Fraunhofer Institute for Open Communication Systems

Denial-of-Service Detection and Mitigation for SIP Communication Networks

vorgelegt von
Diplom-Informatiker
Sven Ehlert

Von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
Dr.-Ing.
genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Jean-Pierre Seifert
Prüfer der Dissertation:	
1.	Prof. Dr. Thomas Magedanz
2.	Prof. Dr. Erwin Rathgeb

Tag der wissenschaftlichen Aussprache: 13.10.2009

Berlin 2009

D 83

Abstract

The Session Initiation Protocol (SIP) is the multimedia communication protocol of the future. Used for Voice-over-IP (VoIP), Internet Multimedia Subsystem (IMS) and Internet Protocol Television (IPTV), its concepts are based on mature and open standards and its use is increasing rapidly within recent years. However, with its acceptance as a mainstream communication platform, security concerns become ever more important for users and service providers. In this thesis we identify different attacks on SIP-based networks with the focus on *Denial-of-Service attacks (DoS) flooding attacks*. We evaluate SIP infrastructure for DoS attack possibilities and demonstrate a completely new attack which utilises a combination of the SIP and Domain Name Service (DNS) system. We propose three different DoS detection and mitigation schemes, including one to handle this particular SIP DNS attack. We also provide a first step into Distributed DoS mitigation by introducing a firewall pinholing scheme. Distributed DoS mitigation is only marginally addressed by current research works. We also evaluate the requirements for a self-sufficient and scalable SIP security framework, where attack countermeasures can be evaluated and tested. We use this framework for our solutions and validate their effectiveness for DoS mitigation. With these solutions, general SIP networks will be more robust against flooding DoS and Distributed DoS attacks.

Deutsche Zusammenfassung

Für Multimedia Kommunikation spielt das Session Initiation Protocol (SIP) eine immer wichtigere Rolle. SIP ist ein ausgereifter, offener Standard für Internet Telefonie (Voice-over-IP, VoIP), Internet Multimedia Subsystem (IMS) oder IP-basiertes Fernsehen (IPTV), dessen Verbreitung in den letzten Jahren stark angestiegen ist. Mit der breiten Akzeptanz von SIP als etabliertem Internetprotokoll werden Sicherheitsaspekte für Anwender und Serviceanbieter immer bedeutender. Diese Dissertation behandelt verschiedene Angriffe auf SIP-basierte Netzwerke mit dem Schwerpunkt Überlaufangriffe (Denial-of-Service, DoS). Wir evaluieren SIP-Infrastrukturen auf mögliche Angriffspunkte und stellen zudem einen komplett neuen DoS Angriff auf SIP-Netzwerke vor. Dieser nutzt eine Schwachstelle bei der Zusammenarbeit von SIP mit dem Domain Name Service (DNS) aus. Wir führen außerdem drei verschiedene Sicherheitsmethoden zur Erkennung und teilweisen Abschwächung von DoS-Angriffen ein, inklusive einer Methode als Gegenmaßnahme gegen den angeführten SIP-DNS Angriff. Weiter präsentieren wir einen ersten Schritt zur Sicherung vor "Verteilten DoS-Angriffen". Dieses Thema wurde bislang nur in ersten Ansätzen von der Forschergemeinschaft behandelt. Zusätzlich erörtern wir die Anforderungen an ein autarkes und skalierbares SIP-Sicherheitsframework zum Evaluieren, Testen und Validieren der vorgeschlagenen Sicherheitsmethoden. Durch die Anwendung dieser Methoden kann in Zukunft ein SIP-Netzwerk besser gegen DoS-Angriffe abgesichert werden.

Acknowledgements

This thesis was hard work, and definitely not possible without the help of others. In particular:

- My supervisors Prof. Dr. Thomas Magedanz and Prof. Dr. Erwin Rathgeb
- My girlfriend Trine
- My family
- My colleagues Dorgham Sisalem, Jens Fiedler and Yacine Rebahi
- My students Ge Zhang, Chengjian Wang, Andreea-Ancuta Onofrei, Erkan Güler, and Martin Becker

You know what you did – a big thanks!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The communication world is changing as technology progresses over time. In recent years[1] two highly successful innovations have dominated communication patterns and habits in people's daily lives: The *Internet* and *mobile communication over cellular networks*.

These technologies have been developed independently of each other, and are based on completely opposite media transportation systems. The Internet is deployed over a best-effort, packet-based channel-access system, while cellular networks establish a fixed bandwidth between users (circuit switching) so that they have a dedicated link for communication.

However, we can observe the trend that different communication platforms are converging together. The goal is here to deliver one all-IP based network platform combining the feature set of the Internet, mobile cellular networks and public switched telephone networks in one single platform. *Voice-over-IP* (VoIP) is a first step into this direction, as it allows standard telephony services over IP networks. In a further step, so-called *Next-Generation-Networks (NGN)* will provide the necessary infrastructure for converging communication networks. International standards for IP-based converged networks are being developed with the aim that these will be used worldwide in a way the Internet is already used today. VoIP is currently defined by the Internet Engineering Task Force (IETF), and a particular NGN in form of the Internet Multimedia Subsystem (IMS) [22] is being standardised for worldwide use. IMS is still under active development from a worldwide alliance, called the 3rd Generation Partnership Project (3GPP), and built on standardised IETF core protocols from the Internet world. Telecom-

[1]The press date of this document is January 31th, 2009

munication providers are changing their basic infrastructures to use VoIP and IMS as the core for their future networks. Worldwide providers are running field tests with IP-based converged networks.

1.1.1 Importance of SIP for Communication Networks

The main building block of NGN is the *Session Initiation Protocol (SIP)* [23] as the core protocol to establish, modify, and terminate the actual *sessions* between users. A session can be any network connection a user initiates that lasts for some time, e.g. a voice call or a live video stream.

Moreover, SIP is also the dominant protocol for VoIP today. Initially specified in 1999, SIP can be used for any session control between multiple participants. VoIP gained in popularity in recent years; more and more providers are offering VoIP services based on SIP. As a first step to establish future NGNs, telecommunication providers are exchanging their old circuit-switched PSTN infrastructure with all-IP, SIP-based voice networks. Even when the end-device at the customer might still be the same analogue phone from 20 years ago, the new underlying network has changed completely to an IP-based packet switched network using SIP. Some years ago, the dominant VoIP signalling protocol was H.323 [24], however in 2008 SIP usage has overtaken H.323 usage by far.

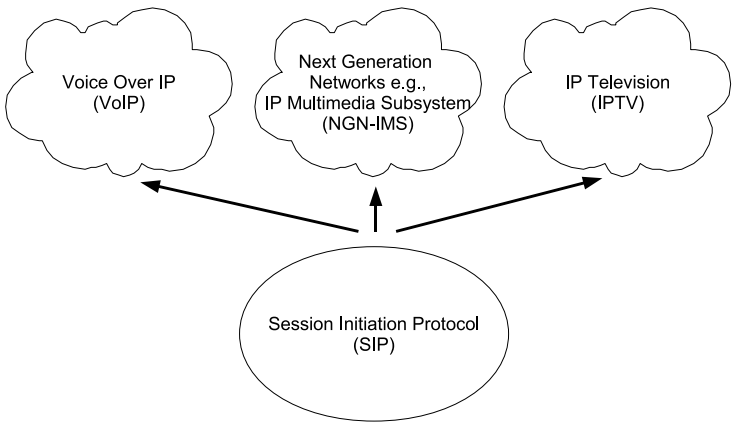

Figure 1.1: SIP is the basis of most future IP communication networks

SIP is also constantly extended for newer use cases [25], and is now widely deployed as the core protocol for Internet Protocol Television (IPTV) offerings. Hence, in future years it is likely that most users will uses communication technology which is at least partly built on top of SIP (Figure 1.1).

1.1.2 Complexity of SIP Networks

With different projected use cases and its claim to be the building block for all-encompassing communication networks, the complexity of SIP-based networks increase. Understanding SIP networks becomes thus ever more difficult, especially under two aspects: Complexity of the core SIP specification and complexity of SIP-based networks. This has a direct impact on network security.

SIP core specification complexity. If the document length of a protocol specification is an indication for the complexity of a protocol, then SIP ranges at *3rd place of all IETF protocol specifications*. Counting 256 pages, only the specification for Internet Open Trading Protocol [26] (276 p.) and Network File System v4 [27] (262 p.) are longer documents. This huge document size is necessary to explain SIP's multi-layered session control mechanism, i.e. it distinguishes between *transactions*, *dialogs* and *sessions* between participants, and for each layer individual parameters and procedures are defined, like the time-to-live value of a session or the re-transmission procedure for transaction messages. Additionally, SIP defines its own reliable transport mode over UDP connections, including four full *state machine specifications*. SIP is already defined in its 2nd revision, changing some core functionality (e.g. multiple request transaction matching), while still remaining fully compatible to the first published version. It can easily be extended through further additions, and more of 100 have already been specified [25], leading to an even complexer specification.

SIP network complexity. Combining the convolute SIP specification with a composite network of multiple entities serving different roles adds another level of complexity. Generally, SIP-networks are based on multiple different entities, e.g. in a VoIP setup, there are multiple different proxies, gateways to legacy networks and multiple end devices for users. Projecting this to a general NGN, it becomes even more challenging to distinguish between the different roles and connections between all involved SIP entities (see, for example, Figure 2.11) like network border gateways, home and visiting proxies, application or conversation servers. On top of that, components can be combined from different vendors, which might differ in implementation and might not support the same set of extensions.

1.1.3 SIP Challenges

To deploy SIP-based networks, it becomes also necessary to guarantee the correct operation of these networks. Multiples steps are necessary to achieve this, beginning from correct and standard compliant implementation for interoperability, over to a scalable design to handle higher loads of traffic and ultimately deploying redundancy systems to guarantee high service availability. Especially with more complex network setups a significant task will be network monitoring, to easily detect *network errors*, to anticipate *network bottleneck conditions* or to detect *security breaches*. It will only be a question of time until *attacks* will be actively directed towards the increasing number of SIP servers. As outlined, SIP is not straight-forward to setup. This increases the chance to have open security issues in the network, and with components available from different vendors, theses components are differently hardened against attacks.

Especially *Denial-of-Service (DoS)* attacks have been a major threat in the Internet world now for many years. Several widely known attacks have been successfully launched at popular Internet services, including DNS root server attacks and WWW server attacks. These attacks are rather easy to mount through the help of so-called Bot-networks (in the form of a *Distributed Denial of Service*, DDoS), which generate a large amount of traffic that the service under attack is unable to handle. Hence SIP, like any other IP-based protocol is also vulnerable to these common IP-based DoS attacks (like SYN flooding, link flooding) [28]. On top of that, possibilities exist to launch attacks that target directly SIP relevant methods and functions, which cannot be dealt with common DoS defence strategies. To counter such attacks, new algorithms tailored directly for SIP environments have to be deployed.

1.2 PhD Scope

The goal of this thesis is to illustrate common *SIP network vulnerabilities*, to indicate ways how to *exploit these vulnerabilities* and to present solutions to *detect and mitigate* attacks on SIP networks.

Security for SIP architectures encompasses a wide field, including privacy concerns, cryptography, unsolicited message handling (the spam problem), fraud or intrusion detection.

In this work we **focus on DoS attacks at the SIP protocol layer**. The SIP specification gives multiple opportunities to launch attacks. Following the taxonomy shown in Figure 1.2, we *concentrate on flooding at-*

tacks, both intentional and unintentional. Payload attacks can be handled by signature-based message checkers like [29], while flow tampering attacks can be prevented by applying message encryption (see Chapter 8). Flooding DoS attacks however, are still an unsolved problem.

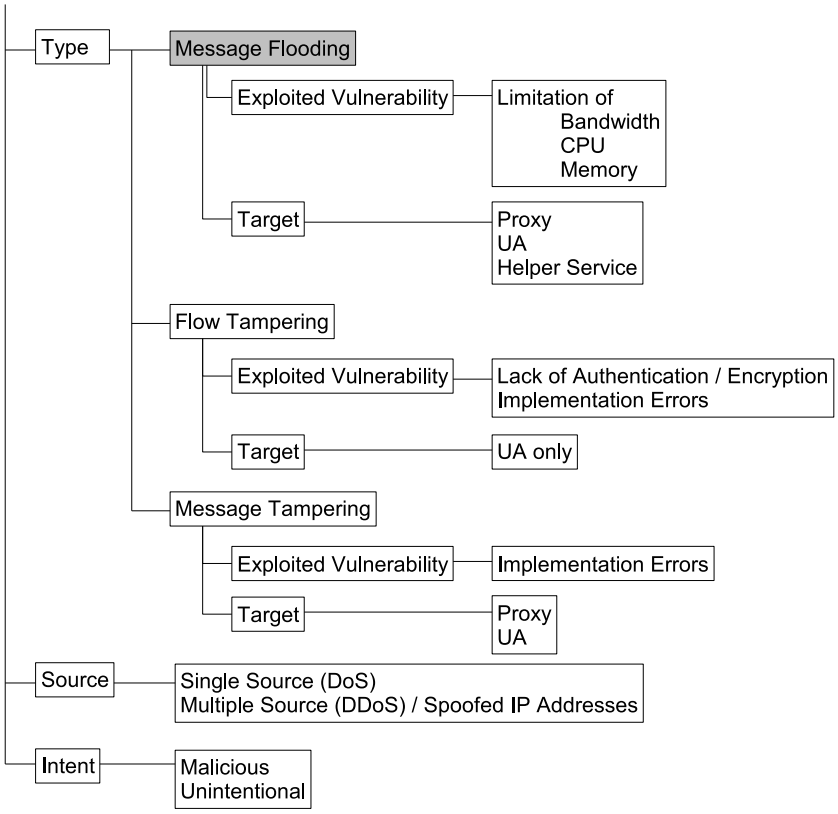

Figure 1.2: Taxonomy of SIP DoS attacks (Scope of this work highlighted)

Prevention has to be physically established within the network, but independent of other components. This is handled by a *dedicated security component* of the network, which *monitors relevant network traffic*. We analyse the requirements for such network security solutions for high flooding attacks, upon which a prototype security solution will be based.

As security threats vary for different setups with varying protocols, we concentrate on the SIP protocol and its dependencies. The defined solutions presented within are specially targeted at SIP networks, however, they might also be applicable to other types of networks. Also, the scope is directed to the SIP network level: Implementation errors or user errors are not considered. We focus on *dedicated SIP related DoS attacks*, and do not consider common IP-based attacks – such attacks and countermeasures have already

been extensively studied in computer science literature, for example see the work of Peng et al. [30].

