by simply observing $\overline{t_e}$. We explain how attackers can craft attacking messages to control $\overline{t_e}$ in the next section.

Certainly, the system administrator can increase n to fork more parallel working processes so that the proxy can handle more messages synchronously. However, each forked process consumes system resources. Too many processes can lead to the proxy being unstable.

5 Two attacking examples

In this section, we describe how attackers can craft attacking messages with high $\overline{t_e}$ to perform a DoS attack. We show two concrete examples. One takes advantages of external processing with DNS servers, and another exploits external processing with web servers.

5.1 Blocking attack using high-latency DNS servers

5.1.1 DNS usage in SIP

The Domain Name Service (DNS) [15] is one of the fundamental infrastructures for most current Internet applications available today, including web and email services. It is a globally distributed database, providing an essential domain name mapping service for Internet users. Also, the DNS plays a key role in SIP networks for the following reasons.

1. Similarly to the email addressing scheme, SIP VoIP users logically belong to their home domains. The format of their SIP identities (also called Uniform Resource Identifier (URI)) [16] is similar to the one of email addresses, including user name and domain name (e.g., sip:ge.zhang@kau.se). As shown in Fig. 2, the identities of both caller and callee contain domain names. A SIP proxy has to contact a DNS server to resolve these domain names to locate remote domains.
2. To interconnect the PSTN with a SIP network, ENUM [17] telephone number mapping is used. This allows the mapping of a PSTN telephone number to a valid SIP address.
3. SIP can utilize different transport layer protocols (e.g., UDP, TCP). To find its right contact server in regard to the used transport layer protocol, a SIP proxy will perform a DNS SRV request for the domain of the regarding SIP URI [18]. The response may contain one or more destination hosts that provide the required services.
4. In some implementations, the callee's proxy may issue a reversing DNS request for the caller's IP address to check whether this SIP message is really coming from the domain as announced in the message. The purpose is to counter SIP identity fraud [19].

5.1.2 Attacking method

Issuing a DNS request to DNS servers is classified as external processing since the SIP proxy cannot resolve domain names itself. The SIP proxy needs to fetch DNS mapping information of the destination domain from DNS servers. In this scenario, REI is DNS mapping information. As a result, the time on resolving domain name is beyond the SIP proxy's control. With a distributed database, the processing

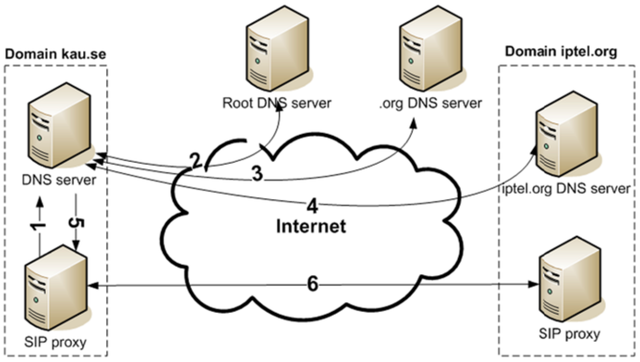
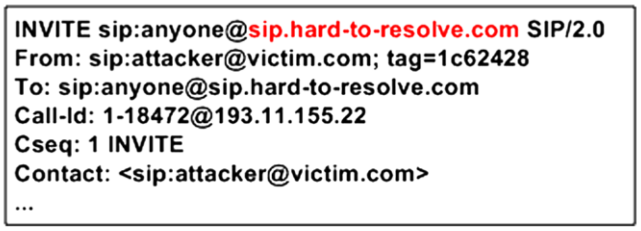


Fig. 4 Procedure of recursive DNS requests



```
INVITE sip:anyone@sip.hard-to-resolve.com SIP/2.0
From: sip:attacker@victim.com; tag=1c62428
To: sip:anyone@sip.hard-to-resolve.com
Call-Id: 1-18472@193.11.155.22
Cseq: 1 INVITE
Contact: <sip:attacker@victim.com>
...
```

Fig. 5 An example attack SIP message using a domain name

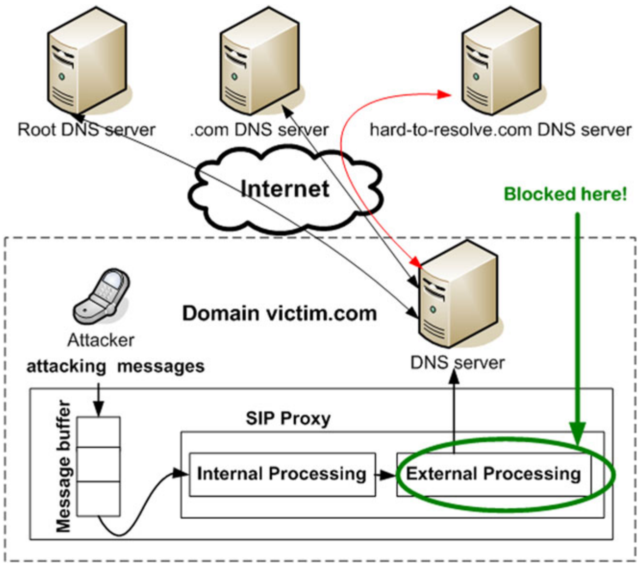


Fig. 6 The victim proxy will be blocked while processing attacking messages

time of DNS request sometimes can be quite long. Whenever a SIP proxy requests a domain name resolution to a DNS server, there are generally two cases to distinguish:

1. The DNS server knows the name mapping. The DNS might know the mapping because it is the authoritative name server for this domain. As such, all mappings for the domain are preconfigured for this domain server. The server might also know the name because it has resolved the same request previously. In this case, the mapping result is still stored in the DNS server's internal cache.
2. The DNS server does not know the name mapping. Thus, this DNS server has to perform recursive requests to other DNS servers that might be able to provide an answer. Finally, the DNS server will receive a response, either containing a valid result or an error message that no mapping is possible. In the former case, the mapping will be stored in the server's internal cache for a period. In most cases, the DNS server can also set a timeout for requests. If no answer is received from recursive request during this timeout, the domain name is considered irresolvable.

In Fig. 4, we show an example where a SIP proxy of the kau.se domain issues a DNS request to sip.iptel.org. The DNS server of kau.se performs a series of recursive requests until it finds the authoritative DNS server of this domain (iptel.org). Finally, this server replies with the IP address of sip.iptel.org to kau.se's DNS server. Only after getting the result, the SIP proxy of kau.se can forward the messages to the iptel.org domain.

The recursive DNS requests might be taking a long time. In step 4 of Fig. 4, the DNS server of kau.se has to contact the DNS server of iptel.org to get the final result. Thus, the query time can be long if the DNS server of iptel.org is poorly implemented or has poor network conditions.

