
Denial of Service Attacks Targeting a SIP VoIP Infrastructure: Attack Scenarios and Prevention Mechanisms

Dorgham Sisalem and Jiri Kuthan, Tekelec
Sven Ehlert, Fraunhofer Fokus

Abstract

In this article we address the issue of denial of service 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. In this context we mainly identify attacks based on exhaustion of the memory of VoIP servers, or attacks that incur high CPU load. We deliver an overview of different attack possibilities and explain some attacks in more detail, including attacks utilizing the DNS system and those targeting the parser. 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.

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 is due to the fact that voice over IP (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 flat rate Internet access, are also cheap.

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. According to a 2004 CSI/FBI survey report 17 percent of respondents detected DoS attacks directed against them, with the respondents indicating that DoS was the most costly cyberattack for them, even before theft of proprietary information [1]. To make things worse, attackers have developed tools to coordinate distributed attacks from many separate sites, also known as distributed denial of service (DDoS) attacks.

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. Furthermore, 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. For an extensive overview on DoS attacks in the Internet refer to [2].

The Session Initiation Protocol (SIP) [3] is establishing itself as the de facto standard for VoIP services in the Internet and next generation networks. Therefore, this article 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.

SIP is a text-based protocol designed to establish or terminate a session between two partners. The message format is similar to HTTP [4], with message headers and corresponding values, such as "From: user@sip.org" to denote the sender of a message.

Several entities form a SIP network, including *user agents* that generate or terminate SIP requests, *registrars*, where users log in and announce their availability in the SIP network, and *proxies* that forward requests in the SIP networks. For a detailed overview of SIP refer to [5]. This article is organized as follows. First, 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 review attacks on memory, CPU, and bandwidth as well as countermeasures, followed by a detailed description of parser and DNS attacks. Finally, a list of operational guidelines for deploying secure SIP services is provided.

Exploitable SIP Resources

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

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 an authentication, authorization, and accounting (AAA) server, a Domain Name Service (DNS) server, or a database, for example. 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 maintain a copy of the received messages while processing those messages. As soon as the destination to which a message is to be sent is determined and the message is 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 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 kbytes (depending on message size, forking, and memory management overhead) lasting about 1 s to tens of seconds if user interaction is involved.
- **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 Third Generation Partnership Program (3GPP) architecture [6].

CPU

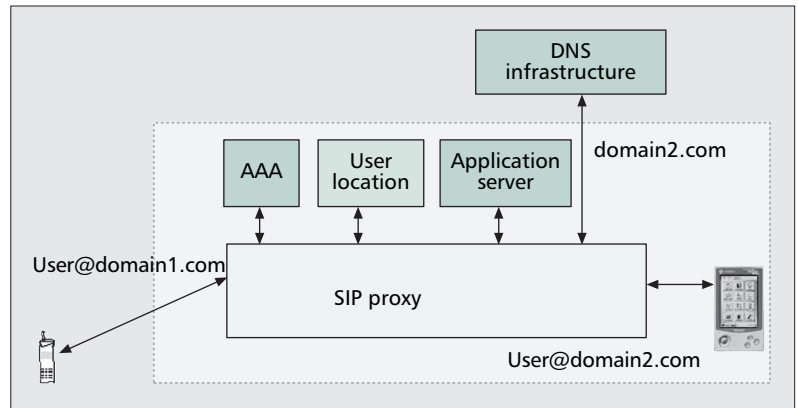
After receiving a SIP message, the SIP server needs to parse the message, do some processing (e.g., authentication), perform transaction mapping and forward the message. Depending on the content and type of the message and server policies, the actual amount of CPU resources might vary. Whereas the CPU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

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 beyond the scope of this article.

Overview of 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. An example of such an unwanted attack we have observed is broken digest authentication; invalid clients calculate wrong digest values, are challenged to



■ Figure 1. General communication scenario with SIP.

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 than real attacks, they occur more frequently and should thus not be forgotten.

We show in the following sections how to mitigate risks caused by both wanted and “unwanted” attacks. Note that the protective countermeasures are not free and require defense capacity. Although 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 resources as well. Thus, the first line of protection starts by deploying a high-performance infrastructure.