1.3 Threat Impact and Requirements

Already in the specification of SIP [23] the possibilities of DoS attacks are mentioned. In 2005, the U.S. National Institute of Standards and Technology [31] and also analysts at DataMonitor [32] have determined DoS flooding to be a serious threat for SIP VoIP infrastructures. This is re-affirmed by threat predictions for 2009 from Georgia Tech [33]. Also, this topic is currently being evaluated and discussed in current conferences and trade shows (e.g. [34, 35]).

The European Telecommunications Standards Institute (ETSI) has released the technical specification describing the requirements for their SIP-based TISPAN NGN. This includes the requirement to provide mechanisms to mitigate DoS attacks [36]. In a threat analysis for ETSI TISPAN, two kinds of DoS attacks are identified. DoS attacks on publicly available interfaces are *considered a critical risk*. The authors rate the attack potential to be highly likely with a high impact on the attacked network. Also a minor risk are DoS attacks on non-publicly addressable interfaces. While such an attack is possible, only a low impact on the infrastructure is expected [37].

Arcor is a major German communication service provider. They are currently deploying large-scale SIP-based communication NGNs. Arcor also postulates DoS protection to be a *requirement for service providers*. They propose the usage of Session Border Controllers as the first line of defence with DoS protection features [38]. Sprint, a US provider is arguing similarly. They postulate that general IP-based DDoS detection methods should be enhanced to handle SIP VoIP attacks [39].

A report from 2008 shows that DoS attacks are already encountered in provider networks [40]. Currently, most attacks are still unintentional attacks due to configuration or setup errors. These attacks already had a severe impact on the infrastructure. Sipera, a developer of VoIP security tools rates DoS attacks to be the most important threat, followed by server hacking and service fraud [41].

Providers are already reacting to the new threat. Session Border Controllers, the combined security and firewall solutions of VoIP/IMS networks are gaining rapidly in popularity, according to iLocus studies [42], with revenues from SBC sells also climbing.

1.4 Related Works

1.4.1 General SIP Security Related Work

In the field of SIP security research, a lot research is being conducted on different fields. Although the work here focuses on security aspects, their scope is completely different from the work presented in this PhD.

VoIP spam handling has received a lot of attention. For example, Croft et al. [43] present a new approach to prevent voice spam by extending the call setup procedure. They include an anonymous mediator into the call path that manages the session setup. Mathieu et al. [44] present a new approach to detect and mitigate spam through a network-level entity. The basic function of the specific entity is to capture, filter and analyse the network traffic, passed from the equipment located in the edges of the network, in order to detect and mitigate voice spam calls. Rebahi et al. [45] present a reputation-based spam prevention scheme. Only callers that have been identified by trusted sources as legal one are allowed to call the destination.

As SIP is extensively used in IMS networks, security research has been provided, e.g. by protecting IMS application servers [46] or through security analysis of the provided authentication schemes [47].

On the field of cryptography, Salsano et al. [48] analysed the processing overhead of SIP authentication procedures. A new topic are SIP Fraud threats that are being discussed [49], but no general solution has yet been proposed. Zhang et al. continue this by presenting billing attacks on SIP VoIP systems [50].

1.4.2 SIP DoS Related Work

The general SIP DoS threat has been published in different works [51, 52, 19, 20]. Until now, a lot of research has been conducted to detect and prevent message flow tampering DoS attacks (from Figure 1.2). Such attacks include the disruption of an ongoing session by injecting false signalling messages, i.e. specially crafted BYE, REGISTER or CANCEL SIP messages. Wu et al. [53] have proposed a correlation framework to detect such attacks. They correlate SIP traffic with RTP traffic and by stateful monitoring they can detect e.g. if after a sent BYE message the media session correctly terminates. Advanced methods are proposed by Sengar et al. [54] and Bouzida et al. [55] which model a RTP traffic state machine. Luo et al. [56] examine the effects of the load that is generated at a SIP DoS target if the target itself incorporates DoS countermeasure methods (*Self-inflicting DoS*). Other research focuses on statistical analysis to differentiate flooding from regular traffic [57, 58, 59].

For SIP flooding detection, the common detection and prevention mechanism is to limit the number of requests during a given time interval to a fixed maximum value. This very basic method is the standard prevention mechanism for most available setups (e.g. the PIKE [60] extension for the common SIP Express Router (SER) [61]) and is also the general DoS protection method available in currently available *commercial SIP security solutions* like [62, 63].

A quantitative and qualitative analysis of related work is placed in Chapter 8.

Differentiation. When this work was started, there was only limited work in the SIP DoS area available (basically only [51, 57, 53]). Some of this PhD work was later the basis for an international SIP DoS security research project [64].

To differentiate from concurrent research efforts, this work gives a deep analysis of the threat situation with countermeasures [19, 20]. Also, we extend basic countermeasure methods [13, 7] to increase their effectiveness. We consider new attacks and countermeasures that have not been covered elsewhere [16, 14]. Also, especially when considering DDoS SIP attacks, there are currently only some methods to detect these attacks (e.g. [57, 59]), but no real mitigation methods have been established. We propose a first step to handle DDoS attacks [3].

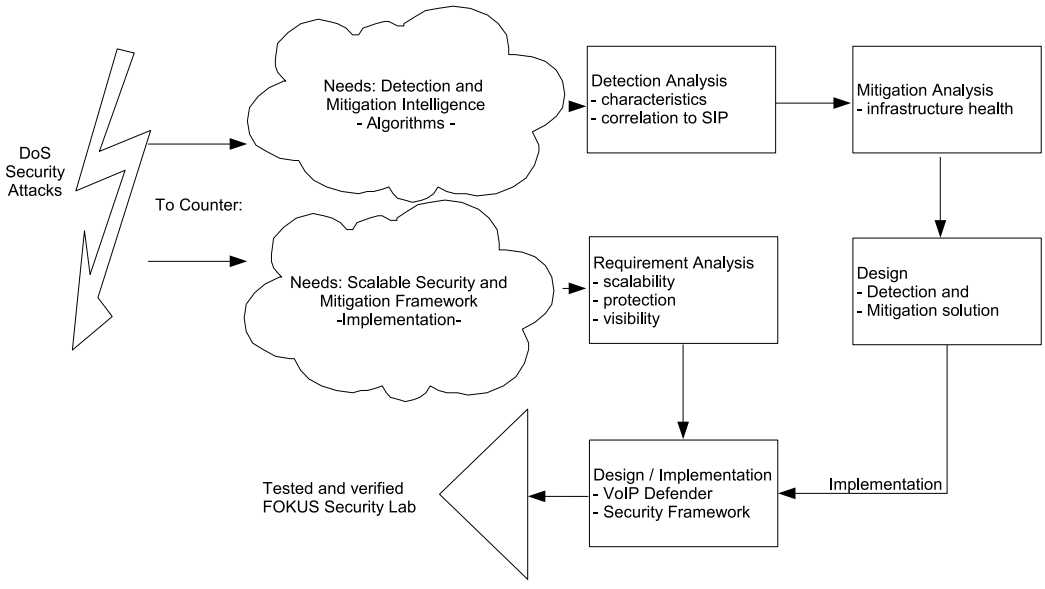

Figure 1.3: The workflow within this PhD

1.5 Methodology

To provide a DoS mitigation solution, a SIP infrastructure is analysed for possible weaknesses that could be exploited by an attacker. Based on this analysis, we model different potential DoS attacks that target exactly those weaknesses. Building upon existing defence solutions available in literature we utilise them where possible and advance them or redefine new methods targeted for the defined attacks. As there are many different possibilities to launch a DoS attack on SIP servers, several different countermeasures have to be considered, as it is *unlikely to define one solution to defend against all possible attacks*.

Flooding attacks are targeting resource depletion. Likewise, for countering these attacks one can not assume infinite resources. We therefore analyse the requirements for a dedicated DoS prevention system, especially under scalability concerns and implement a prototype system to test the defined monitoring and prevention solutions. All security solutions are then tested and verified within this security architecture.

A key point for successful network security is to know the actual status of the network, e.g. where do bandwidth problems and where does latency occur, etc. For this we design and implement a SIP network monitoring and visualisation tool.

Figure 1.3 visualises the necessary workflow steps. Figure 1.4 visualises the designed components within a SIP network.

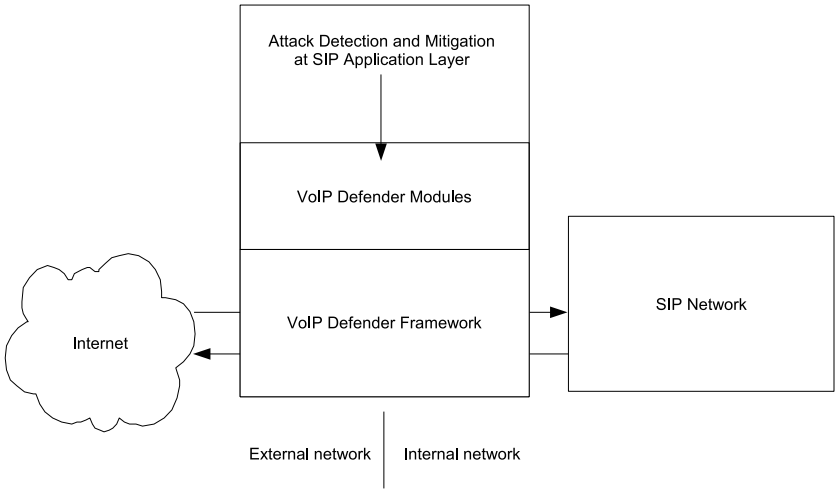

Figure 1.4: Placement of our SIP security framework within a common SIP network

1.6 Major Contribution

The major contributions of this thesis are the *analysis of the SIP DoS threat model* and the *definition of three countermeasure algorithms for DoS / DDoS detection and mitigation*. Within the threat analysis, we outline possible weaknesses, attack points and protection strategies in a SIP infrastructure with regard to DoS [19, 20, 4]. We later present different methods to overcome some of the threats. The mechanisms are in short:

1. A specification-based detection scheme that detects variations from regular traffic [13, 7].

2. A lightweight Distributed DoS prevention scheme based on firewall pinholing [3].

3. A DoS mitigation scheme for attacks on SIP helper services, especially on DNS servers. This is a new attack that has not been covered in literature before [16, 14, 6].

For the first two algorithms *patents* haven been applied at the European Patent Office.

Then, we deliver the requirements for a scalable SIP protection and monitoring solution, which is used to implement the previous algorithms[2]. We demonstrate this concept with a fully tested reference implementation that is actively being used in the Fraunhofer FOKUS network test beds. The framework is built for *extensibility* and can be used also for non-security related tasks [17, 11]. We prove this by using the framework as a SIP management solution. This solution is being actively used in the current FOKUS IMS Management solution OMACO, and already sold as a testbed to an Asian provider. Besides the SIP DoS research project, the research of of this work led to another closely related VoIP security project [65], which deals with a different type of message flooding, i.e. spam flooding.

All research efforts have been evaluated through experts in the field, as all steps of the work have been submitted to international conferences and journals (see the bibliography). Altogether, 19 publications have been already published or submitted for publication related to this work. Among these publications, six journal publications have been accepted for publication and two publications have been accepted at the major annual NGN security conference, IPTCOMM.

Relevant parts of this work are now cited by researchers in the field, including H. Schulzrinne, the inventor of SIP [66], T. Peng, one of the main DoS

[2]This part has been co-developed with Jens Fiedler

researchers [30], IBM Research [67], University of Tübingen [68], University of Texas [69] and others.

1.7 Thesis Structure

The thesis is structured in four parts. The first part gives an overview of the relevant technologies and where we are placed.

The second part is about the theoretical problem analysis. The requirements for protection are analysed. A main requirement will be a scalable monitoring framework. Vulnerabilities of SIP are elaborated and the attacks classified. Based on this we define our protection solution, with the architecture of the protection framework and the definition of countermeasure algorithms. We present three different methods that handle multiple DoS attack scenarios.

The third part is about the actual implementation of the security solution and its evaluation in our test bed.

Finally, in the last part we put our work into perspective in regard to related works by other researchers in the field, and give hints for future work for a full DoS prevention strategy based on multiple security algorithms.

Chapter 2

Background Information

2.1 Multimedia Communication with SIP

The Session Initiation Protocol (SIP) is an internet communication protocol designed to establish, maintain and terminate a *session* among two or more partners. A session here refers to any communication connection between partners that lasts some time. Examples are Voice-over-IP (VoIP) communication, video conferencing, instant messaging, interactive gaming or call forwarding.

SIP was first standardised by the Internet Engineering Task Force (IETF) in 1996 (RFC 2543 [70]) and was revised again in 2002 (RFC 3261 [23]), which is the current version until now.

According to the common IP paradigm SIP is an end-to-end signalling protocol [71] with all logic and state data solely stored in end devices, without support from the network. SIP is a text-based request-response protocol influenced mainly by the Hypertext Transport Protocol (HTTP) [72] and Simple Mail Transfer Protocol (SMTP) [73] of which it has borrowed some of its functionalities, e.g. Uniform Resource Identifiers (URIs) and many message header fields such as "To" and "From". It is highly extensible, and multiple RFCs and other documents define new extensions to the SIP specification [25]. Several books have been published explaining SIP in general and multiple communication applications using SIP [74, 75, 76].

SIP's target is session control, and not the transmission of actual session content. Description and transport of content are managed by other protocols, which are running in conjunction with SIP.

2.1.1 SIP Entities

A SIP communication network as defined by RFC 3261 is at least composed of four general types of logical SIP entities. Each entity has specific functions and can take a client role (initiates requests), a server role (responds to requests), or both. Logical entities can be implemented in one single physical device. For example, a network server working as a SIP proxy server can also function as a SIP registrar at the same time.

Following are the four types of logical SIP entities, which we will illustrate in more detail:

- **User agent**
- **Proxy server**
- **Redirect server**
- **Registrar**

User Agents

In SIP, a user agent (UA) is the endpoint entity. User agents initiate and terminate sessions by exchanging requests and responses. RFC 3261 defines the user agent as an application which contains both an user agent client and user agent server:

- **User agent client** (UAC) as a client application that initiates SIP requests.
- **User agent server** (UAS) as a server application that contacts the user when a SIP request is received and that returns a response on behalf of the user.

Some of the devices that can have UA functionality in a SIP network are: general workstations, dedicated devices like IP-phones or smartphones, network gateways or automatic operation devices like answering machines.

SIP Servers

SIP Servers are essential network elements that enable SIP endpoints to exchange messages, register their user location, and pass signalling traffic seamlessly between networks. SIP servers enable network operators to install routing and security policies, authenticate users and manage user locations.