In this way, an attacker can do a pretest by querying different domain names and observing their reply time. Then, the attacker can craft attacking messages containing those domains with high-latency DNS servers, which we defined as **hard-to-resolve domain names** in this article. The attacker can simply insert a SIP identity with a hard-to-resolve domain name in the request line. An example for such a crafted SIP message can be seen in Fig. 5. The highlighted part of the message can lead to proxy blocking. When a SIP proxy receives attacking messages, it has to spend additional time on domain name resolving (as shown in Fig. 6). Therefore, the SIP proxy is blocked due to the messages it has to process with high $\overline{t_e}$. In the real world, a domain name resolving can cost up to 15 seconds according to our tests.

5.2 Blocking attack using high-latency Web servers

5.2.1 Inter-domain authentication

Spammers frequently use faked SIP identities in order to be untraceable. However, there is no centralized database of user accounts for all domains. It is difficult for the callee's domain to authenticate the originator of a SIP message in an inter-domain context since the callee's domain may do not have the account information of the originator. In this way, spammers can easily send SIP messages with faked identities. To prevent such a identity fraud problem, a solution based on certificates for inter-domain authentication

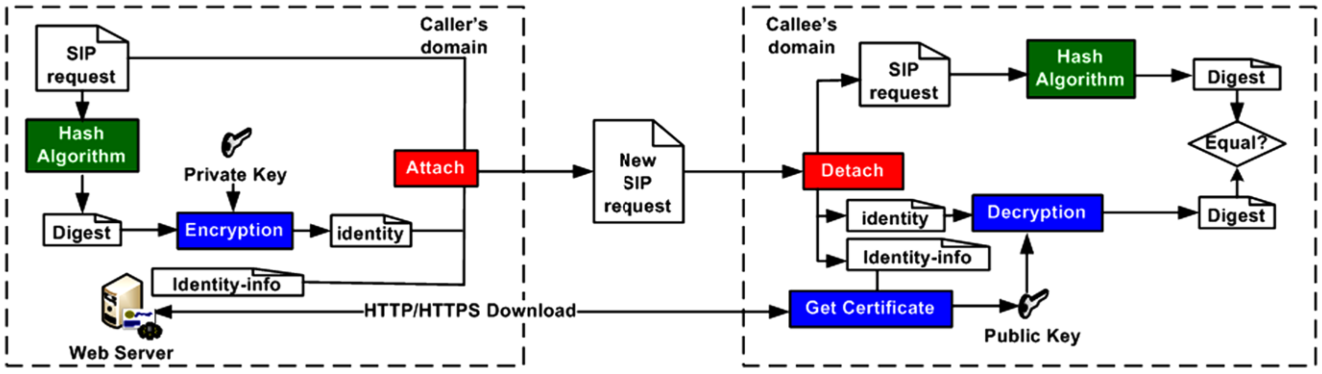


Fig. 7 The inter-domain authentication mechanism defined in RFC 4474

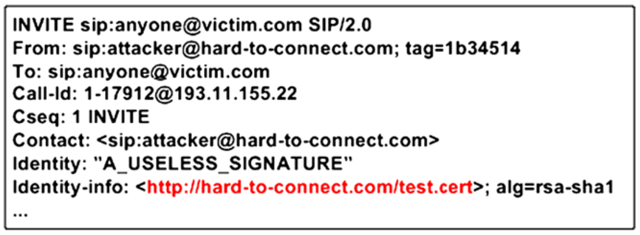
has been proposed in RFC 4474 [20]. Its purpose is to authenticate the originator of inter-domain SIP messages. The method is shown in Fig. 7. For each outgoing message to other domains, the SIP proxy in the caller's domain generates a hash digest for this message. The digest is signed by the caller's proxy with its private key. The generated signature is encoded in a new header field *Identity* added to the original SIP message. Furthermore, the SIP proxy attaches another new header field *Identity-info*, which contains the Uniform Resource Locator (URL) [21] where the certificate can be fetched from. Then, this SIP request is forwarded to the callee's domain. It is regulated that each certificate must be provided by its domain itself. That is, in a SIP message, the URL indicated in the *Identity-info* field and the originator domain indicated in the *From* field should be matched. For each incoming message from other domains, the SIP proxy in the callee's domain first downloads the certificate according to the URL given in the *Identity-info* field. The public key extracted from the certificate is used to decrypt the signature contained in the *Identity* field. Then, a hash digest will be recomputed for the request. The result is used to be compared to the newly generated hash digest. The proxy will continue to process the message only if the two values are equal. This mechanism is recommended to be deployed in future SIP services to prevent SPAM [22].

5.2.2 *Attacking method*

Downloading certificates from web servers on the Internet can be classified as external processing. The REI in this scenario are the certificates of the senders' domain. After receiving an inter-domain message, the proxy must authenticate the message beginning by downloading the certificate from the URL indicated in the *Identity-info* field. Whether the proxy will continue to process this message depends on the authentication result. If this message cannot be successfully verified or its certificate can not be downloaded, the proxy will not process this message further, but replies with an message with "authentication failed". Otherwise, the proxy will continue the processing of this message. Thus, the proxy cannot decide whether it should continue to process this message until it tried certificate downloading. During the time for certificate downloading, there are 4 steps to be performed, which are the following ones.

1. The proxy issues a DNS request for the URL indicated in the *Identity-info* field to get the IP address of the web server which provides the certificate.
2. The proxy tries to establish a HTTP or HTTPS connection with the web server.
3. The proxy tries to download the certificate from the location given in the *Identity-info* field by HTTP or HTTPS transmission.
4. The proxy closes the connection.

If step 1 or step 2 fails, the certificate is considered to be unable to download and the subsequent steps will not be executed. In this way, attackers can increase $\overline{t_e}$ by either using a high-latency DNS server to slow down step 1 or using a high-latency web server to delay step 2. Since we have already shown the attacks using high-latency DNS server before, we will focus more on the attacks using high-latency web server here. For example, if an attacker knows that a website "hard-to-connect.com" is high-latency, he can craft an attacking message pretending to be from "hard-to-connect.com", with its *Identity-info* field filled with "http://www.hard-to-connect.com/test.cer". An example of the attack message is shown in Fig. 8. The highlighted part in the message can result in the proxy to be blocked. Actually, there is no such a file called test.cer existing on this website. However, when the victim proxy receives this request, it can spend an enormous amount of time on connecting to "hard-to-connect.com". The procedure is illustrated in Fig. 9. After waiting for a long time, the proxy will give up processing this message because the certificate cannot be downloaded. However, the attacker does not need the attacking message processed, but only wants the proxy being blocked. In prac-



```
INVITE sip:anyone@victim.com SIP/2.0
From: sip:attacker@hard-to-connect.com; tag=1b34514
To: sip:anyone@victim.com
Call-Id: 1-17912@193.11.155.22
Cseq: 1 INVITE
Contact: <sip:attacker@hard-to-connect.com>
Identity: "A_USELESS_SIGNATURE"
Identity-info: <http://hard-to-connect.com/test.cert>; alg=rsa-sha1
...
```