Besides attacks on SIP itself, as described in this part of the article, it is possible to attack the transport protocols such as TCP, TLS, or IPSec that 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 [7] used to accomplish NAT traversal. Its address translation discovery mechanism relies on NAT boxes in the middle of the media path to change IP addresses. This reliance is an inherent vulnerability that can be misused by malicious parties to change the discovered values and corrupt SIP signaling indirectly.

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.

As a general scenario we consider here a simple topology as described in Fig. 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 an AAA server for checking users' identities. In terms of users, we can distinguish between locally registered users (i.e., users registered at a certain domain or provider) and foreign registered users (i.e., 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 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.

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.

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.

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. If the receiver does not reply, the SIP proxy would need to maintain the state for at least 3 min while it still tries to retransmit the message.

Possible Countermeasures

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 user behavior (e.g., users that cause a sudden increase in the number of served transactions or users involved in broken transactions).

Authentication: In general, verifying the identity of a user before forwarding his/her 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.

Like HTTP, SIP uses digest authentication [8], which requires state maintenance at the server by storing the issued challenge. This can be misused for a broken session attack, if attackers ignore or falsely respond to authentication requests and start another session instead.

A solution to this problem is usage of **predictive nonces** [9] that allow for stateless authentication and introduce limited message integrity. The construct is based on nonces being calculated in a way that makes them valid only for validated messages within a time window. When a challenge-response pair arrives at a server, the nonce is first verified to be correct, followed by the verification of the response. This method works without any changes to the protocol.

Stateless proxy: An obvious protective measure for reducing the risk of memory exhaustion attacks is to perform as much server functionality in stateless mode before going stateful. The **stateless barrier** should be used to perform as many security checks as possible, including stateless authentication of users, checks of unauthorized third party registrations, or filtering of well-known spam sources.

This functionality may be located in a separate server fronting stateful servers, or the stateless logic can be executed in the same server but proceed to stateful execution only after all stateless checks have succeeded. After all the stateless checks, transactional state may be established.

CPU Attacks

Besides the processing power needed for parsing incoming SIP messages, CPU resources are required for the following tasks.

Security Checks — For verifying the identity of a user, a SIP server needs to generate a nonce and then check the credentials of the user. This checking uses hashing schemes such as MD5, which require a low calculation overhead. Thereby a server exhibiting signs of overload due to security checks is a good indication of a bad implementation or underdimensioned 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 CPU resources used depends on the application type and its complexity. If the application server is located on the same hardware platform as the SIP server, the CPU resources used for execution of the applications is no longer available for processing SIP messages. If the application server is located on a different hardware platform, some form of remote communication between the SIP server and the application server is needed. Thus, attacks on the application server result in blocking the SIP server after requesting the execution of an application until the application server generates a reply. This could be used by an attacker by sending requests with varying From headers to users that 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 described earlier. Furthermore, 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. Several external servers can be considered for attack, for example, through constantly querying the user's credentials at an AAA server, or incurring increased load by issuing requests that require the execution of certain complex application logic at an application server.

Forwarding to Nonexistent 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 a 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 to any 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 (e.g., an invitation to a destination served by a temporarily failed DNS server) may temporarily disable a server's operation for several seconds.

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., AAA) a SIP proxy should be implemented using threads or parallel processes with each process or thread responsible for processing one message at a time. 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 than the available number of threads, the server will still block.

Attack Amplification

In the previous section we have outlined general attack possibilities. An attacker will use these attacks in combination with other possibilities to increase the harm caused by the described attacks.

Loops and Forks — 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, thus incurring high load at the server itself.

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. Similar to IP's Time-To-Live, the packet is dropped after reaching the value of 0. Furthermore, a 483 (too many hops) response will be generated.

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 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 stateful. Thus, this attack combines both CPU and memory exhaustion attacks.

To reduce the effects of this attack, the proxy might limit the value of Max-Forwards. That is, the proxy might set a maximum limit for the value of Max-Forwards 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.

