

# DENIAL OF SERVICE ATTACKS AND SIP INFRASTRUCTURE

## *Attack Scenarios and Prevention Mechanisms*

Dorgham Sisalem, Jiri Kuthan

*Fraunhofer Fokus, Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany*

*sisalem@fokus.fhg.de, kuthan@fokus.fhg.de*

Günter Schäfer

*Technical University of Berlin*

*schäfer@tkn.tu-berlin.de*

**Abstract:** In this paper we address the issue of denial of service (DoS) attacks targeting the hardware and software of voice over IP servers or by misusing specific signaling protocol features. As a signaling protocol we investigate here the session initiation protocol (SIP). In this context we mainly identify attacks based on exhaustion of the memory of VoIP servers, attacks on the CPU or by causing excessive communication with external servers such as DNS or authentication servers. We address two kinds of attacks: wanted attacks caused by malicious users and unwanted attacks caused by network misconfigurations, broken implementations or any other unknowledgeable technology use.

A major conclusion of the work is the knowledge that SIP provides a wide range of features that can be used to mount DoS attacks. Discovering these attacks is inherently difficult, as is the case with DoS attacks on other IP components. However, with adequate server design, efficient implementation and appropriate hardware the effects of a large portion of attacks can be reduced. Besides the server implementation and hardware we present different optimizations that reduce the need for contacting DNS servers using caches, policies and extensions to the SIP messages. Further, to reduce the risk of being attacked we describe issues of message monitoring and filtering as well as authentication approaches for different kinds of users.

## 1 INTRODUCTION

Security threats are considered minimal in current circuit switched networks. This is achieved by using a closed networking environment dedicated to a single application (namely voice). In an open environment such as the Internet, mounting an attack on a telephony server is, however, much simpler. This due to the fact that VoIP services are based on standardized and open technologies (i.e. SIP, H.323, MEGACO) using servers reachable through the Internet, implemented in software and provided often over general purpose computing hardware. Therefore, such services can suffer from similar security threats as HTTP-based services. Instead of generating thousands of costly voice calls, the attacker can easily send thousands of VoIP invitations in a similar manner to attacks on Web servers. These attacks are simple to mount and with a flat rate Internet access are also cheap.

Besides launching brute force attacks by generating a large number of useless VoIP calls, attackers can use certain features of the used VoIP protocol to incur higher loads at the servers. This might involve issuing requests that must be authenticated, require database lookups by the VoIP servers or cause an overhead at the servers in terms of saved state information or incurred

calculations. Further, the VoIP infrastructure can be corrupted by launching DoS attacks on components used by the VoIP infrastructure or the protocols and layers on top of which the VoIP infrastructure is based such as routing protocols or TCP.

The session initiation protocol (SIP) [7] is ever more establishing itself as the de-facto standard for VoIP services in the Internet and next generation networks. Therefore, this paper is dedicated to investigating possibilities of launching denial of service attacks on SIP servers and ways for preventing and reducing the effects of such attacks.

Sec. 2 presents some background information regarding denial of service attacks in general. In Sec. 3 and Sec. 4 we describe the resources and functionalities of a SIP server that can be targeted by an attacker in order to incur an overload situation at the server and reduce its availability. We then move to explain methodology of our experimental infrastructure which was used to gain estimates of performance values in Sec. 5. We then review attacks on memory, CPU and bandwidth as well as counter measures in Sec. 6. In Sec. 7 a list of operational guidelines for deploying secure SIP services is provided. The paper is then concluded in Sec 8.

## 2 BACKGROUND

### 2.1 Overview of DoS

Denial of Service (DoS) attacks aim at denying or degrading a legitimate user's access to a service or network resource, or at bringing down the servers offering such services. In the last several years DoS attacks have increasingly become a major problem of computer security. Internet Denial-of-Service attacks have increased in frequency, severity and sophistication [3]. Between the years of 1989 and 1995, the number of such attacks reported to the Computer Emergency Response Team (CERT) increased by 50% per year [4]. According to a 1999 CSI/FBI survey report 32% of respondents detected DoS attacks directed against them [2]. To make things worse, reports in the last few years [1] indicate that attackers have developed tools to coordinate distributed attacks from many separate sites, which is also known as Distributed Denial of Service (DDoS) attack.

In communication networks these attacking techniques can be applied to protocol processing functions at different layers of the protocol architecture. That is attacks can be launched on the network, transport or application layers. From a high level point of view DoS attacks can be classified into the two categories *resource destruction* and *resource allocation*. In a more detailed examination the following DoS attacking techniques can be identified:

- **Resource destruction:** This is achieved by disabling the system completely or controlling it in order to launch another kind of attacks. This could be achieved by
  - hacking into a system and thereby enabling an attacker to control the system,
  - making use of implementation weaknesses such as buffer overrun, or
  - deviation from proper protocol execution.

Most of these attacks can be prevented with a combination of good system management, software engineering and monitoring. However, defending against attacking techniques such as *protocol deviation* requires dedicated analysis for specific communication protocols and integration of this analysis into intrusion detection tools. Protocol deviation attacks usually exploit vulnerabilities of some operating system (OS) implementations concerning the network part of the OS, e.g. the TCP/IP

implementation, and can lead to system crashes by generating only a few or even one single “killer” packet. These attacks are sometimes referred to as “*nuke attacks*”. Defending against such attacks requires detailed knowledge of the features of the protocols and their implementation, monitoring of the traffic carried over those protocols and trying to detect the origin of the attacker.

- **Resource depletion:** With this attack a system is overloaded with a high amount of processing and computation of requests generated by the attacker, which result in making the system unavailable for requests from other users. Such attacks can be realized by causing expensive computations or storage of state information at the attacked system. This could be achieved for example by generating a lot of requests and not completing them. Another approach for such attacks is to generate a high number of useless requests leading to an overload situation of the attacked system.

Further, to increase the effects of an attack and reduce the resources needed by the attacker for launching this attack, the concept of distributed denial of service attacks (DDoS) is often used. In case of DDoS attackers use of the joint power of multiple systems for launching an attack. Therefore, even servers with high amount of resources and high bandwidth connections are vulnerable to such attacks. In addition to the attackers and the victims, a DDoS network consists of so called master- and slave-systems. The slave systems are the actual offenders. They are not controlled directly by the attacker, but by the master systems.