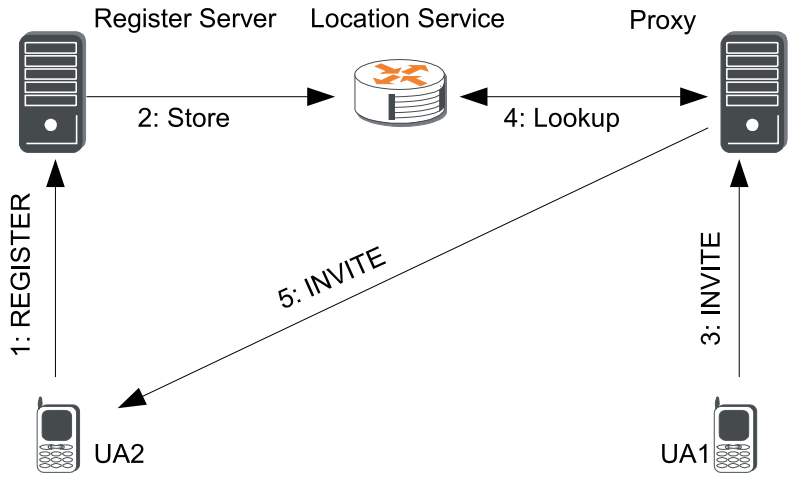

Figure 2.1: Registration and invitation process

The SIP standard defines three general types of server functionality - *proxy*, *redirect* and *registrar* servers. These standard functionalities can be used according to the needs of the specific implementation, and further applications are possible.

With the advance of SIP, server logic has become increasingly complex. SIP servers need to deal with varying network topologies (such as public Internet networks, cellular networks, broadband residential networks), complex routing policies, security and SIP extensions. SIP servers therefore are often required to handle high throughput rates and yield real-time performance and scalability with low delay.

Registrar server A registrar server accepts requests from a user who wants to make himself available to the network. It processes its registration information and stores it into a location service for further reference.

The registrar processes user requests for a specific set of domains. It uses a location service – an abstract location database – in order to store and retrieve location information. The location service may run on a remote machine and may be contacted using any appropriate protocol, see Figure 2.1. The SIP standard leaves this decision to the implementation. Some implementations may co-locate the location service and the registrar server on the same machine.

Redirect server A redirect server receives SIP requests and responds with redirection responses, thus directing the client to contact an alternate set of

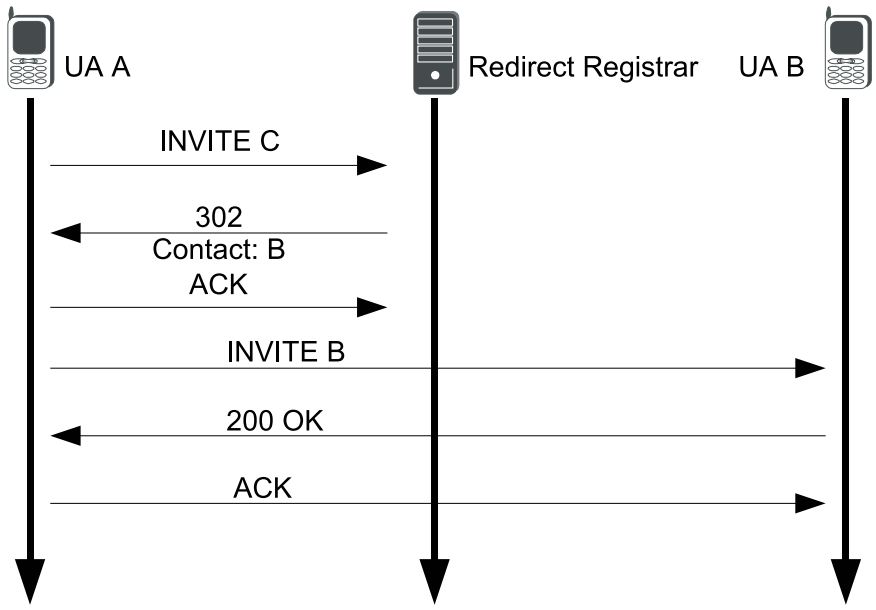

Figure 2.2: Request redirection

SIP addresses.

The scenario in Figure 2.2 illustrates a redirection scenario. Redirection allows servers to push back routing information for a request in a response to the client, thereby aiding in locating the target of the request, while taking themselves out of the loop of further messaging for this transaction. Redirect servers typically are not aware of the state of dialogs (calls, subscriptions), only of the state of the individual transactions they are handling, making them transaction-stateful elements. Redirection is designed as a simple and quick procedure, allowing for redirect servers to be highly scalable and to yield high-performance.

Proxy server SIP proxies perform general routing operation in the network by forwarding SIP requests to user agent servers and SIP responses to user agent clients.

SIP proxies can thus bee seen as general IP routers, albeit at the SIP application level. However, SIP proxies employ routing logic that is typically more sophisticated than just automatically forwarding messages based on a routing table. The SIP standard allows proxies to perform actions such as request validation, user authentication, request forking, address resolving, or to cancel pending sessions.

The versatility of SIP proxies allows the network administrator to use proxies for different purposes and in different locations in the network (such as edge proxy or core proxy). With this versatility a variety of proxy policies are possible, such as allowing calls only for authenticated users that have no debt to the network service provider.

A proxy server is designed to be mostly transparent to UAs. Proxy servers are allowed to change messages only in specific and limited ways. For example, a proxy is not allowed to modify the body of a session initiation request. Apart from a few exceptions, proxies cannot generate requests on their own initiative.

Figure 2.3 depicts SIP entities and their relationships.

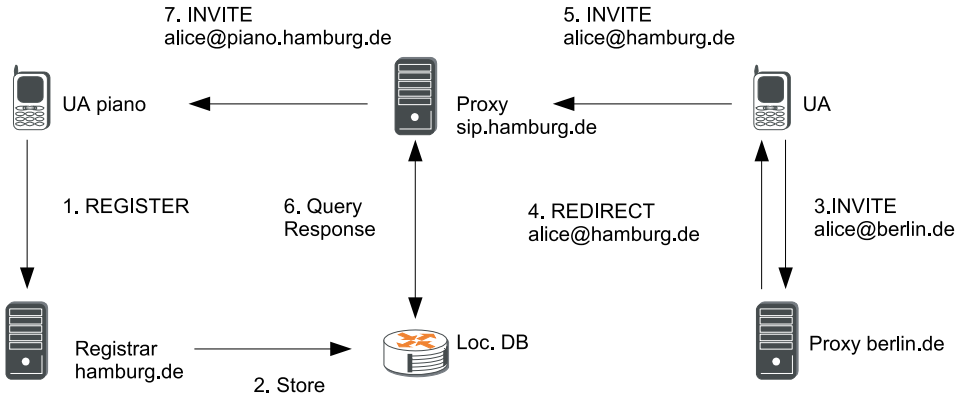

Figure 2.3: Entities forming a SIP communication network.

2.1.2 SIP Structure

SIP is structured as a layered protocol. The protocol behaviour is described by four layers:

1. Syntax and encoding
2. Transport layer
3. Transaction layer
4. Application layer

The lowest layer of SIP is its syntax and encoding, as explained in the following section.

The second layer is the transport layer. It defines how a client or server sends requests and receives responses over the network. All SIP elements contain a transport layer.

The third layer is the transaction layer, where sent messages and responses are correlated. The transaction layer is further outlined in Section 2.1.4.

The highest layer is the transaction user (TU). Most of the SIP entities, except some passive proxies implement one. A TU creates client transaction instances and passes requests down to them.

2.1.3 SIP Message Format

SIP messages are encoded in plain text, using the UTF-8 charset [77]. A message can either be a *request* (SIP method) or a *response* (SIP state code). A SIP message consists of three parts:

- **Start line** - Every SIP message begins with a start line. The start line conveys the message type (method type in requests, and response code in responses) and the protocol version. For requests, the start line is called Request-Line and Status-Line for responses.

  The *Request-Line* includes a Request URI, which indicates the user or service to which this request is being addressed. There are six basic SIP methods defined in the core SIP protocol specification, as visible in Table 2.1.

  The *Status-Line* holds the numeric *Status-Code* and its associated textual phrase, as visible in Table 2.2.

- **Headers** - SIP header fields are used to convey message attributes and to modify message meaning. They are similar in syntax and semantics to HTTP and SMTP header fields and always take the format $\langle name \rangle$: $\langle value \rangle$.

- **Body (content)** - A message body is used to describe the session to be initiated (for example, in a multimedia session this may include audio and video codec types, sampling rates, etc.), or alternatively it may be used to contain opaque textual or binary data of any type which relates in some way to the session. Message bodies can appear both in request and in response messages. SIP makes a clear distinction between signalling information, conveyed in the SIP start line and headers, and the session description information, which is outside the scope of SIP. Common body types are, e.g. Session Description Protocol (SDP, see

Table 2.1: SIP request methods

Method	*Description*
INVITE	Initiates a call, changes call parameters (re-INVITE)
ACK	Confirms a final response for INVITE
BYE	Terminates a call
CANCEL	Cancels searches and ringing
OPTIONS	Queries the capabilities of the other side
REGISTER	Registers with the location service

Table 2.2: SIP response codes

Class	*Description*
1xx	Provisional messages - used by the server to indicate progress, but they do not terminate SIP transactions (searching, ringing, queuing, etc).
2xx	Success answers
3xx	Redirection, forwarding messages
4xx	Request failure (client mistakes)
5xx	Server failures.
6xx	Global failures (busy, refusal, not available anywhere)

Section 2.1.7) or Multipurpose Internet Mail Extensions (MIME) [78] bodies.

SIP Message Headers

SIP requests and responses contain headers following the request or status lines. Those headers are used to transport the information to the SIP entities, some of which are specific to requests and some of which to responses. A header is composed of the header name, followed by a colon and a header value. The main header fields are:

- **To :** the address of the recipient of the request.
- **From :** the global address of the caller.

- **Contact :** the current URI where the sender can be contacted.
- **CSeq :** an integer and a method name, where the integer part of this header is used to detect the non-delivery of the message or out-of-order delivery messages. At the beginning of a transaction, it is randomized and upon arrival of a new message it is incremented by one.
- **Call-ID :** a unique identifier for each call and contains the host address. This header identifies a particular invitation and is the same for all of the messages within one transaction.
- **Via :** used to store all entities where this request has passed so far. It contains the transport protocol, such as UDP and TCP, and also the request route in order to detect the routing loops and to route the responses towards the client who generated the request.
- **Route :** set of hosts a messages has to pass before reaching its final destination.
- **Record-Route :** set by intermediate entities to indicate that they will be included in further message passes.

Table 2.3 gives examples for common SIP header fields.

2.1.4 Dialogs and Transactions

In the SIP session context, two events are defined that last a certain amount of time, which are *dialogs* and *transactions*.

A dialog is a peer-to-peer SIP relationship between two UAs that persists for some time. A dialog is established by SIP messages, such as a 2xx response to an INVITE request. Such a dialog can be terminated with a BYE message at a later time. Depending on the type of a session, dialogs can exist for a considerable amount of time, e.g. during a voice call or a video transmission.

SIP transactions consist of a single request and any responses to that request, which include zero or more provisional responses and one or more final responses. As such, any dialog is created from individual transactions.

The relationship between SIP UAC/UAS transactions is pictured in Figure 2.4.

Transactions are modelled within the SIP RFC through state machines. The state machine defines which events are valid and what and how the state will change if a certain event occurs. State machines are defined both for a UAC and a UAS. Additionally, they are different for INVITE messages

Table 2.3: Common header fields

Header Name	C.	Examples
Authorization		Authorization: Digest username="alice", realm="example.org", nonce="4684b75c60e0378c25b2419b94def5a", uri="sip:example.org", response="3bcbd89de1bf5049bf618e047d", algorithm=MD5
Proxy-Authorization		
Call-ID	i	Call-ID: f81d4fa-a765-00a0@example.com
Contact	m	Contact: <sip:alice@atlanta.ccom>;expires=3600
Content-Length	l	Content-Length: 320
CSeq		CSeq: 98
Expires		Expires: 3600
From	f	From: "Alice" <sip:Alice@example.org>;tag=clzba
To	t	
Max-Forwards		Max-Forwards: 70
Record-Route		Record-Route: <sip:server10.biloxi.com;lr> , <sip:bigbox3.site3.atlanta.com;lr>
Route		Route: < sip:bigbox3.site3.atlanta.com;lr> , < sip:server10.biloxi.com;lr>
Proxy-Authenticate		WWW-Authenticate: Digest realm="example.org", nonce="4684b75c60e0378960c25988b3dbdef5a"
WWW-Authenticate		
Via	v	Via: SIP/2.0/UDP 213.192.59.75; rport;branch=z9hG4bKsjlhytns

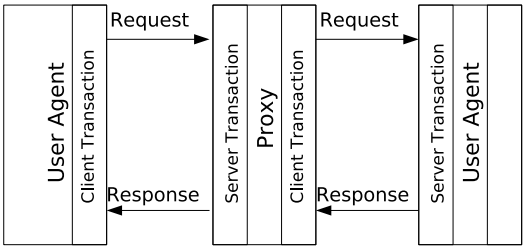

Figure 2.4: UAC / UAS transaction relationships

and Non-INVITE messages, thus resulting in altogether four different state machines:

1. UAS INVITE state machine
2. UAS Non-INVITE state machine
3. UAC INVITE state machine
4. UAC Non-INVITE state machine

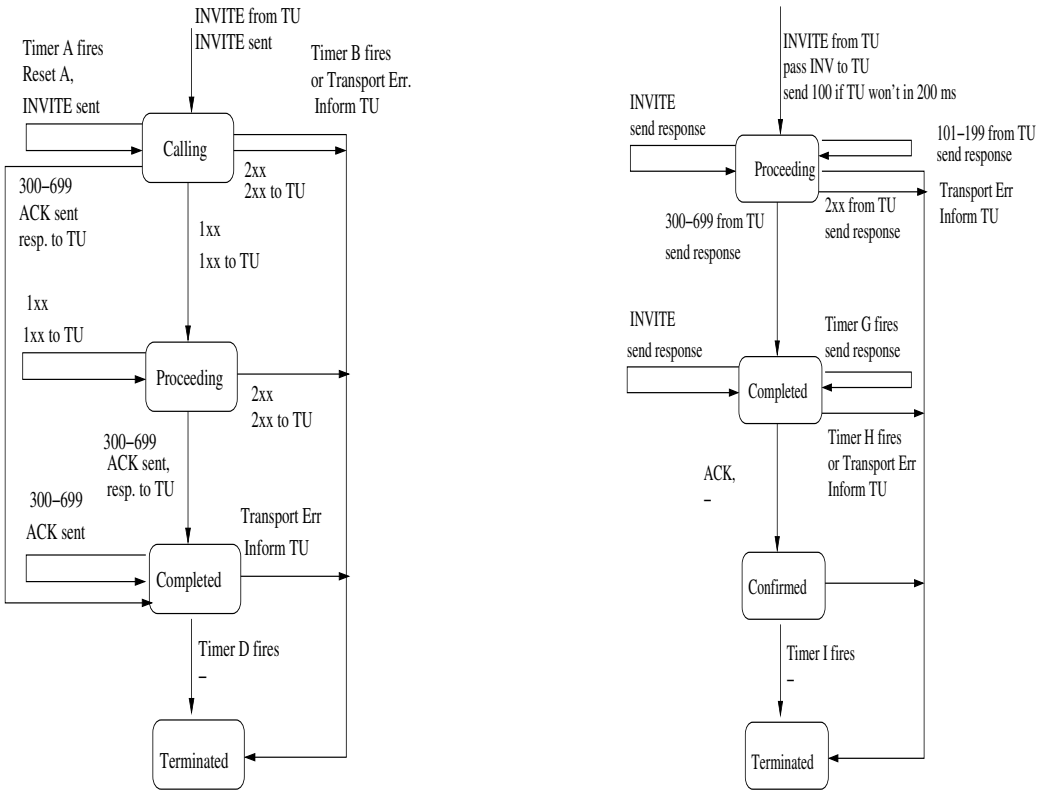

Figure 2.5: INVITE client (left) and server (right) transactions

Figure 2.5 and 2.6 present the state diagrams of the four state machines as specified in RFC 3261.