Fig. 8 An example SIP attacking message using web server

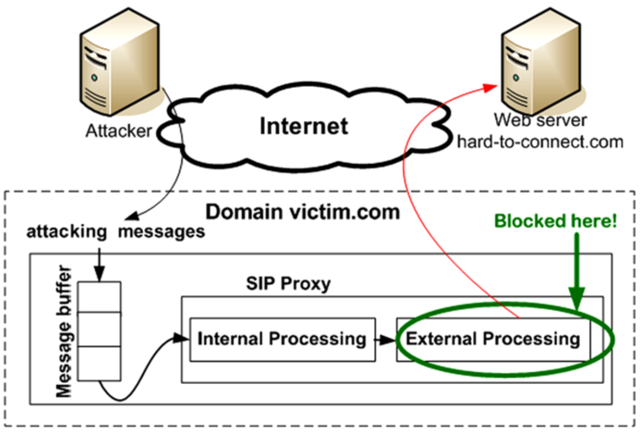


Fig. 9 The victim proxy will be blocked as long as it tries to connect to the high-latency web server

tice, the connection with a latency web server can cost up to 60 seconds according to our pretests.

5.3 Preliminary summary of blocking attacks

As it is heavily depending on existing Internet infrastructures, a SIP proxy has to interact with external servers to fetch REI for processing SIP messages. We define this type of processing as external processing. For instance, when a proxy receives a message targeting other SIP domains, the proxy has to contact DNS servers to resolve the domain name. Similarly, when a proxy receives a message originated from another SIP domain, the proxy needs to download a certificate from such a domain for authentication. Thus, a sophisticated attacker can craft attacking messages which deliberately make a proxy to contact external high-latency servers. After receiving these messages, a SIP proxy will spend a long time on them and is unable to process further legal messages during that time interval.

6 Experiments

In order to investigate the effectiveness of the blocking attacks in the real world, we firstly investigate the distribution of latency between hosts in the Internet. Next, we setup a testbed to perform a series of experiments, including the

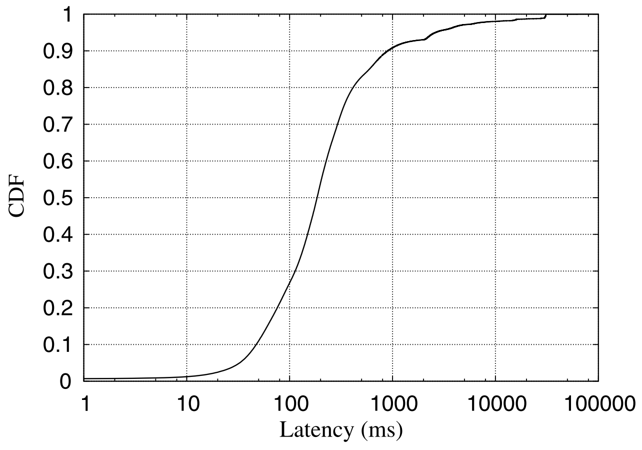


Fig. 10 Cumulative distribution of hosts delays according to the MIT king dataset (logarithmic scale)

blocking attacks by exploiting high-latency DNS server and web server, respectively.

6.1 Measurements of latency in the real world

To launch the blocking attacks in the real world, attackers have to collect a list of high-latency DNS servers or web servers in the Internet. The latency between the servers and the victim SIP proxy should be high enough to cause a large $\overline{t_e}$. But are these high-latency servers easy to find? And, how much latency can these servers achieve? To answer these questions, we used the MIT King dataset [23] as our measurement dataset. King [24] is a method to estimate the latency between two hosts in the Internet, by issuing recursive requests through their DNS servers. The MIT King dataset contains measurement of the latency between nearly 2000 randomly selected DNS servers including 97 million results. Then we took 1 ms as the interval and calculate the cumulative distribution based on the dataset, which is shown in Fig. 10. The result demonstrates that values for latencies are distributed from 100 milliseconds to 1 second in most cases. However, there is still 10% probability for a latency to be higher than 1 second and 2% probability for a latency to be higher than 10 seconds. As can be seen, it is possible to find high-latency servers in the Internet without much overhead. Therefore, in the following experiments, we will investigate the attacking impact by scaling the latency time from 1 second to 14 seconds.

6.2 Test bed

The testbed consists of five components.

1. A SIP domain called **Victim**: A Victim domain includes a SIP UA, a SIP proxy and a DNS server. The UA is used to generate SIP messages to simulate the legal users' traffic behavior. All SIP traffic to and from the UA is relayed

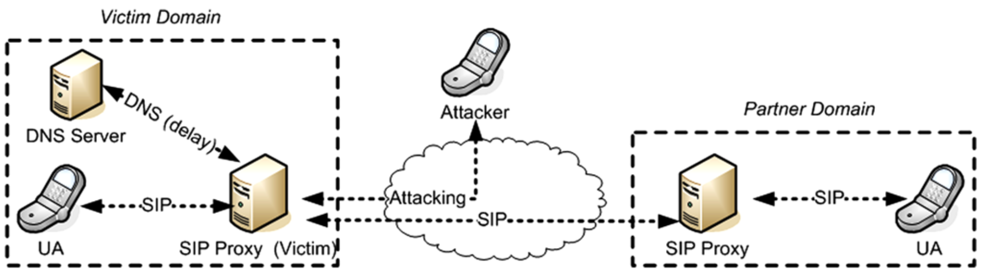


Fig. 11 The test bed for the attack using a high-latency DNS server

by the SIP proxy. This SIP proxy, the attacking target, is implemented according to RFC 3261 and RFC 4474. We configured the victim proxy to be quipped with n ($n = 4$ or $n = 16$) parallel process. These two configurations are most frequently used in practice.

2. A SIP domain called **Partner**: The Partner domain includes a SIP UA and a SIP proxy. It constantly receives SIP messages from the victim domain and sends replies. The partner domain is used to cooperate with the victim domain to build or tear down calls. The SIP proxy and UA in the partner domain will not be under attack during the test.
3. An attacking tool: The attacking tool constantly sends attacking SIP requests to the victim SIP proxy. The attacking tool can be configured to send messages with variable attacking rate r (INVITEs/s).
4. A DNS server: The SIP proxy in the victim domain can contact the DNS server for DNS request. The DNS server can be specially configured to delay some requests for d_{dns} seconds. The purpose of such a delay is to simulate the time consumption on recursive DNS requests to a high-latency DNS server. The measurement in Sect. 5.3 shows that the latency varies from 10 milliseconds to more than 10 seconds. In the following test, we select d_{dns} to range from 1 second to 14 seconds.
5. A web server: The SIP proxy in the victim domain can download certificates from this web server. Similar to the DNS server, this web server can be also configured to delay each response for d_{web} seconds. It is used to simulate the time consumption on network latency. The range of d_{web} is selected accordingly to d_{dns}.