A fair amount of research and development work has already been done in the area of detecting attacks [6], and tracing the attackers back [5]. However, this work was mainly dedicated to popular protocols such as TCP and HTTP. Close to no work was done in the area of detection and prevention of attacks on VoIP systems in general and SIP in specific. While some of the experience gained from the work dedicated to current web and communication systems will be surely reused for the protection of VoIP systems, further work needs to be dedicated in examining the specific possibilities for launching DoS and DDoS attacks on SIP servers and other components of VoIP infrastructures such as databases, gateways and application servers.

## 2.2 Overview on SIP

The most important SIP operation is that of inviting new participants to a call. To achieve this functionality we can distinguish different SIP entities:

- **Proxy:** A proxy server receives a request and then forwards it towards the current location of the callee -either directly to the callee or to another server that might be better informed about the actual location of the callee.
- **Redirect:** A redirect server receives a request and informs the caller about the next hop server. The caller then contacts the next hop server directly.
- **User Agent:** A logical entity in the terminal equipment that is responsible for generating and terminating SIP requests.
- **Registrar:** To assist SIP entities in locating the requested communication partners SIP supports a further server type called register server. The register server is mainly thought to be a database containing locations as well as user preferences as indicated by the user agents.

In SIP, a user is identified through a SIP URI in the form of *user@domain*. This address can be resolved to a SIP proxy that is responsible for the user's domain. To identify the actual location of the user in terms of an IP address, the user needs to register his IP address at the SIP registrar responsible for his domain. Thereby when inviting a user, the caller sends his invitation to the SIP proxy responsible for the user's domain, which checks in the registrar's database the location of the user and forwards the invitation to the callee. The callee can either accept or reject the invitation. The session initiation is then finalized by having the caller acknowledging the reception of the callee's answer. During this message exchange, the caller and callee exchange the addresses at which they would like to receive the media and what kind of media they can accept. After finishing the session establishment, the end systems can exchange data directly without the involvement of the SIP proxy.

For authenticating a user SIP uses the digest authentication mechanisms, which is based on a challenge/reply approach.

In the following we describe some headers included in the SIP messages and that can be misused for launching DoS attacks:

- **Request-URI:** Indicates the destination the request is being sent

- **Route:** Determines the route a request should take. After receiving a request with such a header, the proxy is supposed to forward the message to the address indicated in this header
- **Contact:** Indicated the exact address of a user agent. Proxies might use this header as entry for a location information cache that could be used to speed up future searches.
- **VIA:** The VIA header indicates the path taken by a request so far.

## 3 EXPLOITABLE RESOURCES

Majority of DoS attacks is based on exhausting some of server's resources and causing server not to operate properly due to lack of resource. With SIP servers, there are three resources needed for operation: memory, CPU and bandwidth.

### 3.1 Memory

A SIP server needs to copy each incoming request in its internal buffers to be able to process the message. The amount of buffered data and the time period the server is supposed to keep the buffered data varies depending on whether the server is working in a stateful or stateless mode. In any case, the server will at least need to maintain the buffered data while contacting another entity such as a AAA [13], DNS server or a database for example. Depending on the message type, the number of Via headers and the body of the message, the size of a SIP message might range from a few hundreds of bytes up to a few thousands.

- **Stateless servers:** Stateless servers need only to maintain a copy of the received messages while processing those messages. As soon as the destination to which a message is to be sent to is determined and the message sent out, the server can delete the buffered data.
- **Stateful servers:** In general we can distinguish between two types of state in SIP:
  - **Transaction state:** This is the state that a server maintains between the start of a transaction, i.e., receiving a request and the end of the transaction, i.e., receiving a final reply for the request. A transaction stateful server needs to keep a copy of the received request as well as the forwarded request. Typically, transaction context consumes about 3 kilobytes (depending on message size, forking and memory management overhead) lasting about one to tens of

seconds if user's interaction is involved. After forwarding INVITE for example, the proxy sets a timer of minimally three minutes only after which a callee is considered to be not capable of providing a final response, i.e., a response between 200 and 699. Also, after forwarding a final non 200 response, i.e., a response between 300 and 699, the server needs to wait for the ACK message and retransmits the response for a period of up to  $64 * T1$  seconds with  $T1$  set usually to 500 msec. In case the server has forked a request to different destinations, the server needs to maintain a copy of the incoming request as well as a copy of all forked requests. In case the server receives a response indicating a redirect situation, the server might initiate the redirect transaction by himself. In this case the server will need to maintain the state till the redirect transaction is replied as well.

- **Session state:** In some scenarios servers may need to maintain some information about the session throughout the lifetime of the session. This is especially the case for communication involving firewall or NAT traversal, for accounting purposes or security reasons as is the case for the 3GPP architecture[14].

### 3.2 Bandwidth

This involves overloading the access links connecting a SIP server to the Internet to such a level as to cause congestion losses. By overloading the server's access links one could cause the loss of SIP messages which causes longer session setup times or even the failure of session setups. Protection of bandwidth is a general transport-layer issue unspecific to SIP and is thus considered out of scope in this paper.

### 3.3 CPU

After receiving a SIP message, the SIP server needs to parse the message, do some processing and forward the message. Depending on the content and type of the message and server policies the actual amount of CPU resources might vary. Whereas the CPU capacity of a well engineered and configured proxy should be able to process SIP messages up to link capacity, there are many server operations which make servers block. Such operations may be misused to quickly paralyze server's operation too.