SIP proxy servers may serve in stateless, stateful mode or in a mixed form of these two modes. A stateful proxy is also known as transaction stateful proxy. In stateless mode a proxy simply forwards every request it receives downstream and every response it receives upstream, while a stateful proxy must maintain the client and server transaction state machines during the

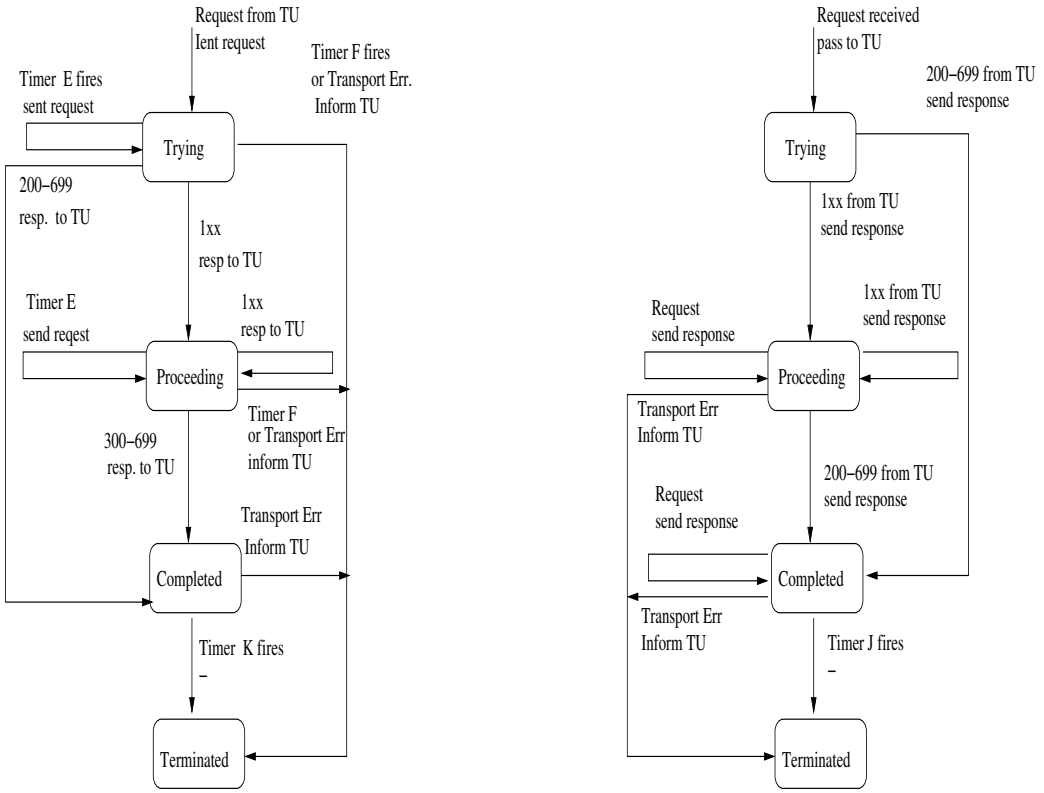

Figure 2.6: Non-INVITE client (left) and server (right) transactions

processing of a request. After receiving a new request a transaction context will be created by a stateful proxy. After forwarding the request the context retains in the server and the following requests and responses will be handled based on this context.

2.1.5 Protocol Operation

SIP's purpose is to manage sessions between users. This includes the management of user location, availability and capabilities and also session setup, handling and termination. We give some examples to illustrate this.

Session Establishment and Interaction

Figure 2.7 shows the interaction between a user agent client (UAC) and a user agent server (UAS) during trivial session establishment and termination.

Session Establishment – Call Flow

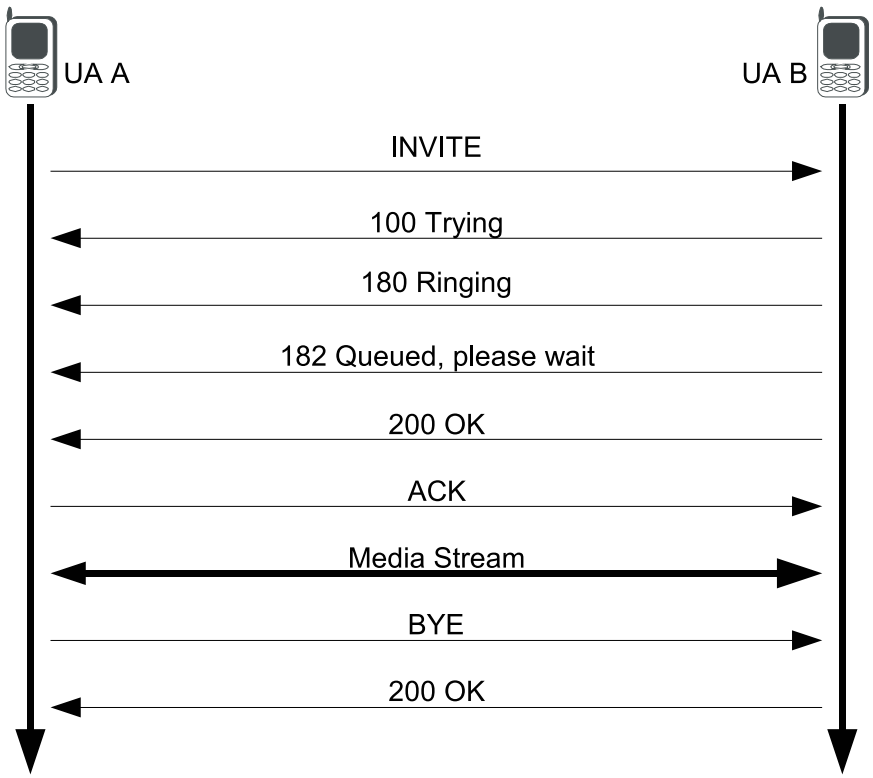

Figure 2.7: SIP session establishment and call termination

1. The calling User Agent Client sends an INVITE message to Bob's SIP address: sip:bob@acme.com. This message also contains an SDP packet describing the media capabilities of the calling terminal.

2. The UAS receives the request and immediately responds with a 100 (Trying) response message.

3. The UAS starts "ringing" to inform Bob of the new call. Simultaneously a 180 (Ringing) message is sent to the UAC.

4. The UAS sends a 182 (Queued) call status message to report that the call is behind one other call in the queue.

5. Bob picks up the call and the UAS sends a 200 (OK) message to the calling UA. This message also contains an SDP packet describing the media capabilities of Bob's terminal.

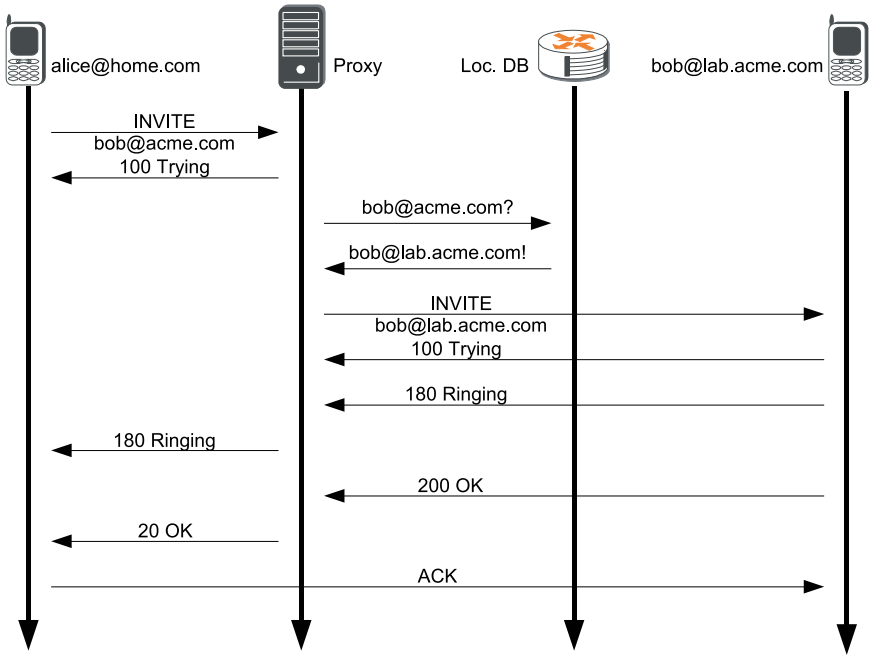

Figure 2.8: Call proxying scenario

6. The calling UAC sends an ACK request to confirm that the 200 (OK) response was received.

Session Termination – Call Flow

1. The caller decides to end the call and "hangs-up". This results in a BYE request being sent to Bob's UAS at SIP address sip:bob@lab.acme.com.

2. Bob's UAS responds with a 200 (OK) message and notifies Bob that the conversation has ended.

Session Proxying

Figure 2.8 shows the call set-up between two user agents with the assistance of an intermediate proxy server.

1. An INVITE message is sent to bob@acme.com, but finds the proxy server sip.acme.com along the signalling path.

2. The proxy server immediately responds with a 100 (Trying) provisional response.

3. The proxy server looks-up Bob's current location in a location service, e.g. by database look up.
4. The location service returns sip:bob@lab.acme.com i.e. Bob's current location.
5. The proxy server decides to proxy the call and creates a new INVITE message based on the original INVITE message, but with the request URI in the start line changed to bob@lab.acme.com. The proxy server sends this request to the UAS at lab.acme.com.
6. The UAS responds first with a 100 (Trying).
7. The UAS responds with a 180 (Ringing) response.
8. The proxy server forwards the 180 (Ringing) response back to the calling UA.
9. When the call is accepted by the user (for example, by picking up the handset) the UAS at lab.acme.com sends a 200 (OK) response. In this example, Bob's UAS inserts a Contact header into the response with the value bob@lab.acme.com. Further SIP communication will be sent directly to it and not via the proxy server. This action is optional.
10. The proxy forwards the 200 (OK) response back to the calling UAC.
11. The calling UA sends an ACK directly to Bob's UA at the lab (according to the Contact header it found in the 200 (OK) response).

2.1.6 SIP Security Mechanisms

The SIP specification does not define its own specific security mechanisms. Instead, it utilized other well-known internet security mechanisms. Security mechanisms can be provided in a hop-to-hop or end-to-end fashion (see Figure 2.9). RFC 3261 defines four security mechanisms:

- HTTP digest authentication
- Transport Layer Security (TLS)
- IP Security (IPsec)
- Secure MIME (S/MIME)

An end-user has the ability to choose its security mechanism according to its current needs [79].

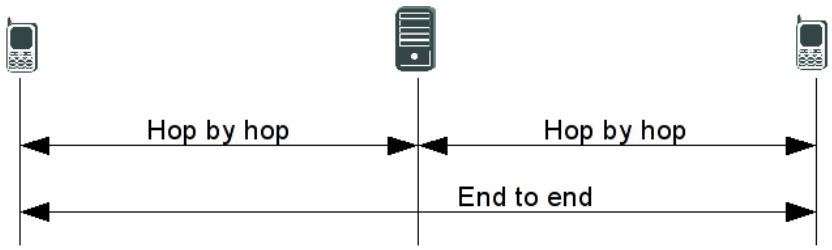

Figure 2.9: Hop-by-hop vs end-to-end security

User Authentication Using HTTP Digest

SIP utilizes HTTP digest authentication [80] to verify the identity of users. Based on the configuration of the SIP server, a server can enforce sender authentication before processing SIP requests. This authentication can be applied to certain requests only, certain users or requests coming from certain proxies or redirect servers.

General proxies generate slightly different authentication requests than registrar servers:

- Proxies generate a 407 (Proxy Authentication Required) response with an additional Proxy-Authenticate header.
- Registrars and redirect servers use 401 (Authentication Required) responses with a WWW-Authenticate header.

A SIP server manages information about users eligible for using this server. The information is in the form of user login name and password, which is checked using hashing algorithms.

Hop-by-Hop Encryption Using TLS and IPsec

SIP messages can be encrypted hop-by-hop with the use of Transport Layer Security protocol (TLS) [81]. Authentication for the corresponding network elements during the handshake procedure is possible by exchanging their certificates. TLS is a widely adopted protocol, especially for secure web traffic. It runs above TCP/IP and below higher-level protocols such as HTTP, FTP or SIP and consequently the TCP header is not encrypted. TLS does not run over UDP, which is currently the predominant transport protocol for SIP. Also, keeping up many TCP connections simultaneously causes additional load on SIP servers. SIP provides a notation to request a secure connection with the SIPS URI, e.g. sips:user@siphome.net.

Support for TLS in SIP UAs is increasing in recent years.

Alternatively to TLS IPsec may be used to encrypt messages in SIP. The IP Security (IPsec) [82] suite provides a set of services to protect IP packets from such attacks. IPsec can provide confidentiality, integrity, data origin authentication services as well as traffic analysis protection. Introducing IPsec in Internet telephony can safeguard signalling and data from network vulnerabilities provided that some sort of trust (e.g. pre-shared keys, certificates) has been established a-priori between the communicating parties.