The experiment variables are listed in Table 1.

Software The attacking tool, victim UA and partner UA are implemented using SIPp [25], an open source SIP traffic generation tool. The SIP proxies are realized by using SIP Express Router (SER) [26]. The web server is implemented based on mini_ httpd [27]. The DNS server is based on a modified dnsmasq [28].

Hardware The testbed of the victim domain with the DNS server is established on a Pentium 4 machine with 512 MB RAM running the Linux Ubuntu operating system, with fast

Table 1 Experiment variables

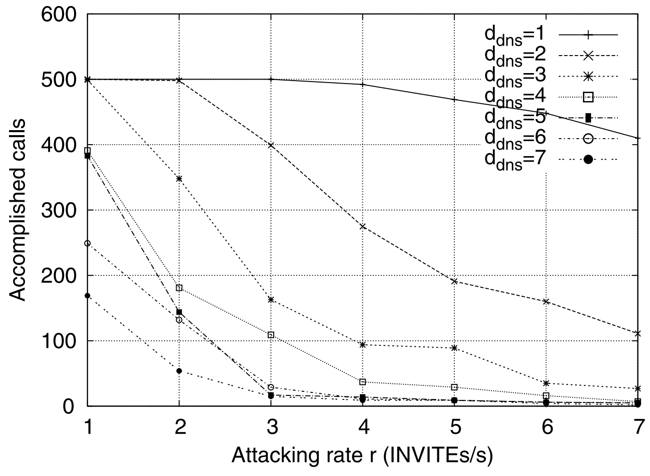
Variable description	Variable	Unit
The number of parallel processes of the victim proxy	n	null
The attacking rate of the attacking tool	r	INVITEs/s
The delay time on a DNS request for an attacking message	d_{dns}	s
The delay time on a connection to the web server for an attacking message	d_{web}	s

Internet access. The partner domain and the attacking tool with the web server are running on other two machines with the same configurations, respectively.

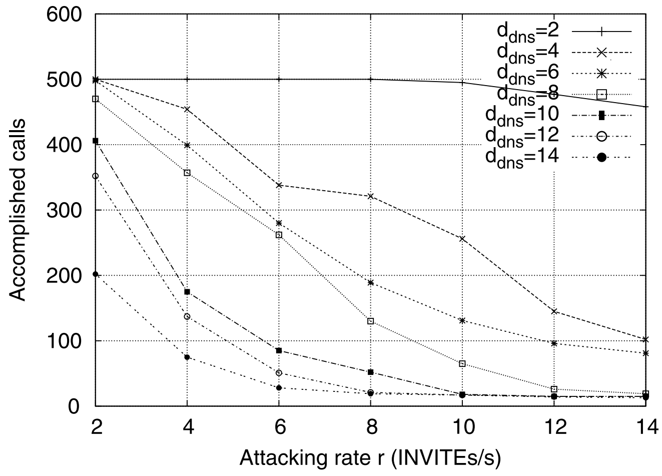
Legal user behavior We made the UA in the victim domain and the UA in the partner domain simulating legal SIP users by constantly building up and tearing down calls according to the 14 steps in Fig. 1. We define that one call has been successfully accomplished if all of these 14 steps were accomplished. In a pretest, we found that our test bed can work steadily at a rate of 50 calls/s without attacks. Then, we tried to accomplish 500 calls between the two UAs with a relatively slow rate: 10 calls/s, but under attack. We took 500 calls as the benchmark and observed how many calls can be accomplished when the victim proxy is under attack. The number of accomplished calls is used to indicate the throughput rate of the victim proxy.

6.3 Attack tests using a high-latency DNS server

In this scenario, we use four components based on the introduced test bed: the victim domain, the partner domain, the attacking tool and the DNS server. The configuration of this test bed is shown in Fig. 11. The attacker pretends to be a roaming user who constantly sends attacking messages to the victim proxy at a rate r. The messages indicate that the attacker wants to talk with a user located in a domain called "hard-to-resolve.com". Thus, the proxy will contact the DNS server to resolve the "hard-to-



(a) The victim proxy equipped with 4 processes (n=4)



(b) The victim proxy equipped with 16 processes (n=16)

Fig. 12 The result of blocking attacks using a high-latency DNS server: throughput rate is reduced as the attacking parameters d_{dns} and r increase

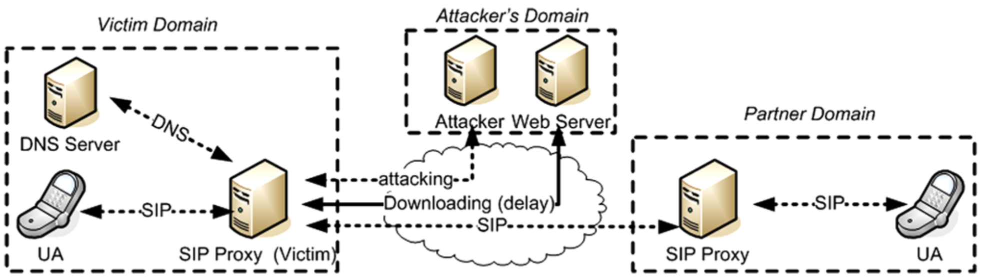


Fig. 13 The test bed for the attack using a high-latency web server

resolve.com" domain. The DNS server is configured to delay the response for d_{dns} on purpose when it receives DNS requests of "hard-to-resolve.com". The purpose of delay is to simulate the time needed for recursive DNS requests. The DNS server does not delay other requests except "hard-to-resolve.com". Meanwhile, the two legal UAs communicate with each other. We observed how many calls can be accomplished between them when the victim proxy is under attack.

The results are illustrated in Figs. 12(a) and (b). The pictures show that the number of accomplished calls are reduced as the attacking parameters d_{dns} and r increase. Therefore, it is easy for an attacker to mount DoS attacks. For example, setting parameter $r = 2$ and $d_{dns} = 4$, the attacker can basically reduce performance of the victim by half of its initial performance. The same attacking parameter configuration does not work with the victim proxy with $n = 16$. However, if the attacker just simply speeds up r to 10, still 50% of all calls can not be accomplished. The experiments show that the blocking attack can be easily realized by using a high-latency DNS server.