## 4 EXPLOITABLE ACTIONS

In terms of the actions taken by a SIP server, which consume time, memory and processing power and are thus exposable for attacks we can distinguish the following actions:

- **Message parsing:** The content of SIP messages is coded as plain text. The SIP standard allows for a great level of freedom in setting the order of the message headers, using small or capital letters, including line breaks and so on. Thereby the time taken to parse a message depends very much on the efficiency of the message parser as well as the content of the message. Our measurements show, that even with a very fast parser parsing may consume up to tens of per cent of server's CPU budget.
- **Security check:** SIP might use secure protocols providing state-of-the-art security (SSL/S-MIME). While attacks on these protocols would affect the behavior of a SIP server, such attacks are not SIP specific and are thus will not be dealt with here. In the context of SIP related DoS attacks, exploitation of weaknesses of digest authentication are more relevant and will be addressed in Sec. 6.4.
- **Supporting services:** We define a supporting service as a service that is used by a SIP server in order to fulfill its task of forwarding a received SIP message to the correct destination. Such services include but are not restricted to:
  - **AAA servers:** Such entities are required to check the identity of a user, its eligibility for using a certain service and collecting information about used services.
  - **DNS servers:** A SIP server needs to contact DNS servers to resolve the SIP addresses included in the SIP messages.
  - **Application server:** To execute user specified routing rules a SIP server might need to contact an application server which executes user specific applications.

This interaction is realized in general over some form of inter-process communication and involves the generation of appropriate requests, exchanging those requests over the network and analyzing the replies. Especially when contacting a remote server, as is often the case when contacting a DNS server, this operation can be time consuming and is such a good candidate for exploitation.

## 5 NOTES ON EXPERIMENTAL MEASUREMENTS

To support our architectural discussion, we performed a series of performance measurements. Results of these measurements were helpful in assessing impact of threats and cost of protective measures. Note however, that these results are only of supporting value and have limited generality. They depend on server architecture, hardware performance, format of SIP messages, complexity of server logic, reliance on DNS and other variable factors. All measurements were gained using our SIP Express Router (SER) [17], a server strongly optimized for speed. SER was designed as a multiprocess application in which each incoming message is handled by a separate process. State information is kept in a shared buffer that can be accessed by all processes. This design avoids the blocking of a SIP message behind another one which require longer processing times.

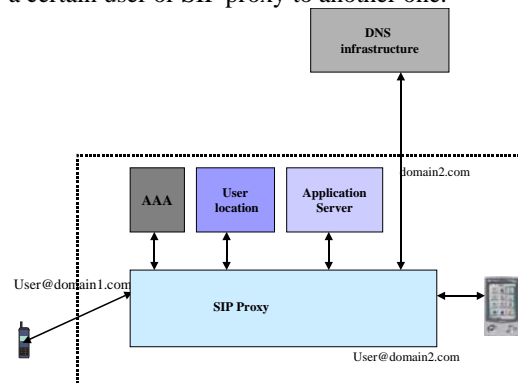
## 6 DENIAL OF SERVICE ATTACK SCENARIOS

While in general DoS attacks are assumed to be mounted on purpose, one should also be aware of the so-called “**unwanted DoS attack**” potential. These usually stem from client implementations of poor quality. While in general such attacks are unintended they may stress servers quite easily. An example of such an unwanted attack we’ve observed is broken digest authentication; invalid clients calculate wrong digest value, are challenged to submit a correct value and continue resubmitting the broken digest value in an infinite loop. Whereas such “careless attacks” are not mounted on purpose, they are not any less harmful than malicious attacks. While such attacks are less massive compared to real attacks, they occur more frequent and should thus not be forgotten.

We show in the following chapters how to mitigate risks caused by both wanted and “unwanted” attacks. Note that the protective countermeasures are not for free and require defense capacity. Though the defense operations are surely less expensive in terms of CPU and memory than operation of an unprotected server under an attack, they do consume server’s resources as well. Thus, the first line of protection starts by deploying a high-performance infrastructure, see [8] for more details.

Besides attacks on SIP itself, as will be described in this part of the paper, it is possible to attack the transport protocols such as TCP, TLS or IPSec

which might be used by SIP. Therefore one should not forget security of supporting protocols either. Even if signaling is made sufficiently secure, security of the whole system may be compromised by a gap in any supporting protocol. An example is the STUN protocol [9] used to accomplish NAT traversal. Its address translation discovery mechanism relies on NAT boxes in middle of the media path to change IP addresses. This reliance is an inherent vulnerability, which can be misused by malicious parties to change the discovered values and corrupt SIP signaling indirectly. (See a posting on IETF Midcom mailing list [9] for more details.). Manipulating DNS entries is another such a case. By maliciously changing the DNS resolution of a SIP server, one can redirect the traffic intended to a certain user or SIP proxy to another one.



**Figure 1 General Communication scenario with SIP**

As a general scenario we consider here a simple topology as described in Figure 1 with a SIP proxy responsible for a certain domain (e.g., domain2.com). That is all calls destined to users registered at this domain (e.g., *user@domain2.com*) would be routed by this proxy. The proxy acts as a registrar and can communicate with a AAA server for checking the users identities. In terms of users we can distinguish between locally registered users which describes users registered at a certain domain or provider and foreign registered users which are users registered at other domains. The proxy responsible for a certain domain has information only on users registered at its domain, i.e., local users.

While in general attacks can be mounted at user agents, i.e., end users and gateways, we will mostly refer to a SIP server as represented by a SIP proxy as the attacked entity. This stems mainly from the fact that proxies are in general the most valuable assets of a service provider.

Note, however, that most of the described attacks could also be directed at user agents.

## 6.1 DoS Attacks Based on Exhaustion of Memory

State maintenance in SIP servers is one of easier targets for DoS attacks. Measurements indicate that a stateful server flooded with a continuous stream of requests belonging to different transactions will run out of memory very quickly. With reasonable server's computational power and bandwidth available on the link between attacker and server, hundreds of megabytes will be exhausted in terms of seconds.