End-to-end Encryption Using S/MIME

SIP messages are capable of carrying MIME bodies, and the MIME standard includes mechanisms for securing MIME contents to ensure either integrity or confidentiality by means of the multipart/signed and application/pkcs7-mime MIME types [83]. S/MIME provides a set of functionalities and SIP utilizes two of them: Integrity plus authentication tunneling and tunneling encryption. However, this solution mandates the deployment of a global S/MIME Public Key Infrastructure (PKI). Such an infrastructure has not been established until today.

Even with S/MIME, no fully end-to-end encryption is possible. Some headers like VIA and Route are needed for Routing purposes in the network and thus cannot be encrypted.

2.1.7 Session Description Protocol

Session Description Protocol (SDP) [84] is commonly used to describe the multimedia sessions in real time in conjunction with the SIP protocol. Despite its name, it does not actually define a network protocol, as it does not have a transport mechanism or any kind of parameter negotiation. Instead it defines a simple message format. The usage of SDP in SIP message bodies allows for an "Offer-Answer" mechanism to transfer the description of the multimedia session and the negotiation of the necessary parameters required for this multimedia session.

SDP is also a text-based protocol like SIP and having a set of lines of text of the form **type = value**. The types are identified by a single letter and the format of the value depends on the type. Table 2.4 lists the types defined in the SDP protocol.

As a SIP example, if a user wants to start a multimedia session with a participant, then the caller sends an INVITE message including the session description in its payload. Upon receipt of the invitation together with the session description, the callee looks for the offered session description and decides if it accepts the sent offers in the session description and returns

Table 2.4: Types defined by SDP

Type	Description
v	Protocol version
o	Owner of the session and session identifier
s	Session name
i	Session information
u	URL containing a description of the session
e	E-mail address to obtain information about the session
p	Phone number to obtain information about the session
c	Connection information
b	Bandwidth
z	Time zone adjustments
k	Encryption key
a	Attributes
t	Time interval when the session is active
r	Repetition time
m	Transport protocol information (media line)

a 200 OK response to the caller including its own offers. In this fashion with an offer-answer mechanism, the parameters of the multimedia session is negotiated between the participants.

2.1.8 Real-time Transport Protocol

The Real-time Transport Protocol (RTP) [85] provides a standardized packet format for the delivery of audio, video or simulation data over the Internet. The real-time data is broken into pieces and encapsulated into packets to be delivered. The receiver of those packets has to put them in the correct order, so that the original data is recovered. For this, the protocol provides the following functionalities:

- **Payload type identification :** It indicates what kind of media content is carried.
- **Sequence numbering :** It enables the receiver to order the received RTP packets.
- **Time stamping :** It enables the recovery of the original timing relationship of the data contained in the payload, implicating that the synchronization between the video and audio data is performed if they

both co-exist in a multimedia session. It also handles the delays (jitters) of the packets within a same stream.

Besides the functionalities listed above, a control protocol, namely Real-time Transport Control Protocol (RTCP), enables monitoring of the RTP packets and controls the synchronization of the streams with the help of the timestamps in the RTP packets. The RTP protocol does not ensure any resource reservation and does not guarantee any Quality of Service (QoS), this has to be handled by other means.

As a SIP example, if a user wants to start a multimedia session with another participant, the parameters required for the RTP transmission is negotiated and carried in the payload of the SIP message before a media communication between the the caller and the callee.

2.1.9 SIP Alternatives

Besides SIP, there exist different alternatives for session (especially voice) handling over IP networks.

The most common standard is H.323 [24] from the International Telecommunication Union - Telecommunication Standardization Sector (ITU-T), it is implemented in various devices for voice and video conferencing. However, due to SIP's popularity, H.323 support is declining.

Skype [86] is a proprietary protocol used in the popular VoIP application with the same name. It provides voice, chat and video service over an encrypted protocol between two hosts. It is based on the Kazaa Peer-to-Peer (P2P) file transfer protocol. Much of Skype's popularity is gained from its ease of use and simple configuration. The protocol specification is disclosed, so several attempts have been conducted to reverse-engineer the protocol [87, 18].

Other, less-known protocols are Inter-Asterisk Exchange (IAX) [88] used in the free Asterisk PBX, or Cisco's proprietary Skinny Call Control Protocol (SCCP) [89].

2.2 SIP Network Application

The SIP specification does not define a specific application for the protocol. Currently, the most prominent applications are *VoIP networks* and so-called *Next Generation Networks*.

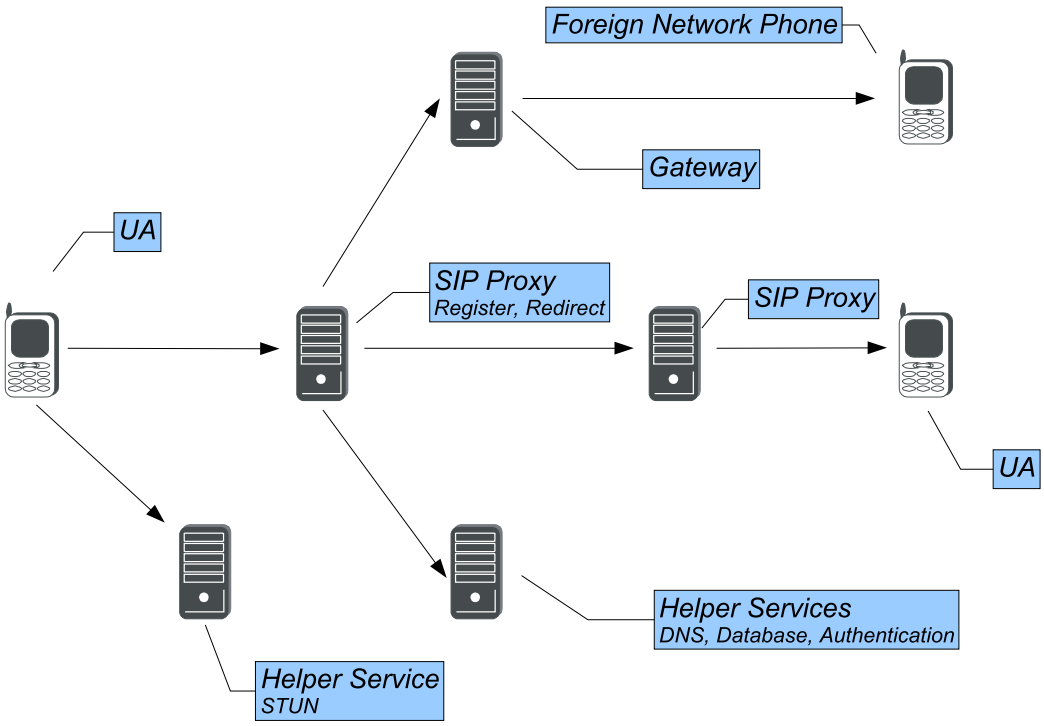

Figure 2.10: Overview of a SIP-based VoIP network

2.2.1 IP-based Telephony Networks

The most prominent usage of SIP is currently in IP-based telephony systems (Voice-over-IP, VoIP). Using standard SIP components, users can place voice calls between each other using common IP networks. There are a lot of different voice user agents available, both in hardware and software (softphone) for all major operation systems. Besides 2-party voice communication, often further features are available, like gateway functionality to interconnect with the Public Switched Telephony Network (PSTN), multi-party conferencing, instant messaging or presence information. A common setup of a VoIP network can be seen in Figure 2.10.

In Germany, multiple providers are currently offering SIP-based VoIP services, including Sipgate, 1&1, freenet and T-Online. VoIP networks are also more and more replacing legacy PBX setups in enterprises.

2.2.2 Next Generation Networks

Since some recent years one can observe the trend that different mobile and fixed communication platforms are converging together. The goal is

here to deliver one all-IP based network platform combining the feature set of the Internet, mobile cellular networks and PSTN networks in one single platform (Fixed-Mobile Convergence, FMC). So-called *Next-Generation-Networks (NGN)* will provide the necessary infrastructure for such converging communication networks. International standards for NGN are being developed with the aim that NGN are to be used worldwide in a way the Internet is already used today. The Internet Multimedia Subsystem (IMS) [22] is such a standardized NGN for worldwide use. It is still under active development from a worldwide alliance, called the 3rd Generation Partnership Project (3GPP).

The key features of IMS are multimedia session management, guaranteed Quality-of-Service (QoS), secure network access and service control.

IMS is based on core IP protocols, with the fundament on SIP for session control. Other important protocols in IMS are Diameter [90] for AAA (Authentication, Authorization, and Accounting) service, or RTP for media transport.

A core IMS setup is depicted in Figure 2.11. IMS defines many different roles, with the most prominent ones are:

- CSCF (Call Session Control Function). This is the core component of an IMS network and manages all sessions in the network. Its role is subdivided into the parts: The Proxy-, Serving, and Interrogating-CSCF (P-, S-, I-CSCF), which roughly correspondent to the SIP terms incoming proxy, registrar, and outgoing proxy.

- HSS (Home Subscription Server). This is the main user database of IMS subscribers (user location database).

- AS (Application Server). The actual services, like VoIP, instant messaging or media streaming are located here.

- MRF (Media Resource Function). Provides media-related functions, e.g. playing announcement tones, or voice stream mixing.

- GF (Gateway Function). Generic term to enclose gateways to other networks, e.g. to the PSTN.

IMS is defined in revisions to adopt to new scenarios and changing requirements. It first appeared in 3GPP Release 5, the most current revision is Release 8.

The European Telecommunications Standards Institute (ETSI) is defining the European View of NGN in its Telecoms & Internet converged Services & Protocols for Advanced Networks (TISPAN) standardization body.

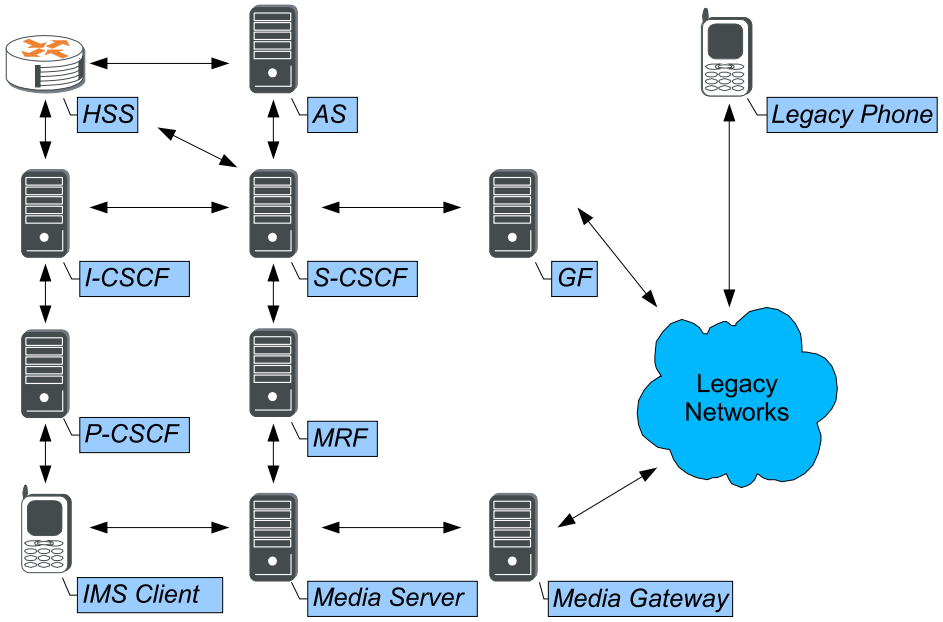

Figure 2.11: Overview of an IMS network

TISPAN [91] is based on the IMS specification, and extends it among other to DSL access, non SIP-based applications and IPTV services. TISPAN's specifications are also updated over time. The currently active specification is Release 2.

2.3 Denial of Service Attacks

Denial-of-Service (DoS) attacks [28] are a class of network attacks performed to interrupt or terminate *applications*, *servers*, or even *whole networks*, with the aim of disrupting legitimate users' communication. Disruption targets are web browsing, listening to online radio, or even interrupting essential communication, e.g. power plant network control traffic. DoS attacks are commonly performed intentionally and in most cases difficult to counter. In many cases it is only possible to *mitigate*, but not to completely *prevent* the attack.

2.3.1 Motivation for DoS Attacks

DoS attacks can have different forms, and they can also be differently motivated. Generally, users might like the feeling of having power to force their will onto others by disrupting some sort of their communication. This is

common among younger internet users that just take available DoS tools and launch attacks on different servers (so called *script-kiddies*).

Another motivation can be to disturb the business model of internet companies. For example, a well-known attack launched in 2000 against the Yahoo web site caused it to be unavailable for several hours. During this time no one was able to access the Yahoo web site. Yahoo lost around US$ 500.000 during this outage, as nobody was able to click on the Yahoo web banners [92].

Only recently have DoS attacks reached the level of international security. The huge attack launched on Estonian web pages in 2007 caused several critical service outages [92]. With the continuing growth of Internet communication in essential state services (e-voting, e-government, power plant control, traffic control, ...) whole state operations might be at risk from DoS attacks in the future.

2.3.2 DoS Targets

There are two common strategies to launch a DoS attack, by either exploiting a *software vulnerability* or by *depleting resources* at the target host.

Vulnerabilities in Application, Protocol Stack or Operating System

One common way to achieve a DoS attack is to exploit vulnerabilities in a software component on the target machine. This includes vulnerabilities in application servers, network stacks, or general operating system vulnerabilities. Vulnerabilities in huge projects are a common case, as it is impossible to predict every situation where software is deployed.

To exploit the vulnerability, an attacker sends a messages crafted in a specific way that takes advantage of that given vulnerability. By launching a *Nuke attack* on e.g. the TCP/IP stack, the whole system might crash eventually.

Examples. One popular example of such an attack is the *Ping of Death* attack [93], which is performed by sending an oversized Internet Control Message Protocol (ICMP) [94] echo request packet to the victim host. ICMP messages are sent within IP datagrams and de-multiplexed by IP for further handling by ICMP. Regular echo request packets only have a length of 64 octets, but no formal size limitation exists. Thus, an ICMP message can be transported in two or more IP packets using fragmentation and thus will be reassembled before passed to ICMP. In this way it is possible to send an echo request whose content comprises more than the maximum allowed size

of 65507 octets. Depending on the OS implementation this may result in a system crash, reboot, kernel dump, etc. This kind of attack can also be performed on other protocols.

A *fragmentation attack* is realised by an attack tool named Teardrop [95] that uses the vulnerability of some operating systems of "overlapping" IP fragments. When IP splits data into fragments and sends the fragments in different packets to a destination host, it uses the More Fragments (MF) and the Fragment Offset fields in the IP header to indicate still pending fragments. The destination host uses this information to reassemble the fragments into a complete packet.