6.4 Attack tests using a high-latency Web server

In this scenario, we employ all five components of the introduced test bed: the victim domain, the partner domain, the attacking tool, the DNS server and the web server. Since the feasibility of the attack using high latency DNS servers has already been proved in our previous tests, we configure the delay time of the DNS server $d_{dns} \equiv 0$ in this test. Hence, readers should bear in mind that *the DNS server in this test does not delay any request any more*. The configuration of the testbed for this test is shown in Fig. 13. The attacker pretends to be a SIP proxy relaying inter-domain SIP messages to contact the victim proxy. The messages are actually attacking messages which indicate their certificates can be downloaded from the web server. The victim proxy needs to authenticate these messages and thus has to communicate with the web server for downloading the certificates. The web server has already been configured to delay every downloading request for d_{web} seconds. The purpose is to simulate the time needed for network latency on connections. Otherwise the setup is similar to the previous experiment.

The test results are shown in Figs. 14(a) and (b). The pictures show that the number of accomplished calls are reduced as the attacking parameters d_{web} and r increase. Therefore, it is also possible to mount DoS attacks by using high-latency web servers. However, the impact of these attacks are lower compared to the impact of the previously discussed attacks in Sect. 5.2. The reason is that for the attacks exploiting high-latency DNS servers, the victim proxy con-

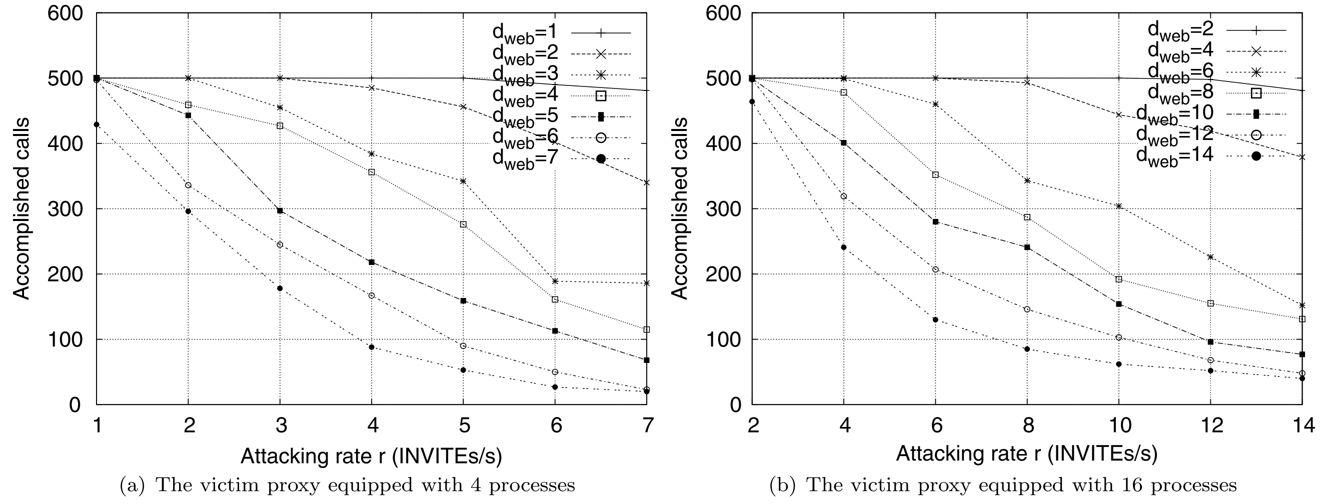


Fig. 14 The result of blocking attacks using a high-latency web server: the throughput rate is reduced as the attacking parameters d_{web} and r increase

tacts the DNS server twice for each attacking message (the first time for a SRV request, and since no result is received, the proxy will continuously issue another A request,[2] so it will be delayed twice for each attacking message. But for the attack exploiting high-latency web servers, the proxy contacts the web server only once for each message. Nevertheless, the result still shows the impact of the attack using web server is large enough to launch DoS attacks on SIP proxies.

7 Defence solutions

In this section, we present and discuss several defence solutions against blocking attacks. The solutions are classified in two categories: (1) Proxy-based; (2) Cache-based.

7.1 Proxy-based solutions

Proxy-based solutions aim at minimizing the impact of blocking attacks by increasing the CPU utilization of the victim proxies. The solutions include increasing n, restricting t_e and adopting asynchronous processing.

Solution 1: increase n Observed from the experiments, blocking attacks work most effectively if the proxy utilizes few parallel processes. For example, as shown in Fig. 12(a) and (b), when $n = 4$, the attack with parameter $r = 6$ and $d_{dns} = 3$ can almost achieve a full DoS. When $n = 16$, the same attack can only reduce the performance of the proxy by around 40%. This shows that an increase of n is helpful to enhance the performance during attacks. However, we also can observe from the experiments that the attacks will still work as long as the attacker increases the attacking rate r, which can be easily achieved by a Distributed Denial of Service (DDoS). On the other hand, n should be confined within a reasonable range, as each process of the proxy consumes CPU and memory resources. Configured with too many processes, the proxy will take the risk to be overloaded or even crash.

Solution 2: setup a limitation on t_e It has been shown that the more time spent on external processing t_e, the more performance of SIP proxies is reduced during attacks. In this way, the system administrator can arbitrary set a timeout of t_e for SIP proxies, (e.g., 5 seconds). Thus, for a message whose external processing has already taken 5 seconds and is still going on, the proxy stops waiting and cancels the processing for this message. However, the proxy can still be blocked at least for the timeout during attacks. Another problem is that many messages from legal users might take a long t_e as well. Simply setting up a timeout for t_e does not solve the problem completely and may cause a high false positive rate annoying legal users.

Solution 3: asynchronous processing The easiest way to implement external processing is to use some provided library functions (e.g., `gethostbyname()` for DNS requests and `curl_ easy_perform()` for downloading certificates). However, these functions were designed without taking blocking attacks into account. As a result, the SIP proxy cannot do anything besides waiting during external processing. A solution is to re-implement these functions

[2]A SIP domain may own several proxies which support different transportation protocols. For DNS resolving, a SIP proxy will issue a SRV request to find out the corresponding transportation protocol supported by the target domain, and then issue an A request for the selected proxy. However, if the SRV request fails, the proxy will issue the A request of the target domain directly [18].

using an asynchronous processing model: after issuing an external processing request, the proxy would not wait until an answer for this request is received. Instead, it puts the request into an event queue, saves the transaction data, sets current external processing on hold and continues handling subsequent requests. When a reply for a previous external processing arrives, the suspending transaction is scheduled to continue. In this way, the external processing will not be blocked. Unfortunately, however, since all the states and SIP messages of unfinished external processing have to be saved, the implementation complexity and memory requirements increase considerably. In previous tests [4], we found that asynchronous processing easily leads to the depletion of memory of SIP proxies during attacks.