*Experimental note: To demonstrate the speed at which memory is consumed by a SIP server, we flooded a stateful server powered by a PC (128 MB RAM, dual 700 MHz Athlon CPU) with SIP requests at a pace of 35 Mbps. 100 MByte of RAM were exhausted within 22 seconds resulting in exhaustion speed of 36.4 Mbps (4.5 Mbyte/sec).*

### 6.1.1 Exploitation approaches:

- **Brute force attacks:** The simplest method for mounting an attack on the memory of a SIP server is to initiate a large number of SIP sessions with different session identities, i.e., with different To, FROM or call identities for example.
- **Broken sessions:** With brute force attacks memory is only consumed for the duration of a transaction and is released afterwards. To intensify the effects of memory usage, the attackers might infer only parts of a session. This could be done for example by sending INVITE messages to a cooperating receiver. The attacked stateful SIP proxy maintains the session state and awaits the response of the receiver. In case the receiver does not reply, the SIP proxy would retransmit the INVITE message for a period of 32 seconds. In case the receiver replied with a provisional reply, i.e., between 100 and 199, the proxy would need to maintain the state for at least three minutes. Latest SIP specifications even allows a UAS to extend proxy server's transaction time by sending provisional response as long as the UAS wishes to. In the case of a non-cooperative receiver, i.e., one that does not know about the attack and assumes the call to be an ordinary one, the receiver would generate a final answer. The sender is now expected to reply with an ACK message. In case the attacker does not

generate this ACK, the stateful proxy would assume a loss of the final response and retransmit the reply for a period of  $64 \times T1$  seconds with  $T1$  usually set to 0.5 seconds.

### 6.1.2 Counter measures

#### 6.1.2.1 Monitoring and filtering

Similar to web and mail servers SIP proxies need to maintain lists of suspicious users and deny those users from establishing sessions. These lists can be established by monitoring the transactions served by the proxy and logging information such as:

- Which users caused a sudden increase in the number of served transactions?
- Which users are involved in broken transactions?
- Which users are involved in syntactically wrong transactions? This describes users which start sessions that get rejected either by the proxy or by the user due to wrongly formatted SIP messages. This is especially important for filtering out wrongly implemented user agents.
- Note that as the SIP user names included in the FROM headers can be easily faked, it is also important to note the IP address of end hosts generating suspicious messages. Monitoring and filtering is especially beneficial for detecting unwanted attacks. That is, while we would expect an attacker to vary the session identity by changing the TO, FROM and call identification as well as faking their IP addresses, unwanted attackers would probably use the same session identities throughout a session and maintain the FROM header and IP address constant for all initiated sessions.

#### 6.1.2.2 Authentication

In general verifying the identity of a user before forwarding his messages would prevent malicious behavior as the user would be easily traceable – naturally, this is only true if it is not possible for an attacker to presume the identity of a valid user. Using digest authentication, the authenticating server generates a nonce and adds it to a 407 reply in a proxy-authenticate header. The user agent would then terminate the first transaction and generate a new request with his credentials calculated based on the shared secret with the provider and the nonce. In between generating the 407 reply and receiving the new request of the user, the server needs to maintain a copy of the nonce. Thereby, this procedure can be misused

for the broken session attack. That is, a user could generate a request and once asked to verify his identity would just ignore the challenge and start a new request with another session identity. Thereby, the memory of the attacked server would be consumed by the saved nonce and the transaction data. This comes in addition to the wasted CPU resources for calculating the nonces. Another approach that avoids the attack on the memory is to use a so called stateless authentication. Designers of stateless authentication need to build the digest authentication in a way, which does not take storing transactions and/or challenges at servers. This takes putting all verification information in a self-contained manner in SIP messages. Yet another operational constrain is that message authenticity needs to be verifiable by any server in a server farm. A solution to this problem is usage of **predictive nonces** [18]. Predictive nonces allow for stateless authentication and introduce limited message integrity. The construct is based on nonces being calculated in such a way which makes them valid only for validated messages within a time-window. Challenges which are computed as a cryptographic hash of protected request header fields, timestamp and a secret shared among servers in a server farm. When a pair challenge-response arrives at a server, the nonce is first verified to be correct. If it is not correct, it means it was not generated by a server with secret knowledge or some of the protected header fields have been changed. This method works without any changes to the protocol. The only requirement put on interoperability is not to change protected header fields in resubmitted requests, a behaviour which all implementations known to us support. Note: there is a pitfall in generating stateless replies to INVITE request. If a stateless server sends a negative reply to such a request, an ACK will confirm receipt of the reply. The server is then required to consume the ACK and not to forward it anywhere. To implement it statelessly, the ACK must include piece of information hinting "that is an ACK to this server's reply, do not forward". This in-message knowledge allows a server to decide whether to consume or relay the ACK. The only such a place in SIP specification is To-tag. The algorithm is as follows: a stateless server puts its signature in to-tags replies to INVITE and seeks it in ACKs. If present in an ACK, the ACK is dropped. Unfortunately, this algorithm fails with re-INVITES, which already have to-tags in them. Then the server cannot put its signature in replies,

subsequent ACKs are not recognized to be consumed and are mistakenly forwarded. However, we believe that ugliness of this special case is outweighed by benefits of stateless operation. Harm caused by relayed local ACKs is minimal -- downstream servers will drop them. A straight-forward fix would be to drop ACK requests for non-200 replies from SIP specification or to have another place in requests to mirror server's signature [19]. None of these fixes is unfortunately possible without harm to interoperability.

### 6.1.2.3 *Stateless proxy*

An obvious protective measure for reducing the risk of memory exhaustion attacks is to perform as much server's functionality in stateless mode before going statefull. The "**stateless barrier**" should be used to perform as many security checks as possible – these may include

- Stateless authentication of users
- checks of unauthorized 3-rd party registrations,
- detection of replay attack
- presence of virus bodies
- filtering of well-known spam sources.

This functionality may be located in a separate server fronting stateful servers (and perhaps taking care of load distribution). The other alternative is to have the stateless logic executed in the same server but to proceed to stateful execution only after all stateless checks succeed. After all the stateless checks, transactional state may be established. While some functions such as static forwarding (e.g., least-cost gateway routing) may be easily executed statelessly, in the following we list some functionalities taht inherently require stateful mode:

- Request forking to avoid confusion of upstream clients unaware of forking.
- Accounting to report on the results of transactions as opposed to reporting on all individual messages.
- Retransmission buffer, particularly important if a SIP path is known to include a wireless hop to absorb too rapid retransmissions.
- Some services such as forward\_on\_event, e.g., forward to voicemail on busy. In this case the original request needs to be kept for the new request instantiation.
- Servers using a registration database as the basis for the routing decision need to be at least conditionally stateful to preserve forwarding coherency throughout a session

and avoid thereby routing inconsistency due to REGISTER updates.