There are plenty of other attack tools that use similar mechanisms to bring down destination hosts, with names such as Targa, SYNdrop, Boink, Nestea Bonk, TearDrop2 and NewTear [28].

Such vulnerabilities are easy exploited by an attacker, but also easily eliminated. As soon as the vulnerability is detected, it can be fixed by modifying the source code. Usually, vendors provide *patches* for their software soon after a new exploit has become known. The local system administrator then has to install the patch to prevent further attacks.

Attacking a Resource

The second common DoS attack is to overwhelm a resource at the target. The attack tries to overwhelm resources at the target by generating more requests than the target can handle. There are three common resources an attacker can exploit:

1. Memory
2. CPU power
3. bandwidth

The exploitation is possible because all these three resources are finite.

Given a sufficiently large number of agents, it is possible to simply send any type of packets as fast as possible from each machine and consume all available network bandwidth at the victim. This is a *bandwidth consumption attack*. The victim cannot defend against this attack on its own, since the legitimate packets get dropped on the upstream link, between ISP and the victim network. Thus, frequently the victim requests help from its ISP to filter out offending traffic.

However, in order to realise a "maximum DoS gain" even with low rate attack traffic, most resource depletion attacks try to bind more resources than

just consuming bandwidth in the network. Therefore, those attack packets are crafted in a way to not only consume as much resources as possible while in transmission, but also to consume as much resources as possible directly at the destination entity while it evaluates them.

Such attacks can be made on algorithms, such as hash functions that would normally perform its operations in linear time for each subsequent entry. By injecting values that yield worst-case conditions in the algorithm, such as all values hashing into the same bucket, the attacker can cause the application to perform their functions in exponential time for each subsequent entry, thus causing a *CPU power consumption attack*.

As long as the attacker can freely send data to the server that is processed using the vulnerable hash function, it can overload CPU utilisation of the server. In the worst case it could degrade a request what would normally be a sub-second operation into one that takes several minutes to complete. The victim might be able to immunise the host from this kind of attacks by changing the middleware to remove the vulnerability.

Examples. A common example for a *memory exhaustion* attack is the *TCP SYN attack* [96] with which the attacker tries to exhaust the memory available at a victim host for storage of new incoming TCP-connections. The attacker sends a stream of TCP SYN packets to a victim's listening TCP port. Each request originates from a different source by forging the IP source address field, which is commonly referred to as IP spoofing.

For each incoming packet the destination host must search through existing connections. If the incoming packet does not match any connection, the host has to allocate a new data structure for the connection. Finally it will send a TCP SYN/ACK packet back to what it believes is the originator of the packet. The resulting situation after sending this packet is often called "TCP - half open".

If the forged source address of the TCP SYN packet refers to an actually running system, this system will receive a TCP SYN/ACK packet for which it did not send a SYN packet. Consequently, it will most likely send a TCP RST packet to the victim of the DoS attack, which will delete the corresponding entry of its internal connection database. If, however the forged source address does not reference a running system, the victim host will wait for an answer until a TCP timeout expires, so that the memory allocated for this connection is occupied for a longer period of time.

By using forged source addresses in attacking packets other hosts become secondary victims of the attack. The phenomenon of receiving unsolicited packets as a result of a DoS attack is sometimes also referred to as "backscat-

ter". This effect can be used for a rough estimation of the amount of flooding attacks in the Internet [97].

Additionally, there are several possibilities to misuse the fragmentation ability of IP. For example, it is easy to send a sequence of fragments without the first fragment. While the IP processing entity of the destination system waits for the arrival of all fragments, is not able to re-assemble the fragments and pass them up to the next layer, to free its own buffer memory. This attacking technique can exhaust the memory of systems that perform re-assembly of IP packets.

Most attacks work because of an inherent asymmetry in certain protocols. This asymmetry enables the attacker to create a large amount of work and consume substantial resources at the server, while sparing its own resources. Generally, the only fix that works against protocol attacks is creating a protocol patch that balances the asymmetry in the server's favour. If the fix requires changing the protocol, both the sender and receiver must use the new version of the protocol. Changing commonly used Internet protocols for any reason is normally not possible.

DoS CPU Attacks on the Authentication Mechanism. For client authentication, cryptographic mechanisms are generally used, which is a main security service in many Internet protocols. However, the involvement of computationally expensive operations that are inherent to cryptographic protocols are a common target for CPU exhaustion attacks.

Authentication. Authentication, the proof of the identity of an entity or the origin of a message, is the most fundamental security service as most other security services build upon it. There are basically two variants of authentication: Data origin authentication, i.e. to ensure that data has not been altered, and entity authentication, i.e. to verify the identity of an entity.

As in communication networks direct verification of the above is difficult or insecure, authentication of peer entities is usually established by running a cryptographic protocol. Like any other protocol, a cryptographic protocol must fulfil the following requirements:

- Everyone involved in the protocol must know the protocol and all of the steps to follow in advance,
- Everyone involved in the protocol must agree to follow it,
- The protocol must be unambiguous, that is every step is well defined and there is no chance of misunderstanding, and

- The protocol must be complete, i.e. there is a specified action for every possible situation.

- As an additional property a cryptographic protocol has to ensure that it should not be possible to do or learn more than what is specified in the protocol.

While the last issue has received a lot of attention during the past 20 years in research on design and analysis of cryptographic protocols, the aspect of guaranteeing that every involved entity agrees to follow the protocol has not been questioned extensively. However, recent attacks show the vulnerabilities of cryptographic protocols against DoS attacks: cryptographic protocols require a server to execute computationally expensive cryptographic algorithms.

The *cryptographic algorithms* [98] that are generally used in authentication protocols can be classified according to the following scheme:

- **Encryption algorithms**, which may be further divided into:

  - *Symmetric encryption algorithms*, that uses a single key for encryption and decryption of data. This key has to be kept secret between two entities participating in a secure exchange, explaining the common term secret key encryption for this class of algorithms. Prominent algorithms in this category are the Data Encryption Standard (DES) [99] and the Advanced Encryption Standard (AES) [100].

  - *Asymmetric encryption algorithms*, that uses two different keys for encryption and decryption of data. Each entity possesses a pair of keys: one private key which is only known to the issuer and one public key which is publicly announced. The sender encrypts messages with the public key of the receiver. As the corresponding private key is only known to the receiver, he is the only one able to decrypt the message. Prominent examples for asymmetric encryption algorithms are the Rivest-Shamir-Adleman (RSA) algorithm [101] and the ElGamal algorithm [102].

- **Integrity check values**, that may be further subdivided into:

  - *Modification Detection Codes* (MDC), which are computed using a class of algorithms called Cryptographic Hash Functions. An MDC alone does not protect a message, it represents a digital

fingerprint which needs to be signed using either a secret or a private key in order to ensure that the message originated from a specific sender. Common algorithms in this class are the Message Digest 5 (MD5) [103] and the Secure Hash Algorithm (SHA-1) [104].

– Message Authentication Codes (MAC) which are computed over a message using also public key algorithms. The sender encrypts the message or message digest with his private key. Knowing the sender's public key this allows every other entity to verify, that the message actually did originate by the sender, as only the sender knows the corresponding private key. A prominent example for this is HMAC [105].

Regarding the potential DoS threats arising of the usage of these algorithms, it turns out that asymmetric cryptographic algorithms inhibit the highest DoS risk as they require an additional processing overhead, with a factor of 100 - 1000 in terms of primitive operations in comparison to symmetric encryption algorithms. Cryptographic hash functions can be computed even faster than symmetric encryption algorithms and therefore inhibit the lowest DoS risk.

Examples for DoS Attacks on Authentication Protocols. DoS attacks on authentication protocols make use of the fact that the protocols use up a lot of the server's resources. These resources are in the form of expensive computation power and server memory to store data relevant to the connection, i.e. identities, nonces and similar state data. Figure 2.12 shows an example of a malicious client launching a DoS attack on the authentication function of a server.

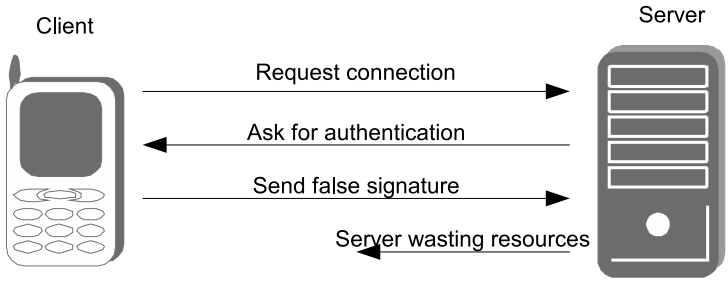

Figure 2.12: Example DoS attack on authentication

Other scenarios to spend server resources are for example an attacker starting an authentication procedure and then leaving the server at an inter-

mediate stage. If the attacker can convince the target server to perform a large amount of computations, e.g. through the verification of many dummy signatures, legitimate client request to access the server are processed with a lower priority. Another possibility is to stimulate the server to generate digital signatures. In case public keys with short exponents are used, the generation of RSA signatures, for example, requires much more computational effort than verification due to the deployment of a relatively larger exponent in the signature generation.

Many protocols including SIP make use of Transport Layer Security (TLS) [81] for securing the exchange of signalling messages between SIP entities. The TLS authentication protocol offers a wide variety of cryptographic options. Figure 2.13 shows the principal message exchange during TLS session establishment.

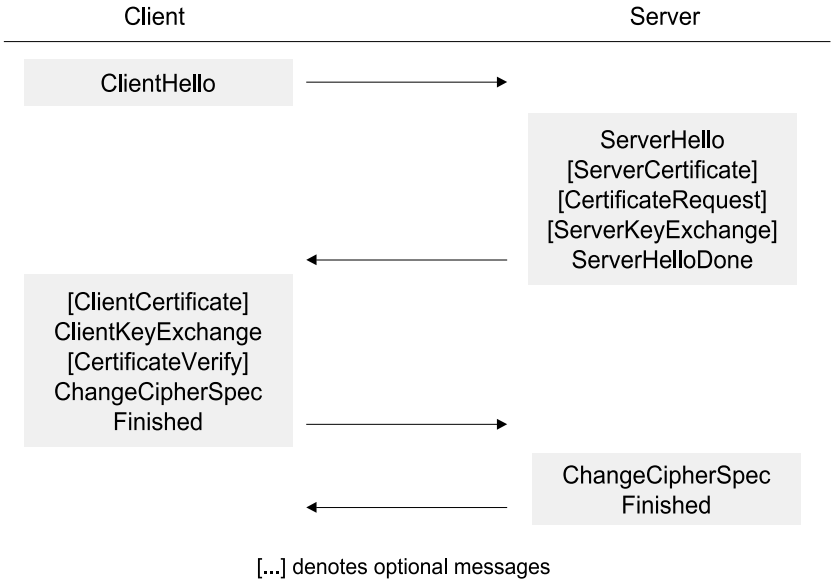

Figure 2.13: Overview of the TLS session establishment

SSL supports three methods for establishing session keys:

1. RSA: a so-called pre-master-secret is randomly generated by the client and sent to the server encrypted with the servers public key.

2. Diffie-Hellman (DH): a standard Diffie-Hellman exchange [106] is performed and the established shared secret is taken as pre-master-secret, and

3. Reuse of previously negotiated security state: if a previously established session allows resuming, the pre-master-secret can be used to generate fresh session keys without requiring expensive asymmetric computations.

A DoS attacker will likely use the first two cases to create load during the decryption process.

2.3.3 Distributed Denial of Service Attack

Over the time DoS attacking strategies have become more elaborate with one of the most severe forms being Distributed Denial of Service (DDoS) attacks. In case of DDoS, attackers make use of the joint power of multiple systems when launching an attack. Therefore, even servers with a high amount of resources and high bandwidth connections are vulnerable to such attacks.

Distributed Denial of Service attacks can be realised using different topologies. The so-called Master-Slave-Victim Topology is illustrated in Figure 2.14 on the left hand side. The master either directly represents the attacker or is controlled by the attacker while slaves represent terminals being controlled by the master system. In this topology each slave terminal attempts to flood the victim with a massive amount of packets using forged source addresses so that the victim is unable to detect the real source of the flood.

The Master-Slave-Reflector-Victim Topology depicted on the right hand side of Figure 2.14 is characterised by the use of amplifying networks to increase the damage. The attacker commands each slave to send spoofed echo request packets to the broadcast address of an amplifying network. The source address is spoofed and equal to the address of the victim. Every host of the amplifying network will reply to the victim. Note that with this topology the source address of actual attacking traffic arriving at the victim is not spoofed, because reflectors send with their real source address, assuming that they received an echo request from the victim.

This construction of DDoS networks has a lot of advantages: the identity of the attacker is concealed and the stability of such attacking networks is greater, because the masters do not know all of the clients, so that the attacking network can handle partial failure caused by detection of some master and slave systems. To increase the power of attacks, slaves can be installed on systems with high bandwidth connections.

DDoS Procedure

A DDoS attack has to be diligently organised by the attacker.

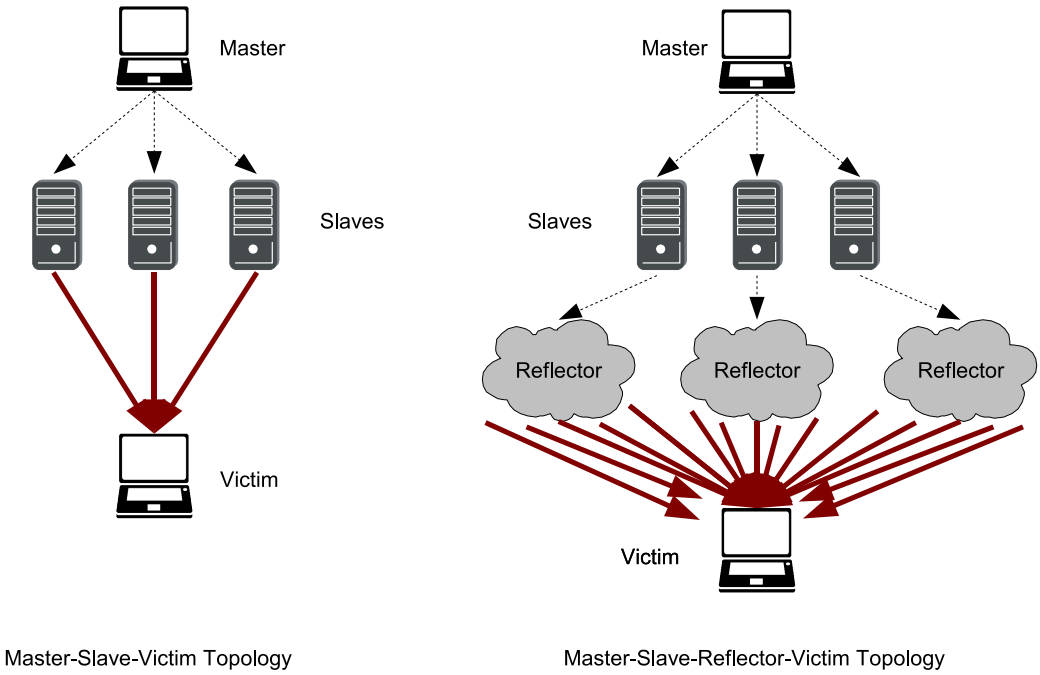

Figure 2.14: Different attacking topologies of DDoS networks

Firstly, it recruits the agent army. For a sussesful attack, multiple hosts have to be put into the agent army. There are special tools for this procedure available, e.g. Trinoo, Tribe, Bagle and more [107].