Distributed DoS Attacks — Attacks mentioned so far have a single source, which may easily be detected. Detection of an attacker is much harder if an attacker manipulates many devices on the network to strike a victim. A well described use of this mechanism is the 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 a 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 it difficult to detect the presence of an attack and its originator. 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 the topmost Via header of requests sent to different reflectors. The reflectors will send back a reply (not found, authentication challenge, call does not exist, etc.). Reflectors may be any SIP servers, including registrars, proxy servers, and SIP phones. Servers may attempt to discard requests coming from IP addresses other than advertised in Via. Unfortunately, this mechanism has many limitations: it can be fooled by spoofed IP addresses and 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.

Detailed Attack Scenarios

Message Parsing

In order to figure out how to handle an incoming message, the server needs to parse at least part of the message and check its consistency. However, due to the free text format of the SIP protocol even a perfectly valid SIP message can be constructed in a way to hamper proper parsing. Here we give a list of possibilities how to complicate message parsing:

- An attacker can create unnecessary long messages in a simple way by adding additional headers (like informative header fields, e.g., Supported) in conjunction with a large message body. Many SIP messages may include bodies, even when they are not needed in every message. Instead of only depleting processor power, longer message also increase network utilization and memory usage. For an attacker to be effective with this method, he has to utilize only well formed header fields, as other header fields should be ignored by a well implemented parser. Server implementations should thus check messages for a certain size limit and reject messages exceeding this limit with a 413 (Request Entity Too Large) message.

- Under certain conditions clients have to send messages using a congestion controlled protocol, which generally results in the usage of TCP. To avoid fragmentation, the condition is met if a request is within 200 bytes of the path maximum transmission unit (MTU), or if it is larger than 1300 bytes when the path MTU is unknown. By forcing a server to accept TCP connections, it becomes vulnerable to general TCP DoS attacks, as additional state is created even in a stateless proxy. As a countermeasure SIP entities could be configured to not support TCP messages; however, this would not be compliant with the SIP specification.

- Poor parser implementations can be rendered inoperable by including message bodies of a size that does not match that indicated in the Content-Length header.

- Additionally, the SIP standard mandates that headers that have multiple values can be separated into individual header fields so each only contains one value. If multiple message headers of the same field are included in a message where these headers are spread all over the message, this will further complicate parsing. Figure 2 illustrates three possibilities to compose a message with multiple Contact fields. Especially, the following header fields can be distributed in such a way in a SIP message: Accept-Encoding, Accept-Language, Alert-Info, Allow, Authentication-Info, Call-Info, Contact, Content-Encoding, Content-Language, Error-Info, In-Reply-To, Proxy-Require, Record-Route, Require, Route, Supported, Unsupported, User-Agent, Via, and Warning.

- Some message headers are more vital for processing than others. Vital header fields are all routing-specific fields, like To, Via, and Route. So messages with these fields placed toward the end of the message require more processing power to parse.

```

From: ...
To: ...
Contact: <sip:user1@sip.org>
Contact: <sip:user2@sip.org>
Contact: <sip:user3@sip.org>
Contact: <sip:user4@sip.org>
Call-Id: ...
CSeq: ...

From: ...
To: ...
Contact: <sip:user1@sip.org>,
Contact: <sip:user2@sip.org>,
Contact: <sip:user3@sip.org>,
Contact: <sip:user4@sip.org>
Call-Id: ...
CSeq: ...

Contact: <sip:user1@sip.org>
From: ...
Contact: <sip:user2@sip.org>
To: ...
Contact: <sip:user3@sip.org>
Call-Id: ...
CSeq: ...
Contact: <sip:user4@sip.org>

```

Figure 2. Multiple header possibilities.

One way to accomplish this is by inserting multiple informative header fields, e.g. Allow or Supported, before the routing fields.

• SIP as defined in RFC 3261 [1] is a refined version of the previous standard as defined in RFC 2543 [11]. Some of the newer design decisions are made to simplify certain operations. However, any RFC 3261 compliant SIP element must be able to handle RFC 2543 messages, which can complicate processing. As such, this can be used by an attacker. Among these modifications are:

VIA headers. Via header fields contain a branch parameter. If this branch parameter does not start with the magic cookie “z9hGbkK,” the message is considered to be pre-RFC 3261. This would indicate a fallback to the more complex RFC 2581 message handling routine.

Missing tag field: Messages with a To and From header field, but without a tag field, need to be checked by the UAS against all ongoing transactions, thus requiring more processing overhead.