## 6.2 CPU Attacks

At a SIP server, CPU resources are required for the following tasks:

- **Message parsing:** In order to figure out how to handle an incoming message the server needs to parse at least a part of the message. To achieve optimal behavior, a SIP server should only parse those parts that are needed for its correct functioning. In general, a server that is overloaded with message parsing is an indication of a bad implementation of the server or under dimensioned hardware.
- **Security checks:** For verifying the identity of a user, a SIP server needs to generate a nonce and then check the credentials of user. This checking uses hashing schemes such as MD5 which require relatively a low calculation overhead. Thereby a server exhibiting signs of overload due to security checks is a good indication of a bad implementation or under dimensioned hardware.
- **Application execution:** A SIP server might need to execute a certain application, i.e., a CPL or CGI script or some other kind of application, after receiving a request. The amount of the used CPU resources depends on the application type and its complexity. In case the application server is located on the same hardware platform as the SIP server, then the CPU resources used for the execution of the applications is no longer available for processing SIP messages. In case the application server is located on a different hardware platform then some form of inter-process communication between the SIP server and the application server is needed. Thereby attacks on the application server result in blocking the SIP server after requesting the execution of an application and until the application server generates a reply. This could be used by an attacker by sending requests with varying FROM headers to users which have registered with the attacked SIP server some sort of applications to be executed whenever receiving a request. To make sure that such users exist, the attacker might register himself as a legal user and ask for the execution of a complicated application whenever a request is addressed to this registered identity. The attacker can now send requests to the registered user and overload the server this way. Such attacks can only be countered by monitoring and filtering mechanisms as was described in Sec. 6.1.2.1. Further, the provider should offer users only simple and secure tools for service creation that do not allow the user to specify an application that causes infinite loops, see [10] for a comparison of different service creation tools for SIP.
- **Interaction with external servers:** As already indicated a SIP proxy might need to contact an external server to fetch some information or realize a service. This not only consumes processing time but also can cause the server to block and reject new incoming messages while the SIP proxy is awaiting an answer from the contacted server. Depending on the type of external server we can identify different attack possibilities:
  - **AAA Attacks:** By sending requests requiring authentication such as register messages or invitations towards a protected user agent such as PSTN gateway, an attacker can cause the SIP proxy to continually query the user's credentials from a AAA database.
  - **Application execution attacks:** As described above, a user can incur an increased load on the SIP proxy by repeatedly generating requests that require the execution of certain application logic. This would require the proxy to contact the application server and wait for the result of the application execution.
  - **DNS attacks:** AAA and application servers can be assumed to be administered by the provider of the SIP proxy. Thereby, these servers can be dimensioned in such a manner as to withstand a high load of requests using high speed hardware and network connection to the SIP proxy. However, for the case of DNS the provider must rely on the public DNS infrastructure. A DNS attack is realized by forcing the SIP proxy to try and resolve a non existent host name when relaying requests or replies. A SIP server contacting a DNS server and blocking till it receives the answer would block for a few seconds. An attacker can force a SIP proxy to contact a DNS server by putting an

irresolvable address in such as the VIA, Route, Contact or Request-URI.

- **Forwarding to non-existent TCP receivers:** Another form of attack that can cause a server to block is to cause a server to attempt to communicate with an unresponsive server by forcing it to contact an **unresponsive address** using a TCP connection. That is, a caller that has indicated his wish to use TCP as the transport protocol can force the SIP proxy to initiate a TCP connection to a certain destination. If the contacted callee does not respond, SIP processing may be blocked until a failure timeout expires. A SIP proxy can be forced to forward a message to an unresponsive address by adding the unresponsive address a one number of headers such as the VIA, Route, Contact or Request-URI.

Attempting to protect SIP infrastructure against these threats is inherently difficult, as DoS attacks are generally difficult to distinguish from legitimate use. Risks are high: even with an unwanted attack, a single transaction (for example, an invitation to a destination served by a temporarily failed DNS server) may temporarily disable server's operation for several seconds.

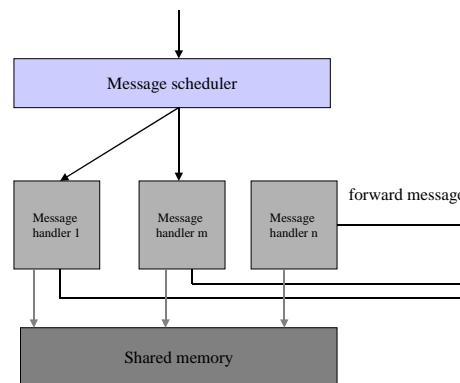
### 6.2.1 Countermeasures

- **Server design:** The first line of defense against any DoS attacks is achieved by using well-dimensioned hardware with fast CPUs and large memory and high speed network connection. Additionally, the software itself needs to be designed with security, speed and attack possibility in mind. This implies deploying some or all of the following server design options:

- **Clean and efficient implementation:** Implementers need especially to use efficient and fast memory allocation schemes, event handling and parsing mechanisms.
- **Parallel processing:** In order to avoid blocking incoming messages while the server is busy processing a message or while waiting for an answer from an external server (e.g., DNS) SIP proxies should be implemented using threads or parallel processes with each process or thread responsible for processing one message at a time. Such a design is depicted in Figure 2. Here a core part only acts as a message scheduler distributing incoming messages between

the processes. Each process is then responsible for parsing the message, initiating any DNS requests or requesting the execution of an application and finally forwarding the message. State information can be shared among the processes using some form of shared memory. Note however, that even with a thread based server, server blocking is still a danger. If an attacker generates a higher number of messages that cause the server to block than the available number of threads the server will still block.

*Experimental note: We blocked a proxy, which was designed to use four parallel threads by an absolutely legitimate invitation to an irresolvable destination for a couple of seconds. The original request with irresolvable DNS address in request-uri blocked the first thread whereas subsequent retransmissions made all remaining threads busy within next 7.5 sec. The server was then completely blocked for next few seconds until DNS query timeout expired.*



**Figure 2** Design scheme of a parallel SIP server

- **Non-Blocking design:** Another option is to design all requests to external servers as non-blocking. That is, after issuing a DNS request the server would not wait till an answer for the request was generated but would queue the request in an event queue, save the data of the transaction, break it and move to processing the next request. When a reply for the request arrives the main process is notified and the broken transaction is scheduled to continue. Thereby, the server would only block on memory exhaustion. However, while this approach reduces the possibility of completely blocking a server, the

implementation complexity and memory requirements increase considerably.