7.2 Cache-based solutions

As mentioned before, the purpose of external processing is to get REI for a message. Generally speaking, the same REI (e.g., DNS mapping or certificate) can be reused for messages coming from or targeting the same domain. To avoid wasting time on external processing, a SIP proxy can cache fetched REI locally. Thus, there is no need to interact with external servers all the time if its REI can be found locally. The philosophy behind cache-based solutions is to transfer tasks from external processing to internal processing. All the following proposed solutions are based on cache. To explain this section better, we firstly define four kinds of SIP messages from a proxy's point of view. Then, the cache-based solutions are represented.

1. α **message:** An α message originates from a legal user. The proxy has successfully fetched the REI of this message before. This may has happened because the proxy previously processed another message with the same REI that is needed. For example, consider two messages targeting at two different callees located in the same domain, the REI (DNS mapping) for these two messages are the same. Furthermore, for α messages, their REI is still saved in the local cache on proxy.
2. β **message:** A β message originates from a legal user as well. The proxy has also fetched the REI of this message before. However, contrary to an α message, the REI is **not** saved in the local cache. This may be due to two reasons: Firstly, there is a maximum limitation on the size of a cache. So some cached items will be erased to make room for new items when the size limitation is reached. Secondly, cached REI (both the DNS records and certificates) is assigned with a Time-To-Live (TTL) timer by their owner. Cached information will be removed from the cache when the TTL is expired. The cache refreshing is necessary in order to not only keep the proxy with updated information of remote hosts, but also to prevent cache poisoning attacks [29].

Table 2 A taxonomy of SIP messages from a proxy's perspective

Type	Sender	Its REI has been fetched before	Its REI is cached
α	legal user	Yes	Yes
β	legal user	Yes	No
γ	legal user	No	No
δ	attacker	No	No

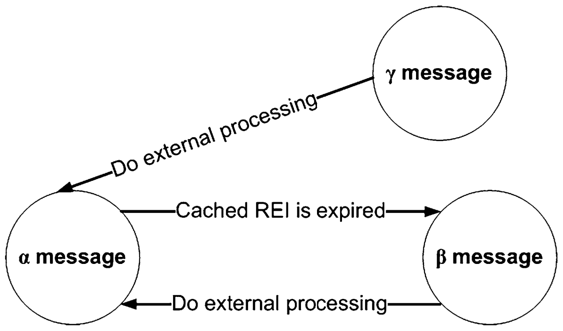


Fig. 15 State transitions between α, β and γ messages

3. γ **message:** A γ message is sent from a legal user. The proxy has not fetched the REI of this message before. As a result, its REI is not cached, either.
4. δ **message:** A δ message is generated by an attacker. To maximize the impact of attack, an attacker can randomize domain names or URLs in the message with automation tools to bypass the caching (e.g., adding random hosts names to the left side of a domain). Thus, for each δ message, it is unlikely that the REI has already been fetched for another previous message. In consequence, its REI is not cached as well.

A summary of these four message types is shown in Table 2. Furthermore, the same legal message can be classified as α, β or γ message at different times. The state transitions are shown in Fig. 15. If a proxy does external processing for a γ message, the REI of this γ message is cached. Then, if the proxy receives the same message another time, it becomes an α message. After its cached information has expired, the same message will be a β message. Certainly the message can become an α message again if the proxy reissues the external process. An ideal solution would process all messages from legal users including α, β, and γ messages and, while δ messages would be filtered by the proxy.

Solution 4: simple deployed caches The local cache consists of both a **positive cache** (for such a domain with fetched valid information) and a **negative cache** (indicates that it is unlikely to get a result for such a domain). Deploying a cache can accelerate processing for the messages with their REI cached. However, given that an attacker generates

```
Input: $n$: the number of parallel working processes
if $number\_of\_blocked\_processes < n - 1$ then
    if CheckCache() = $NoResult$ then
        $number\_of\_blocked\_processes{+}{+}$;
        ExternalProcessing();
        $number\_of\_blocked\_processes{-}{-}$;
        UpdateCache();
    end
    else
        GetFromCache();
    end
end
else if $number\_of\_blocked\_processes \geq n - 1$ then
    if CheckCache() = $NoResult$ then
        Reply($System\ Busy$);
    end
    else
        GetFromCache();
    end
end
```

Algorithm 1: Unblocking cache

randomized attack messages (δ message), it is unlikely for a proxy to find the REI from local cache. A victim proxy can still be blocked by δ messages. Thus, simply deploying caches cannot eliminate the impact of blocking attacks.

Solution 5: unblocking cache This solution uses the cache based on an unblocking mechanism, so called unblocking cache. We implemented an unblocking cache as a countermeasure against DNS blocking attack in our previous work [4]. Given a proxy with n parallel working processes, we define one of them as an **emergency process**. As long as $n - 1$ processes are blocked by external processing, the emergency process will refuse to do external processing any more. Instead, it will only check the result in its local cache and reply with an error message (e.g., 5xx server error) if its external information cannot be found in the cache. The algorithm of the unblocking cache is shown in Algorithm 1. The benefit of using an unblocking cache is two-fold: firstly, when a proxy is under attack, the emergency process can at least handle α messages without being blocked. Moreover, since the emergency process refuses to do external processing at all when the proxy is under attack, the throughput rate of it should not be much reduced. As a result, it is helpful to clear up the message buffer in time and then accept more fresh messages into the message buffer.

The result of our past experiments showed that this solution is more effective than previous ones to counteract blocking attacks. However, the proxy still cannot process β messages during attacks even with the unblocking cache solution. The reason is that this solution refuses to do external process at all as long as $n - 1$ processes are blocked. This may cause problem in practice: for a long lasting blocking attack, all cached information in a proxy may be removed as soon as their TTL expires. Thus, the emergency process will refuse to process any message as the local cache becomes empty. In this way, the impact of denial of service still exists. To address this problem, we have to improve the unblocking cache so it can process β messages during attacks as well.