Parsing attacks can be countered by an efficient implementation (e.g., by parsing only those parts 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 underdimensioned hardware. Additionally, monitoring incoming messages for suspicious content will further mitigate parser attacks.

DNS Attacks

A rather simple way to disturb server operation is to include unresolvable host names in a SIP message. A SIP message can contain URIs in varying header fields, including Via, Route, Record-Route, Contact, and Request-URI.

A SIP server encountering an unresolvable address in a header field (e.g., Via: unresolvable.domain.org) has to wait for the resolver reply to continue operation. If the DNS subsystem knows about the domain, a timely answer might even arrive in case of an unresolvable address. However, often no answer can be provided until a timeout occurs at the DNS system. We have witnessed through simulation that in such cases a SIP server can be blocked for up to 5 s through one simple message (Fig. 3). We describe a number of approaches for reducing the effects of such attacks.

Reduced FQDN usage: Every fully qualified domain name (FQDN) in a SIP message needs to be resolved into a numeric IP address. Hence, FQDN should not be used unless necessary. The standard already provides a means to reduce FQDN usage in Via headers that indicate the path taken by the request so far. That is, each proxy that receives a 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. To avoid the need to resolve a URI included in a VIA entry of a reply, it is possible to add a “received” parameter to the VIA entry of the request with the numeric IP address of the sending entity, thus eliminating the need to resolve this URI after receiving a reply.

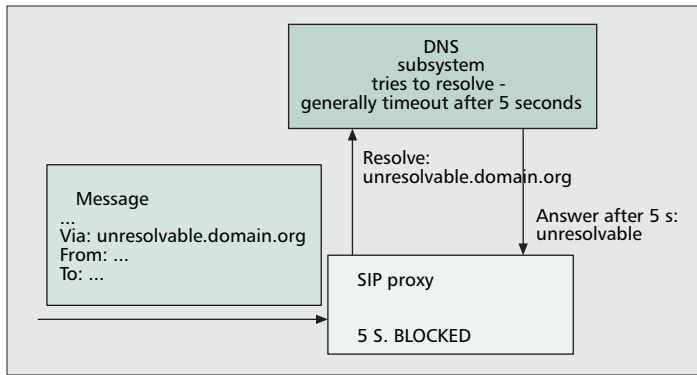
Nonblocking design: Another option is to design all requests to external servers as non-blocking. That is, after issuing a DNS request, the server would not wait until an answer for the request is generated, but would queue the request in an event queue, save the data of the transaction, break it and 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. This way, 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, which can also lead to a denial of service, but this time because of insufficient memory.

DNS caching: While the described countermeasures might reduce DNS dependency, in case of a real DNS attack advanced DNS caching is mandatory to successfully counter the attack’s effects. A DNS cache answers to DNS resolve requests from the SIP proxy. It saves the results of the latest DNS queries; if the SIP proxy tries to resolve the same address a second time, the stored result in the cache can be returned instead initiating another time consuming query. While different operating systems already provide DNS caches, for optimal usage in SIP they need extended functionality. For example, a SIP entity additionally uses DNS records to locate other proxies, including NAPTR/SRV [12] records. A general operating system (OS) DNS cache does not consider such records for caching. Furthermore, a dedicated SIP DNS cache needs a specialized refresh strategy, as it should preferably keep in its cache table the addresses of known and frequently used proxies, while general OS DNS caches follow a most recently used (MRU) policy. To be effective against DNS attacks, a combination of the cache with a SIP proxy with nonblocking parallel message processing capabilities is suggested. As an example, assume a SIP proxy with x parallel message processing queues. Then allow y message queues, where $x - y$ should be set reasonably low (e.g., 2) to concurrently resolve addresses through the cache. If no more than y message queues need to resolve an address concurrently, this is an indication of a DNS attack underway. In this case the cache only answers requests from its stored content, and returns an unresolvable message for any request that cannot be answered directly from the cache, thus preventing proxy blocking.

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 the packet’s source IP address to avoid DNS resolution overhead on sending replies.
3. If the deployment scenario allows it, authenticate statelessly to avoid memory exhaustion attack.



■ Figure 3. DNS blocking attack.