- **DNS handling:** A preventive measure for reducing the effects of DNS attacks is to reduce the amount of DNS requests issued by the server. This can be achieved in one of the following ways:
  - **Receive in VIA:** The VIA list in SIP requests indicates the path taken by the request so far. That is, the caller adds its URI as the first entry in the VIA list. Each proxy that receives this request adds its URI to the list. The receiver of the request adds the VIA list to its replies and then sends the reply to the topmost VIA entry. Each proxy receiving the reply removes the VIA entry indicating its URI and forwards the reply to the new topmost entry. For the case of multihomed proxies or user agents or for the case of traversing a network address translator the address indicated in the VIA entry might differ from the IP address of the entity that forwarded the request to a proxy. For this proxy to exactly identify the IP address to which to forward the replies to, a proxy can add a “receive” parameter to the topmost VIA entry in the received request. When receiving the reply to this request, the proxy does not forward the reply to the URI indicated in the VIA entry but to the address it had previously added in the receive parameter. To avoid the need for resolving a URI included in a VIA entry of a reply, one can always add a receive parameter to the VIA entry of the request with the IP address of the sending entity. While this behavior is optional in case the IP address of the sender and the URI in the VIA entry, it is very helpful in avoiding issuing a DNS request after receiving a reply.
  - **Restricted contacts:** Another approach is to restrict the form of the contact addresses in the registration message to IP addresses and not to host names. This avoids the need to resolve the contact address when forwarding a request.
  - **DNS caching:** DNS caches save the results of the latest DNS queries and can be used for answering future queries. Different operating systems like Solaris8 already include this feature. In case this is not included then the server should implement its own DNS cache.

*Experimental note: We repeated the same experiment like above. We blocked a proxy which was designed to use four parallel threads by an absolutely legitimate invitation to an irresolvable destination for a couple of seconds. The original request with irresolvable DNS address in request-uri blocked the first thread whereas subsequent retransmissions made all remaining threads busy within next 7.5 sec. With DNS cache enabled (nscd), all processes were unblocked at once. Without DNS cache, each processes unblocked on timeout expiration – total blocked time was longer.*

## 6.3 Attack Fans

A natural prevention of DoS attacks is low-bandwidth: under normal conditions, a SIP attacker is prevented from flooding a server with bogus messages causing resource exhaustion by a link bottleneck. An inventive attacker is thus motivated to use an attack amplifier, which multiplies harmful effect of few bogus messages.

### 6.3.1 Loops and Spirals

SIP unfortunately provides excellent means for attack amplification. Specifically, SIP's ability to loop and fork requests may account for exhausting a server's memory and CPU with a single originating attack message.

In the loop-amplification scenario, the attacker needs to convince a proxy server to rewrite a request to a location, which resolves to the server itself. That will result in the server loading itself under high pressure.

As an example for such an attack a user might register at one domain, e.g., domain1.com, as user *user@domain1.com* with a contact address of *user@domain1.com*. Thereby when a request is sent to *user@domain1.com*, the proxy would check the user location database, rewrite the R-URI of the request to the contact address indicated in the contact address of the user's registration and forward the request to domain1.com, i.e., to itself. The same procedure is started again.

To encounter such an attack the provider might check the contact addresses indicated in the registration messages and reject registration messages in which the contact is similar to the user's identity.

This can be overcome by registering two users at the same provider or two different providers with the contact address of each user indicating the address of the other. That is, *user1@domain1.com* would indicate *user2@domain2.com* as his contact address and

`user2@domain2.com` would indicate `user1@domain1.com` as his contact address. This would cause all messages sent to either users to be looped through the two providers.

The SIP specification provides several means to mitigate infinite loop by using a header named Max-Forwards, which is decremented for, by each traversed proxy. In a similar manner to IP's Time-To-Live, the packet is dropped after reaching the value of 0. Further a 483 (too many hops) response be generated.

Loop detection is realized by comparing a transaction identity of a received message with that of the currently active transaction. This identity is comprised of the FTOM, TO, R-URI, call-id and top-most VIA entry. Thereby, for each received message, all of those headers need to be checked and compared to the ones already saved at the server.

As requirement strength for loop-detection has been decreased to SHOULD (for performance reasons) and an attacker can set up legitimate SIP routing spirals, only Max-Forwards can be relied on. Note that a UAC is supposed to set the initial Max-Forwards value, leaving the length of a loop up to an attacker unless a proxy server rewrites this value. That is, if only Max-Forwards was used for loop detection, a request sent to the manipulated account would generate Max-Forwards messages.

While such attacks are easy to launch, they require the attacker to register himself with the provider beforehand and would allow the provider to discover the identity of the attacker once the attack was recognized. The risk of being detected is thereby one barrier for actually using this attack. Besides manipulating ones own account, an attacker can try to manipulate the contact address of other users. This could be achieved by exploiting the weaknesses of digest authentication as will be described in Ses. 6.4.

*Note: The latest SIP specification, RFC 3261, mistakenly disables implementation of loop detection, thus opening a back-door for loop-based attack amplification. The problem is it suggests using Via header field branch parameter as input value for the next-hop branch value. Because of that, two looped request with identical request-uri will not be recognized as a loop – SIP specification mandates the branch parameter to constitute a part of transaction key. The problem was reported shortly after release of RFC3261 to the working group, whose consensus was to ignore loop checking in favor of Max-Forwards checks.*

### 6.3.2 Forking

The other amplification mechanism is forking. An attacker can easily register N locations with an address resulting in N-times higher overhead during every request sent to the address. That is, we consider an attacker having registered N+1 accounts at N+1 providers. At each provider the attacker registers N contact addresses pointing to the other accounts. With such a scenario a request would make each of the SIP servers at each of the providers to be involved in the routing of  $N^{Max-Forwards}$  messages. In addition to the overload due to message processing using forking to amplify an attack is especially attractive, as forking proxies need to be statefull. Thereby, this attack combines both CPU and memory exhaustion attacks.