Solution 6: priority mechanism The priority mechanism is an improvement of the unblocking cache. The purpose of the priority mechanism is to enable processing of both α and β messages during an attack. The priority mechanism works in a similar manner to the unblocking cache. The difference between them is that the priority mechanism employs a **priority list** instead of the content of cache to decide whether a message can be processed. The priority list includes a list of domain names or URLs, from which the proxy has previously successfully fetched REI. It only contains domain names or URLs without their detailed external information. Further, it is constantly updated after each successful external processing. Since the priority list **only** contains domain names or URLs, it does not consume much memory resources and it is not vulnerable to cache poisoning. Therefore, more entities can be managed with a priority list than with a local cache. The algorithm of the unblocking cache is shown in Algorithm 2. When $n - 1$ processes are blocked, the emergency process enables to handle both α messages and β messages in this way: After receiving a message, the emergency process first checks the local cache to determine whether it is an α message. If it is an α message, the emergency process gets REI from cache and processes this message. If it is not an α message, the emergency process will check the priority list to find out whether it is a β message. If it is a β message, the proxy will do external processing for this message and cache the result. Otherwise, for other message types (γ message and δ message), the emergency process will give up processing. In this way, this solution increases the trusted circle and enables the proxy to do external processing for β messages.

One may argue that it may take some time for an emergency process to do external processing for a β message. However, since a β message has been successfully processed before, we assume that the message can be trusted and the external processing of it will not take too long. Furthermore, its REI can be cached locally again after successful external processing. This means that the same external processing will not be repeated continuously. Therefore, we assume that this drawback is acceptable.

7.3 Solution comparison

We conducted an experiment to compare the proposed solutions. Since we already proved that proxy-based solutions are insufficient [4], we only apply the cache-based solution prototypes on a victim proxy to observe and compare the effectiveness of the three solutions. The setup of our test bed for these experiments is shown in Fig. 16. The attacking tool generates δ messages to the victim proxy and the DNS server will delay the requests for δ messages. Changing from previous test beds, the partner domain is no longer used in this test. Instead, we selected 15 SIP providers from the real world. Each SIP provider has its own domain. The reason why we change the test bed elements is that we need different domain names in the real world to test deployed caches. To simplify the test, the legal UA in this experiment does not build calls as in previous tests. Instead, our SIP UA only sends OPTIONS[3] messages to these 15 SIP providers through the victim proxy. The total amount of sent OPTIONS messages is 500. We have tested that the UA will receive all 500 "200 OK" message replies for OPTIONS messages if there is no attack. We further observe how many "200 OK" messages will be received by the UA when the victim proxy is applying the different solutions and is under attack. We start the UA first followed by the attacking tool in order to let the priority list and the local cache get the information of these domains of SIP providers. Each repeated experiment lasts for 1000 seconds.

```
Input: $n$: the number of parallel working processes
if $number\_of\_blocked\_processes < n - 1$ then
    if CheckCache() = $NoResult$ then
        $number\_of\_blocked\_processes{+}{+}$;
        ExternalProcessing();
        $number\_of\_blocked\_processes{-}{-}$;
        UpdatePriorityList();
        UpdateCache();
    end
    else
        GetFromCache();
    end
end
else if $number\_of\_blocked\_processes \geq n - 1$ then
    if CheckCache() = $NoResult$ then
        if CheckPriorityList() = $NoResult$
        then
            Reply($System\ Busy$);
        end
        else
            $number\_of\_blocked\_processes{+}{+}$;
            ExternalProcessing();
            $number\_of\_blocked\_processes{-}{-}$;
            UpdatePriorityList();
            UpdateCache();
        end
    end
    else
        GetFromCache();
    end
end
```
Algorithm 2: Priority mechanism

Figure 17 plots the testing result: one point for every tenth received "200 OK" messages over elapsed time. In this test, the best performance is achieved by the priority mechanism: the UA received all 500 "200 OK" messages. Next, after deploying the unblocking cache solution, nearly 420 "200 OK" messages were received by the UA. During the first 300 seconds, the unblocking cache gives the same performance as the priority mechanism. However its throughput is slightly reduced 300 seconds later as the distances between each points increase. This is caused by the refreshing of some cache entries with expired TTL. Since some REI is not in cache any more and the emergency process refuses to do external processing, some messages can not be processed any more. That is why the proxy missed some legal messages. Nevertheless, the deployed priority mechanism can process all the messages, since the emergency process is allowed to do external processing for β messages, despite of whether their REI are still in cache or not. The experiment also proved that simply applying a basic cache is insufficient for defending against the attacks: the proxy only worked for the first 180 seconds and less than 20% "200 OK" messages were replied.

[3] The SIP OPTIONS method allows a UA to query the capabilities of a SIP proxy. The proxy should then reply with a "200 OK" message containing corresponding information.

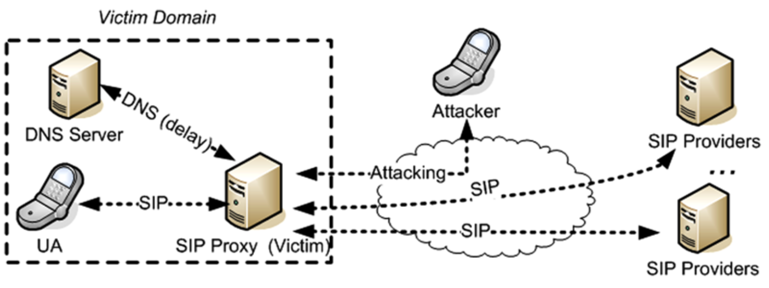


Fig. 16 Test bed used to compare the effectiveness of different solutions

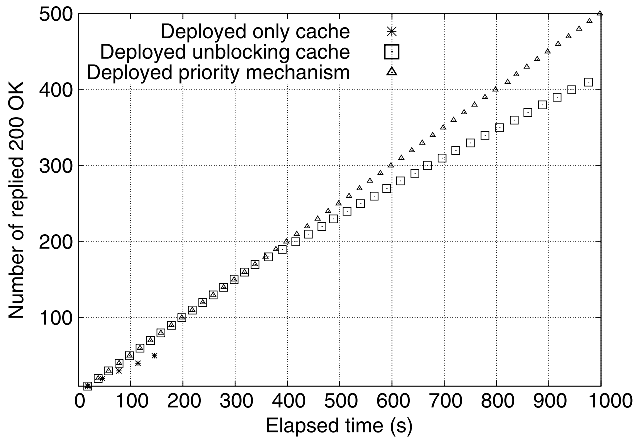


Fig. 17 Test results of the cache-based solutions comparison: Each point in the figure indicates every tenth received "200 OK" message over time

8 Conclusion

Since SIP proxies have to frequently contact external servers to fetch related information for processing, the time consumption on interactions with external servers is beyond the control of SIP proxies. This vulnerability increase extra processing time and can be exploited by attackers to reduce the throughput of SIP proxies for mounting a DoS attack on SIP proxies. We define this attack as a blocking attack. Our experiments have proved that the blocking attack can be launched with a low attacking rate by taking advantage of high-latency DNS and web servers. It can be predicted that more Internet-based infrastructures can be deployed for such blocking attacks in the future with increasing SIP applications. Hence SIP designers and implementors should carefully take this vulnerability into account. Finally, we compared several defending solutions and proposed a priority mechanism to enhance the performance of service during the attack.