To recruit an army, the attacker needs to look for vulnerable machines, breaking into them, and installing its attack code using an attack tool. An example program that employs scanning to identify vulnerable hosts is an *Internet worm*. Internet worms are automated programs that propagate from one vulnerable host to another, in a manner similar to biological viruses. A worm has three distinct primary functions:

- **scanning**, to look for vulnerable machines
- **exploitation**, which compromises machines and establishes remote control
- **payload**, code they execute upon compromise to achieve some attack function

The worm will install its control or DoS code on the machine and attempts to duplicate its code on other victim machines. The attacker needs to exploit a vulnerability in the machines that it is intending to recruit in order to gain access to them. The vast majority of vulnerabilities provide an attacker with

administrative access to the system, and it can add/delete/change files or system settings at will.

Exploits typically follow a vulnerability exploitation cycle:

1. A new vulnerability has been discovered in attacker circles and is being exploited in a limited fashion.

2. The vulnerability makes it outside of this circle and gets exploited at a wider scale.

3. Automated tools appear, and non-experts (script kiddies) are running the tools.

4. A patch for the vulnerability appears and gets applied.

5. Exploits for a given vulnerability decline.

One vulnerability that is not mitigated by patching, is weak passwords. Some exploits contain a list of common passwords. They try these passwords in a brute-force or iterative manner, one after another. All too many times, these exploits succeed in finding a weak login/password combination and gain unauthorised access to the system.

The attacker next establishes communication channels between machines, so that they can be controlled and engaged in a coordinated manner. This is done using either a handler/agent architecture or an IRC-based command and control channel. Once the DDoS network is build, it can be used to attack as many times as desired against various targets.

The attacker needs to decide on a propagation model for installing his malware. A simple model is the central repository, or cache, approach: The attacker places the malware in a file repository (e.g. an FTP server) or a web site, and each compromised host downloads the code from this repository. One advantage of the caching model for the defender is that such central repositories can be easily identified and removed. Figure 2.15 illustrates propagation with a central repository.

Another model is back-chaining, or pull-approach, wherein the attacker carries his tools from an initially compromised host to subsequent machines that this host compromises. Figure 2.16 illustrates propagation with back-chaining.

Finally, the autonomous, push, or forward propagation approach combines propagation and exploit into one process. The difference between this approach and back chaining is that the exploit itself contains the malware to be propagated to the new site, rather than performing a copy of that malware after compromising the site. The worm carries a DDoS tool as payload,

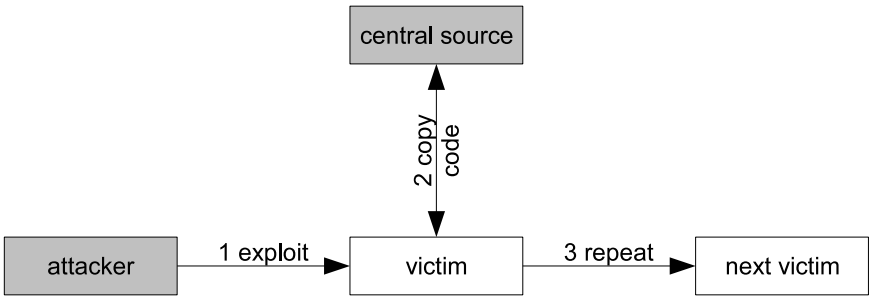

Figure 2.15: Propagation with central repository

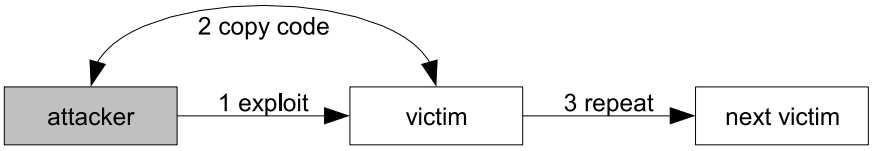

Figure 2.16: Propagation with back chaining

and plants it on each infected machine. Figure 2.17 illustrates autonomous propagation.

When the agent army has been recruited in sufficiently large numbers, the attacker communicates with the agents using special "many-to-many" communication tools. The purpose of this communication is twofold:

1. The attacker commands the beginning/end and specifics of the attack.

2. The attacker gathers statistics on agent behaviour.

An example are handler/agent networks, where the attacker controls the network by issuing commands to the handler, which in turn relays commands to the agents, as shown in Figure 2.18. Commands may consist of unencrypted text, obfuscated or encrypted text, or numeric (binary) byte sequences. Analysis of the command and control traffic between handlers and agents can give insight into the capabilities of the tools without having access to the malware executable or its source code.

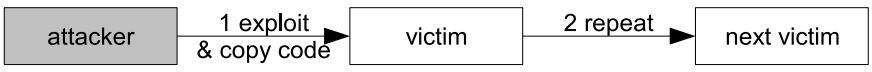

Figure 2.17: Autonomous propagation

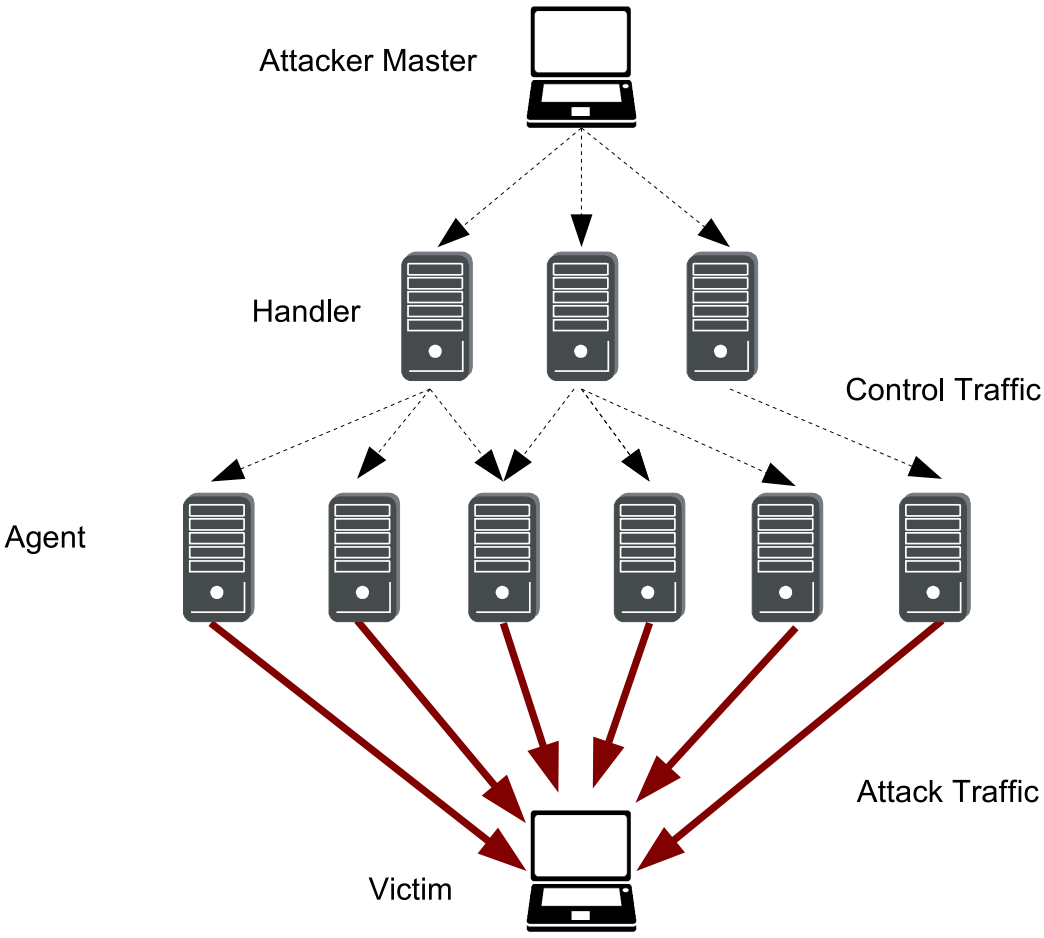

Figure 2.18: DDoS network control architecture

2.3.4 Mitigation Countermeasure Analysis

Both DoS and DDoS are a huge threat to the operation of the Internet sites, but the DDoS problem is more complex and harder to solve. First, it uses a very large number of machines. This yields a powerful weapon as nearly any target, regardless of how well-provisioned it is, can be taken offline. Gathering and engaging a large army of machines has become trivially simple, because of the previously described, widely available DDoS tools. The second characteristic of some DDoS attacks that increases their complexity is the use of seemingly legitimate traffic. Resources are consumed by a large number of legitimate-looking messages; when comparing the attack message with a legitimate one, there are frequently no telltale features to distinguish them.

How can one defend against the difficult problems raised by distributed

denial-of-service attacks? There are two classes of victims of DDoS attacks: the owners of machines that have been compromised to serve as DDoS agents and the final targets of DDoS attacks. Defending against the former attack is the same as defending against any other attempt to compromise your machine.

DDoS Defences Challenges

The main problem that permits effective DDoS handling is the problem of large scale. DDoS is a distributed threat that requires a myriad of overlapping "solutions" for various aspects of the DDoS problem, which must be spread across the Internet because attacking machines may be spread all over the Internet. The following is a list of challenges for DDoS defence [28].

- **Uncertainty of defence placement**. Ideally, a defence solution should be located close to the attacker, to allow fast reaction to the attack. Unfortunately, it is seldom known where the attacker is located. Also, network closely to the attacker might not be under control of the target, so it is mostly not even possible to locate defences there. Hence, most defences are placed close to the target, with the main drawback that there the defence mechanisms also can be overwhelmed by large-scale DDoS traffic. Ideally, the defence should be located a different places.

- **Lack of detailed attack information.** It is widely believed that reporting occurrences of attacks damages the business reputation of the victim network. Therefore, very limited information exists about various attacks, and incidents are reported only to government organisations under obligation to keep them secret.

- **Lack of defence system benchmarks.** Many vendors make bold claims that their solution completely handles the DDoS problem. There is currently no standardised approach for testing DDoS defence systems that would enable their comparison and characterisation.

- **Difficulty of large-scale testing.** DDoS defences need to be tested in a realistic environment. This is currently impossible due to the lack of large-scale test beds, safe ways to perform live distributed experiments across the Internet, or detailed and realistic simulation tools that can support several thousand nodes.

It is an impossible task to devise a solution targeting the listed challenges. The sheer size of the Internet renders any complete solution ineffective. The

goal should therefore be to devise protection methods compromise between completeness and effectiveness.

DDoS Defence Goals

Whether the DDoS defence strategy is preventive, reactive, or a combination of both, there are some basic goals it wants to achieve:

- **Effectiveness.** A good DDoS defence should actually defend. It should provide either effective prevention that really makes attacks impossible or effective reaction ensuring that the DoS effect goes away.
- **Completeness.** A good DDoS defence should handle all possible attacks. If that degree of perfection is impossible, it should at least handle a large number of them.
- **Provide service to all legitimate traffic.** As mentioned earlier, the core goal of DDoS defence is not to stop DDoS attack packets, but to ensure that the legitimate users can continue to perform their normal activities despite the presence of a DDoS attack.
- **Minimum false-positive rates.** Good DDoS defence mechanisms should target only true DDoS attacks. Preventive mechanisms should not have the effect of hurting other forms of network traffic.
- **Low deployment and operational costs.** DDoS defences are meant to allow systems to continue operations during DDoS attacks, which, despite being very harmful, occur relatively rarely. The costs associated with the defence system must be compensated with the benefits provided by it. Other operational costs relate to overheads imposed by the defence system.

Countermeasures

As DDoS are difficult to counter, there does not exist one single solution to prevent against these attacks. Instead multiple different solutions have been developed that target different aspects of the attacks. We give some examples here.

Ingress filtering. A very effective measure against flooding attacks is called ingress filtering. It primarily counters IP spoofing as used in flooding attacks. The usage of ingress filtering for defeating DoS attacks is described by Ferguson and Senie in [108]. The main idea is depicted in Figure 2.19.

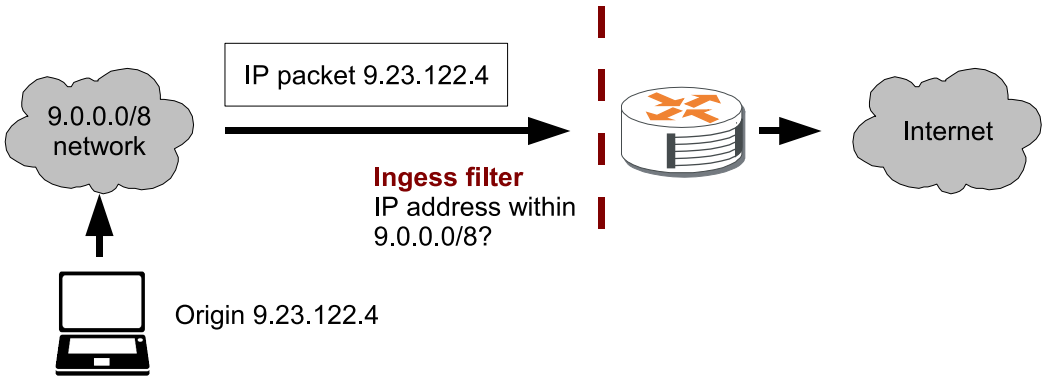

Figure 2.19: Ingress filtering

In the example depicted in Figure 2.19 an attacker with IP address 9.23.122.4 resides within the subnetwork 9.0.0.0/8, which provides Internet connectivity by the ISP via a router. The input link of that router must be observed, so that only packets with source addresses within 9.0.0.0/8 may pass the router. All other packets should be dropped since their source address is not correct (because of spoofing or false configuration). Additionally, information on suspicious packets can be logged. Likewise, ISPs that provide connectivity to single end users (e.g. dial-up connections) would allow only one possible correct source address.