To reduce the effects of this attack, the proxy might use loop detection. However, this increases the complexity of message processing considerably. Another option is to limit the value of Max-Forwards. That is the proxy might set a maximum limit for the value Max-Forwards that it might accept. If a request was received with Max-Forwards set to a higher value, the proxy would reset this value to its defined limit. Due to the higher danger of forked requests, the proxy might have a value for normal requests and a lower one for forked ones. This reduces the risk of extended loops by using large Max-Forwards values.

### 6.3.3 Distributed DoS Attacks

Attacks mentioned so far have a single source, which may be easily detected. Detection of attacker is much harder if an attacker manipulates many devices on the network to strike a victim. The first well-described use of this mechanism is so called "Reflection Distributed DoS attack". In this attack, an attacker sends forged TCP connection requests to innocent public Internet hosts, **attack reflectors**, and forges source IP address. In reply to these requests, all approached hosts flood a victim with replies. The victim is then hit by a flood of replies coming from a variety of hosts, making difficult to detect presence of an attack and its originator.

Such an attack as well as other generic-transport layer attacks (see a CERT report [20] for more details on DoS trends) can also be used to paralyze a SIP infrastructure. Further, such attacks can be replicated at the SIP level by forcing a proxy to forward messages to some victim. Here we can distinguish two possible attack methods:

- **Reply forwarding:** An attacker can force a SIP proxy to act as a reflector by including the victim's address in top-most Via header of requests sent to different reflectors. The reflectors will send back a reply (not found, authentication challenge, call does not exist, etc.) The reflectors may be any SIP servers including registrars, proxy servers and SIP phones. Servers may attempt to discard requests coming from other IP addresses than advertised in Via. Unfortunately, this mechanism has many limitations: It can be fooled by spoofed IP address and it disqualifies SIP clients behind NAT.
- **Request forwarding:** Similarly, innocent proxy servers may be misused to route requests to a victim. SIP headers such as Route, VIA or Request-URI may be used to force a proxy server to route a request to a victim. This scenario is more harmful than reply forwarding. Request processing logic is typically more complex than reply processing logic. It may take consulting an SQL server to determine user location, resolving an irresolvable DNS destination, verifying MD5 credentials, and other CPU-consuming operations. Even worse, techniques such forking in attack reflectors may be used to amplify the attack strength.

#### 6.4 Exploiting Digest Weaknesses

The digest authentication algorithm [12] is currently the most frequently deployed security mechanism with SIP. Unfortunately, the digest authentication features several major weaknesses, which can be easily exploited. Firstly, it provides only limited message integrity (the body of the message may be included in the digest calculation but not the header) and secondly, several request methods do not use even the digest algorithm.

Lack of message integrity is particularly bad, as it allows attackers to easily change content of message. An attacker may act as a man in the middle, grab a message with valid credentials, change it in his favor and send it to a server. Even more easily, as many implementations accept the same credentials within a period of time for performance reasons, an attacker may replay a message captured on the net and modified maliciously.

Impact of attacker's ability to modify messages is very bad. An attacker may register as a legitimate user or redirect a valid conversation to his devices. Though a prohibitive challenge for an attacker would be to impersonate victim's voice in VoIP scenarios, there are other SIP

applications, such as instant messaging, in which discovering stolen identity is hard or impossible.

To a certain extent, the security risk can be mitigated by use of predictive nonces. This technique, previously discussed in Sec. 6.1.2.2 and proposed in [18] allows cryptographic binding of critical header fields to challenges, thus preventing attackers from changing them and guaranteeing their integrity.

The other problem with digest is that SIP's transactional model provides only very weak authentication of two request methods, CANCEL and ACK. The main reason is that these methods operate in hop-by-hop mode and may be generated by any server in a signaling chain. As hardly every server in the chain will have an established security association with other communication parties, authentication of these requests is effectively impossible. Other reason why this is impossible is that sequence numbers of CANCELs and ACKs must be the same as of requests to which they relate and thus cannot be challenged as resubmitted requests would have to have a higher Cseq number which would then no more match the original message.

Lack of authentication of CANCEL and ACK allows attackers to inject these requests on their own. Faking a CANCEL request may result in denial of session establishment.

Unfortunately, as long as transport security does not become widely deployed, there is no way to prevent hop-by-hop request CANCEL and ACK from being forged.

## 7 SUMMARY OF OPERATIONAL GUIDELINES

As a general summary of techniques to deploy in order to reduce the risks of DoS attacks we recommend the following processing order after receiving a request:

1. Check if there is already an established transaction for the incoming request and if so, absorb it; proceed to the next step otherwise
2. If a request has a DNS name in topmost Via, ignore it and use packet's source IP address to avoid DNS resolution overhead on sending replies
3. If deployment scenario allows it then authenticate statelessly to avoid memory exhaustion attack
4. Make routine checks
  - a. Check for presence of viruses
  - b. Scan for well-known attack patterns;
  - c. If max-forwards higher than local policy mandates, rewrite it to a lower value to

prevent a request from exhausting server resources by loops

- d. Drop all suspicious packets
5. Optionally, establish transaction state. That helps to avoid burdening the server's resources with executing potentially expensive service logic for each retransmission received; however, don't establish the state if request wasn't authenticated as this would allow anonymous attackers to exhaust memory quickly

Additional considerations apply to processing REGISTER requests:

1. Use predictive nonces to assure that Contacts in REGISTERs are not forged.
2. Use a quota for number of contacts per address of record to prevent escalation of forking amplification
3. Deny suspicious contact addresses; these may include private IP addresses (RFC1918) for a public server (accepting them would allow an attacker to route requests to the private networks to which the SIP server is connected to) or DNS names (they might cause a blocking server to block and a non-blocking server to run out of memory)

## 8 SUMMARY

In this presented we presented an overview of possible approaches for mounting denial of service attacks. In general we distinguish between attacks on the transport and authentication protocols used by SIP but not part of SIP and attacks on SIP itself.

A major conclusion of the work is the knowledge that SIP provides a wide range of features that can be used to mounting DoS attacks.