References

1. Voice over IP Security Alliance (VOIPSA). http://www.voipsa.org/. Accessed at 16 September 2008.
2. Geneiatakis, D., Dagiuklas, T., Kambourakis, G., Lambrinoudakis, C., Gritzalis, S., Ehlert, S., & Sisalem, D. (2006). Survey of security vulnerabilities in session initiation protocol. *IEEE Communications Surveys & Tutorials*, *8*(3), 68–81.
3. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., & Schooler, E. (2002). SIP: Session initiation protocol. RFC 3261.
4. Zhang, G., Ehlert, S., Magedanz, T., & Sisalem, D. (2007). Denial of service attack and prevention on SIP VoIP infrastructures using DNS flooding. In *IPTComm '07: Proceedings of the 1st international conference on principles, systems and applications of IP telecommunications* (pp. 57–66). New York, NY, USA, July 2007. ACM.
5. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). Hypertext transfer protocol—HTTP/1.1. RFC 2616.
6. Handley, M., & Jacobson, V. (1998). SDP: Session description protocol. RFC 2327.
7. Sisalem, D., Kuthan, J., & Ehlert, S. (2006). Denial of service attacks targeting a SIP VoIP infrastructure: attack scenarios and prevention mechanisms. *IEEE Network*, *20*(5), 26–31.
8. Sengar, H., Wijesekera, D., Wang, H., & Jajodia, S. (2006). Fast detection of denial of service attacks on IP telephone. In *14th IEEE international workshop on quality of service*, New Haven, USA, June 2006. IEEE.
9. Chen, E. Y. (2006). Detecting DoS attacks on SIP system. In *1st IEEE workshop on VoIP management and security*, Vancouver, Canada, April 2006. IEEE.
10. Sengar, H., Wijesekera, D., Wang, H., & Jajodia, S. (2006). VoIP intrusion detection through interacting protocol state machines. In *DSN '06: the international conference on dependable systems and networks*, June.
11. Ehlert, S., Wang, C., Magedanz, T., & Sisalem, D. (2008). Specification-based denial-of-service detection for SIP voice-over-IP networks. In *3rd international conference on Internet monitoring and protection*, Bucharest, Hungary, July 2008. IEEE.
12. Conner, W., & Nahrstedt, K. (2008). Protecting SIP proxy servers from ringing-based denial-of-service attacks. In *The tenth IEEE international symposium on multimedia (ISM)*, Berkeley, USA, December 2008. IEEE.
13. Geneiatakis, D., Kambourakis, G., Lambrinoudakis, C., Dagiuklas, T., & Gritzalis, S. (2007). A framework for protecting a SIP-based infrastructure against malformed message attacks. *Computer Networks*, *51*(10), 2580–2593.
14. Fei, A., Pei, G., Liu, R., & Zhang, L. (1998). Measurements on delay and hop-count of the Internet. In *IEEE GLOBECOM'98—Internet mini-conference*, Sydney, Australia, November 1998. IEEE.
15. Mockapetris, P. V. (1987). Domain names—implementation and specification. RFC 1035.
16. Berners-Lee, T., Fielding, R., & Masinter, L. (2005). Uniform resource identifier (URI): Generic syntax. RFC 3986.
17. Faltstrom, P., & Mealling, M. (2004). The e.164 to uniform resource identifiers (URI) dynamic delegation discovery system (DDDS) application (ENUM). RFC 3761.
18. Rosenberg, J., & Schulzrinne, H. (2002). Session initiation protocol (SIP): locating SIP servers. RFC 3263.
19. Nassar, M., State, R., & Festor, O. (2007). VoIP honeypot architecture. In *IEEE international symposium on integrated network management*, Munich, Germany, May 2007. IEEE.
20. Peterson, J., & Jennings, C. (2006). Enhancements for authenticated identity management in the session initiation protocol (SIP). RFC 4474.
21. Berners-Lee, T., Masinter, L., & McCahill, M. (1994). Uniform resource locators (URL). RFC 1738.
22. Rosenberg, J., & Jennings, C. (2008). The session initiation protocol (SIP) and spam. RFC 5039.
23. The MIT "king" dataset: http://pdos.csail.mit.edu/p2psim/kingdata/. Accessed 16 January 2009.
24. Gummadi, K. P., Saroiu, S., & Gribble, S. D. (2002). King: estimating latency between arbitrary Internet end hosts. In *IMW'02: proceedings of the 2nd ACM SIGCOMM workshop on Internet measurement* (pp. 5–18). New York, NY, USA. ACM.
25. SIPp: http://sipp.sourceforge.net/. Accessed 16 September 2008.
26. Express Router, S. I. P. (SER): http://www.iptel.org. Accessed 16 September 2008.
27. Minihttpd: http://www.acme.com/software/mini_httpd/. Accessed 16 September 2008.
28. Dnsmasq: http://www.thekelleys.org.uk/dnsmasq/doc.html. Accessed 16 September 2008.

29. Stewart, J. (2003). *DNS cache poisoning—the next generation* (Technical report). http://www.lurhq.com/dnscache.pdf. Accessed 4 November 2008.

Ge Zhang received the B.S. degree in computer science from Anhui University of Technology, China in 2003 and the M.S. degree in computer science from Blekinge Institute of Technology, Sweden in 2007 respectively. He also did internship as a research student at Fraunhofer Institute FOKUS, Germany from 2006 to 2007. Currently, he is pursuing his Ph.D. degree at Karlstad University, Sweden. His research interests focus on VoIP applications, IMS, network security and privacy.

Simone Fischer-Hübner has been a Full Professor at the Computer Science Department of Karlstad University, Sweden, since June 2000, where she is the head of the PriSec (Privacy & Security) research group. She received her Doctoral (1992) and Habilitation (1999) Degrees in Computer Science from Hamburg University/Germany. Her research interests include IT-security and privacy-enhancing technologies. She was a research assistant/assistant professor at Hamburg University (1988–2000) and a Guest Professor at the Copenhagen Business School (1994–1995) and at Stockholm University/Royal Institute of Technology (1998–1999). She is the vice chairperson of IFIP (International Federation for Information Processing) Working Group 11.6 on "Identity Management" and coordinator of the Swedish IT Secure Network for Ph.D. students.

Sven Ehlert is a senior researcher at Fraunhofer Institute FOKUS in Germany. His research interests are SIP and VoIP/NGN communication networks, security applications, and Intrusion Detection Systems. He has lead several multinational VoIP/NGN security-related research projects including denial-of-service detection and VoIP spam detection. He graduated from Technische Universität Berlin with his major in communication protocols in computer science.