4. Make routine checks:

- Check for presence of viruses.
- Scan for well-known attack patterns including parser attacks.
- If max-forwards higher than local policy mandates, rewrite it to a lower value to prevent a request from exhausting server resources by loops.
- Drop all suspicious packets.

5. Optionally, establish transaction state. This helps to avoid burdening the server's resources with executing potentially expensive service logic for each retransmission received; however, do not establish the state if the request was not authenticated as this would allow anonymous attackers to exhaust memory quickly.

Additional considerations apply to processing REGISTER requests:

1. Use predictive nonces to ensure 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 (RFC 1918) for a public server (accepting them would allow an attacker to route requests to the private networks to which the SIP server is connected) or DNS names (they might cause a blocking server to block and a nonblocking server to run out of memory).

Summary

In this article we have presented an overview of possible approaches for mounting denial of service attacks. In general, we distinguish attacks on the transport and authentication protocols used by SIP from those that are 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 mount DoS attacks.

Denial of service attacks are 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 of distinguishing "friends" from "enemies." In addition to this black-and-white classification, there is a large gray area of "unwanted attacks," users trying to access server resources in good will but exhausting them through unknowledgeable misbehaving implementation.

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 as well and need to rely on solid performance.

Another observation is that there is no "one-size-fits-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 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.

Acknowledgments

This work has been partly conducted in the European Union funded COOP-005892 project SNOCER (<http://www.snocer.org>).

References

- [1] CSI and FBI, "2004 CSI/FBI Computer Crime and Security Survey," Mar. 2004.
- [2] J. Mirkovic *et al.*, *Internet Denial of Service: Attack and Defense Mechanisms*, Prentice Hall, 2005.
- [3] J. Rosenberg *et al.*, "Session Initiation Protocol," RFC 3261, 2002.
- [4] R. Fielding *et al.*, "Hypertext Transfer Protocol — HTTP/1.1," RFC 2616, June 1999.
- [5] A. B. Johnston, *SIP — Understanding the Session Initiation Protocol*, 2nd ed., Artech House, 2004.
- [6] 3GPP Tech. Spec. 3GPP TS 24.228 "Technical Specification Group Core Network; Signaling Flows for the IP Multimedia Call Control Based on SIP and SDP," 2003.
- [7] J. Rosenberg *et al.*, "STUN — Simple Traversal of UDP Through NATs," RFC 3489, Mar. 2003.
- [8] J. 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
- [9] J. Franks *et al.*, "HTTP Authentication: Basic and Digest Access Authentication," Internet Engineering Task Force, RFC 2617, June 1999.
- [10] J. Kuthan, "Comparison of Service Creation Approaches for SIP," Int'l. SIP Conf. 2000, Mar. 2000.
- [11] M. Handley *et al.*, "SIP: Session Initiation Protocol," RFC 2543 (obsoleted), Mar. 1999.
- [12] J. Rosenberg and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers," RFC 2543, June 2002.

Biographies

DORGHAM SISALEM (dorgham.sisalem@tekelec.com) is the director for strategic architectures at Tekelec. Previously he was leading the VoIP, IMS, and security activities at the Fraunhofer Institute Fokus, Berlin, Germany. He has been in the VoIP arena since 1995, and has been involved in various research and development activities in the areas of QoS, security, and multimedia communication. He holds a Ph.D. in engineering from the Technical University of Berlin, and has more than 40 papers in conferences and journals in the area of VoIP and QoS.

JIRI KUTHAN (jiri.kuthan@tekelec.com) graduated from the University of Salzburg in 1998. Between 1998 and 2004 he worked as a researcher at the Fraunhofer Institute Fokus. There he was involved in research projects related to security, scalability, and service creation in VoIP. He was also the leading force behind the development of the SIP Express Router. In 2004 he founded iptelorg, which provided VoIP solutions for large ISPs. Since 2005 he is AVP, NGN solutions at Tekelec involved in the design and realization of secure and scalable IMS solutions. He has various publications in the area of VoIP security and deployment experience.

SVEN EHLERT (ehlert@fokus.fraunhofer.de) is a senior researcher at the Fraunhofer Institute Fokus. He graduated from the Technical University of Berlin. His research interests are VoIP communication, security applications, and multicast protocols. He has participated in several VoIP security related projects, including denial-of-service detection and Skype analysis.