Denial of service attacks is a hard class of problem to solve. The inherent problem root is the desire to offer services to the public Internet together with the difficulty to recognize "friends" from "enemies". In addition to this black-and-white classification, there is large gray area of "unwanted attacks", i.e., users trying to access server resources in a good will but exhausting them through unknowledgeble, misbehaving implementation.

Our operational experience at the public iptel.org SIP site tells that currently misbehaving implementations and misconfigurations are the most compelling issue. Real world examples include implementations constructing invalid credentials and resubmitting them in infinite loops, upstream clients not recognizing final replies and resending original requests, phones misconfigured with wrong password and

resubmitting credentials every few minutes, or phones misconfigured to use too short registration period. Such errors have substantial impact on server performance and constitute up to tens of percent of server load.

Thus, the major prerequisite to successful DoS defense is reasonable performance. Even though various countermeasure mechanisms may make attacks less harmful, they consume server resources too and need to rely on solid performance.

Other observation is that there is no "one-size-fits-it-all" solution. Which countermeasures are best deployed depends very much on the capabilities of the systems in use and network architectures. A good example demonstrating the trade-offs is protection against attempts to block servers by irresolvable addresses in topmost Via header fields. The answer is easy in closed environments. Network administrator can assume users are easy to track and sets defeating harms caused by erroneous implementations and misconfigurations as priority. He configures the SIP infrastructure to process all requests statefully, which will avoid blocking more CPU by absorbing retransmissions. The same technique would be quite deadly in public environments. Knowledgeable attackers can easily flood a server with irresolvable requests belonging to different transactions, which would not only result in blocking lot of CPU resources, but also quickly exploiting memory.

Nevertheless, we believe there are practices, which can well support servers' robustness reasonably. Particularly, we recommend avoidance of server blocking. Server blocking may be caused by added-value server logic or by specification-dictated DNS resolution. Other advisable practice for closed environments is to authenticate users prior to processing their requests. Authentication must be then designed statelessly to avoid memory attacks and not result in CPU blocking in sending challenges.

The most widely deployed digest authentication provides only rudimentary security and we strongly advice to strengthen it. Particular methods we suggest are binding of challenges to timestamps data in authenticated requests. This technique allows simple message integrity and stateless digest processing.

We have been experimenting with these techniques and were able to reduce server load at our public site by 34% per cent. Nevertheless, caution is advisable in interpreting these numbers as a shift in usage patterns may quickly necessitate a different deployment strategy.

## 9 REFERENCES

- [1] Computer Emergency Response Team. CERT Advisory CA-2000-01 Denial-of-Service Developments. <http://www.cert.org/advisories/CA-2000-01.html>, January 2000.
- [2] Computer Security Institute and Federal Bureau of Investigation. 1999 CSI/FBI Computer Crime and Security Survey. Computer Security Institute Publication, March 1999.
- [3] Y. Chen. Toward a quantitative understanding of DoS. Class Proposal, University of California, Berkeley, USA, [http://www.cs.berkeley.edu/~yanchen/course/261\\_proposal.html](http://www.cs.berkeley.edu/~yanchen/course/261_proposal.html), 2000.
- [4] J. D. Howard, An Analysis of Security Incidents on the Internet. PhD thesis, Carnegie Mellon University, August 1998.
- [5] D. Moore, G. Voelker, S. Savage. Inferring Internet Denial-of-Service Activity. In Proceedings of the 10th USENIX Security Symposium, pages 9--22, August 2001. V. Paxson. An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks. White Paper, AT&T Center for Internet Research at ICSI, International Computer Science Institute Berkeley, USA, 2001.
- [6] D. S. Reeves, V. Mahadik, X. Wu. Detection of Denial of Service Attacks Based On  $\chi^2$  Statistic And EWMA Control Charts. White Paper, NC State University, Advanced Networking Research Group, January 2002.
- [7] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Spark, M. Handley, E. Schooler, "Session Initiation Protocol", RFC 3261
- [8] Jiri Kuthan, "Accelerating SIP", in SIP 2002, Paris, January 2002. <http://www.iptel.org/~jiri/publications/sip2002-kuthan-acc00.pdf>
- [9] Christian Huitema: "STUN Security Showstopper", a posting to IETF/Midcom mailing list. <http://www1.ietf.org/mail-archive/working-groups/midcom/current/msg02239.html>
- [10] Jiri Kutha, "Comparison of Service Creation Approaches for SIP", International SIP conference 2000, March 2000
- [11] P. Srisuresh, J. Kuthan, J. Rosenberg: "Middlebox Communication Architecture and framework", February 2001, IETF, Internet Draft
- [12] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," Internet Engineering Task Force, RFC 2617, Jun. 1999.
- [13] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, D. Spence, "Generic AAA Architecture", RFC 2903 , Experimental, August 2000
- [14] 3GPP Technical Specification 3GPP TS 24.228 " Technical Specification Group Core Network; Signaling flows for the IP multimedia call control based on SIP and SDP", 3rd Generation Partnership Project, 2003
- [15] Dierks, C. Allen, "The TLS Protocol Version 1.0" RFC 2246, January 1999.
- [16] Rosenberg, Weinberger, Huitema, Mahy: "STUN - Simple Traversal of UDP Through NATs", Internet Draft, work in progress, IETF, April 2002. draft-ietf-midcom-stun-00.txt
- [17] iptel.org: "SIP Express Router", proprietary product-sheet. <http://www.iptel.org/ser>
- [18] Rosenberg: "Request Header Integrity in SIP and HTTP Digest using Predictive Nonces", expired Internet Draft, work in progress, IETF, June 2001. draft-rosenberg-sip-http-pnonce-00.txt
- [19] Marshall et al.: "SIP Extensions for supporting Distributed Call State", Internet Draft, work in progress, IETF, August 2001. draft-ietf-sip-state-02.txt
- [20] Gibson: "Distributed Reflection Denial of Service", on-line tutorial, <http://grc.com/dos/drdo.htm>
- [21] Houle, Waver: "Trends in Denial of Service Attack Technology", CERT report, October 2001, [http://www.cert.org/archive/pdf/DoS\\_trends.pdf](http://www.cert.org/archive/pdf/DoS_trends.pdf)