There are also some disadvantages of this method. First, it is a big effort to implement these filtering rules Internet-wide. Then, an attacker may forge the address to an address of another host in the own network. Nevertheless, it is much easier to trace the true source since the possible address range is reduced.

Furthermore, there are some specific services (e.g. Mobile IP) which are affected by ingress filtering, because the source address of packets sent from visiting mobile hosts does not match with the network where the station is attached. Possibilities like reverse tunneling are examined to solve this incompatibility.

Packet filtering. As a basic packet filtering measure, it is already very useful to restrict traffic sent to or from special IP addresses (at least at border routers between networks, e.g. a router connected to an ISP):

- incoming or outgoing packets with broadcast addresses like 0.0.0.0 and 255.255.255.255
- incoming packets with addresses from the (internal) local network itself

- incoming or outgoing reserved addresses (private address space):
  - 10.0.0.0 - 10.255.255.255 (10/8, reserved)
  - 127.0.0.0 - 127.255.255.255 (127/8, loopback)
  - 172.16.0.0 - 172.31.255.255 (172.16/12, reserved)
  - 192.168.0.0 - 192.168.255.255 (192.168/16, reserved)

Stateless Operation. The idea of using stateless protocols at the beginning of an authentication protocol and its benefits were recognised by Yung et al. in [109]. Aura and Nikander generalised the approach to create stateless servers that maintain connections by passing the state data to the clients [110].

The overall goal of this approach is to relief the server from storing state information before an anonymous DoS attack can be ruled out. Aura and Nikander therefore suggested that the first few steps of an authentication protocol should be stateless until client authenticity has been successfully checked or the client has in some other way clearly shown its commitment to honest use of the service.

As illustrated in Figure 2.20 the basic idea of stateless protocols is not to store state at a server that represents a target for potential DoS attacks, but to send the session state back to the client. In order to ensure the confidentiality and integrity of the state data, however, appropriate security measures (encryption and integrity protection) should be provided to it. As only the server needs to be able to read and check the state information, symmetric cryptography can be used for this purpose allowing for a relatively efficient implementation.

Stateful Operation		Stateless Operation
1. C → S: Msg_1		1. C → S: Msg_1
2. S → C: Msg_2	S stores $State_{S1}$	2. S → C: Msg_2, $State_{S1}$
3. C → S: Msg_3		3. C → S: Msg_3, $State_{S1}$
4. S → C: Msg_4	S stores $State_{S2}$	4. S → C: Msg_4, $State_{S2}$
...		...

Figure 2.20: Stateful vs. stateless protocol operation

While stateful protocols are vulnerable to memory exhaustion attacks, making them stateless results in protocols being more resistant against at-

tacks and allows them to recover faster after an attack. Stateless protocols are in particular more resistant to attacks that leave connections in a half-open state as they pass the task of storing state data to the client and the network. However, one drawback of this approach is that it increases storage and bandwidth requirements on the client side and in the network.

Cookies. A *cookie* is a data structure which is generated by a server and given to a client at the start of a protocol run, requesting that the client stores and returns this data structure with subsequent protocol messages.

The Photuris key management protocol by Karn and Simpson [111] was probably the first protocol to implement a cookie exchange so that the server could check if the initiator of a key management dialogue is actually on the route between the server and the source address given in the initial request. The Photuris specification lists the following requirements for cookies:

- the cookie must depend on the addresses of the communicating parties,
- nobody else must be able to forge a cookie that will be accepted by the server,
- the cookie generation and verification must be fast enough so that they will not represent a target for DoS attacks, and
- the server must not keep per-client state until the clients IP address has been verified (i.e. it has received a cookie generated by itself)

The recommended cookie generation method is based on a cryptographic hash function (such as MD5). It is also recommended that the cookie be calculated over a secret value only known to the server, the IP source and destination addresses, and the transport layer source and destination ports.

A protocol that makes use of the cookie idea usually proceeds as follows: when a client wants to connect to a server, the server will send a cookie containing information that is unique to that particular request and a hash value computed over this information and over a secret known only to the server. The client must then return the cookie in the next message. Only if the cookie is verified and accepted by the server, subsequent protocol processing will continue.

Summarising, the cookie measure enables countering simple address spoofing attacks as in this case the attacker will most likely not be able to receive the cookie and is thus not able to force the server to continue with the protocol execution. However, if the attacker is able to eavesdrop the servers first reply containing the cookie, e.g. because it is located close to the server and

can eavesdrop on the servers communication lines, the cookie mechanism can not provide any protection.

Furthermore, the cookie idea can be realised together with the idea of stateless protocol operation without requiring additional computations. In this case the connection state can be sent back to a requestor which already contains information specific to a particular request and is as well integrity protected.

Client Puzzles. The idea of using client puzzles to slow down the sending rate of a potential attacker was first presented by Dwork and Noar [112] who proposed to ask the sender of an email to solve a so-called client puzzle for each message, thereby increasing the cost of junk mailing. Juels and Brainard [113] presented a simpler puzzle that may be given to a client during TCP's connection setup phase in order to counter TCP-SYN attacks. Aura et al. [114] further developed this idea to cover attacks on cryptographic protocols. The following discussion is mainly based on the last reference.

The idea behind client puzzles is that a client puzzle can be easily generated and verified by a server, while clients must use a significant amount of computational resources in order to solve it. Furthermore, the puzzles' difficulty can be easily scaled based on factors such as server load or server trust of the client.

The idea can be illustrated with a simple example: a server generates two random numbers NS and X' and computes a cryptographic hash value $h = H(NS, X')$ of them. The server then provides the client with one of the random numbers NS and k bits (for example 8 bit) of the hash value. The client must then guess random numbers and compute cryptographic hash values until k bit of a resulting hash value match the value that has been supplied by the server. As cryptographic hash functions can not be inverted, the client on the average has to try $2k - 1$ different random numbers until he finds one number X so that 8 bit of $H(NS, X)$ match the value provided by the server. However, in order to generate and check the client puzzle, the server just needs to compute the cryptographic hash function two times. This effort on the server side can be further reduced by just generating and sending one random number NS and the parameter k to the client and always requiring the first k bit of $H(NS, X)$ to be of a fixed value, e.g. 0.

The basic properties of a client puzzle as required by Aura et al. are as follows:

- the creation and verification of a puzzle is inexpensive to a server,
- the server can adjust the cost of solving a puzzle (from zero to impossible),

- the puzzle can be solved on most types of client hardware,
- the pre-computation of solutions is impossible,
- the server does not need to store any client-specific data while a client solves the puzzle (The server does not know X at the time it creates the puzzle as in order to know it, it would have to solve the puzzle itself, additionally the client also generates an own nonce, so the server can not know this particular puzzle at this moment. In order to check a client puzzle, the server needs to be able to recognise its own random numbers to ensure freshness of the puzzle. Note, that NS is not client-specific and therefore does not imply a growing storage need when DoS attacks occur. This protects against memory depletion as the server does not store client-specific data before (!) the client has successfully solved a puzzle.),
- the same puzzle may be given to several clients, while ensuring that knowing the solution of one or more clients does not help a new client in solving the puzzle, and
- a client can reuse a puzzle by creating several instances of it, however, the solution to a puzzle should not be reusable (this may in fact require storage of solved puzzle instances for a given period of time).

Client puzzles provide an effective means to significantly slow down potential DoS attackers while at the same time only minimally increasing the length of messages (about one byte for parameter k and up to eight bytes for the solution X). The overall concept is to make the client commit its resources before the server commits its resources. It protects servers at the early stage of a normal authentication where the computations are the most CPU intensive.

2.4 Intrusion Detection Systems

2.4.1 General Overview of Intrusion Detection

Intrusion detection systems (IDS) are software- or hardware systems intended for the automation of monitoring and analysis of events occurring in computer systems or networks with the aim of indicating potential security problems. IDSs can be generic in nature to be customised with detection rules specific to the environment in which they are deployed (e.g. Snort [115] and Prelude

[116]), while others are specifically targeted towards particular environments or specific classes of intrusion incidences.

In the last two decades, IDS have been an active research area in the computer security field due to the fact that the usage of computing tools and networking all over the world have grown rapidly, and in parallel the security violations caused by the users of computing tools and networking facilities have also increased.

There are three fundamental functionalities of an IDS: monitoring, detecting and responding. In order to realise those three functionalities, an IDS is composed of:

- **Sensors**: generate security events.
- **Console**: monitors events and controls sensors.
- **Engine**: records events logged by sensors in a database, uses a system of rules to generate alerts from security events received and informs the responsible authority.

Today, intrusion detection systems are not only able to alert an administrator if something irregular occurred in the network but can even prevent from intrusion attacks. They also may provide the capability of stopping malicious attackers by dropping respective packets or by applying other appropriate countermeasures (e.g. unregistering of malicious users).

Intrusion detection systems do not guarantee a complete security; on the contrary, when it is used in conjunction with other security tools such as firewalls, access control, security policies, data encryption and user authentication, the security of the system to be protected is enhanced.

2.4.2 Intrusion Detection System Types

There exists three general concepts of IDS, as shown in Figure 2.21, and outlined in the following paragraphs.

Network-based IDS

Network Intrusion Detection Systems (NIDS) are placed at strategic points within the network in order to monitor traffic to and from all devices of the network. Ideally, a NIDS scans all inbound and all outbound traffic. The network-based IDS can scan network packets at the router- or host-level, verifying packet information, and logging suspicious packets by adding supplementary information.

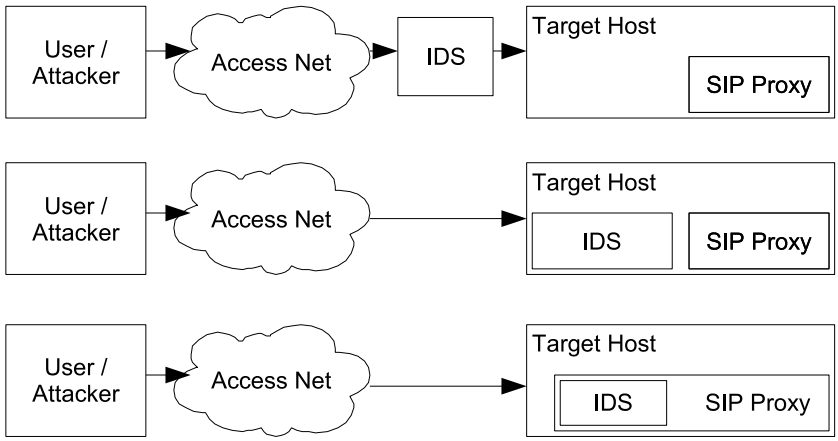

Figure 2.21: Difference between IDS systems. Top: Network IDS (NIDS). Middle: Host IDS (HIDS). Down: Extension Module

Based on this collection of suspicious packets, a network-based IDS can scan its own database of known network attack signatures and assign a severity level to each packet. If the severity level exceeds certain thresholds, an alarm is raised.

Although network-based systems are able to simultaneously monitor numerous hosts, they may suffer from performance problems, especially in case of increasing network speeds. Many network-based systems simplify assumptions about network properties, e.g. about packet fragmentation and can suffer from resource exhaustion problems when they have to maintain attack-state information for many attacked hosts over a long period of time.

Despite of these deficiencies, they are popular because they are easy to deploy and manage as standalone components, and they have in the ideal case only little or no impact on the protected system's performance.

Generally, NIDS fail if the traffic is encrypted. This is because the NIDS requires access to the data part of the packets, not just the headers in order to detect intrusion attempts. If in the future encrypted traffic is used throughout all networks, this is expected to become a very serious problem for network-based systems. NIDS also cannot be used to defend switched networks unless the IDS is incorporated into the switch itself. This is true because the NIDS needs to put the network interface into promiscuous mode to capture all packets in the network, not just the ones addressed to the system on which the NIDS is running. Switched networks cannot be used in this way.

Snort [115] is a prominent open source NIDS.

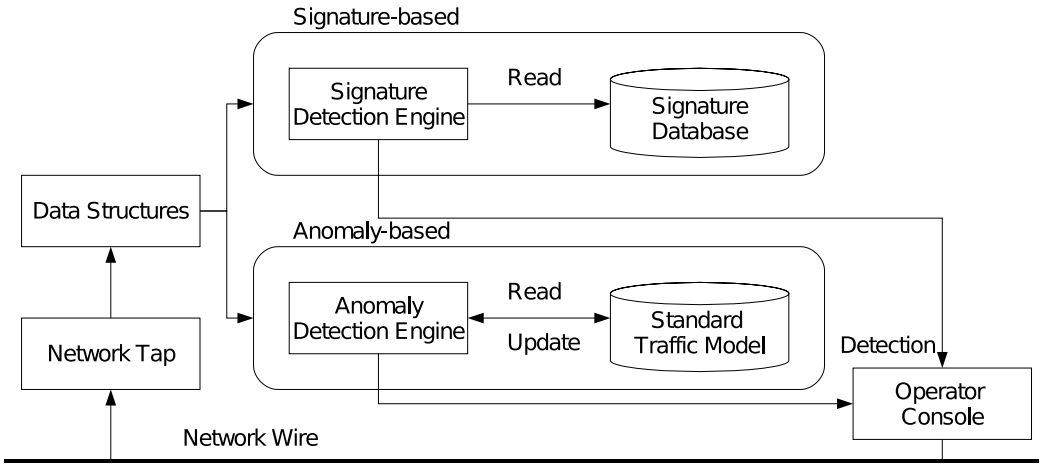

Figure 2.22: NIDS architecture

Host-based IDS

A Host Intrusion Detection System (HIDS) analyses several areas in order to determine misuse (malicious or abusive activity inside the network) or intrusion (breaches from the outside). HIDS consult several types of log files (kernel, system, server, network, firewall, and more), and compare the logs against an internal database of common signatures for known attacks. The HIDS filters log entries, analyses them, re-tags anomalous messages using its own severity rating scale, and collects them in its own specialised log for further administrator-based analysis.

A HIDS can also verify the data integrity of important files and executables. It checks a database of sensitive files (and any files that one may want to add) and creates a checksum of each file. If any of the file checksums are altered in an irregular way, an alert is raised.

Examples for HIDS are Tripwire [117] or the Simpler WATCHer [118].

Hybrid IDS

Hybrid Intrusion Detection Systems combine capabilities of network based IDS and host based IDS. Thus these systems are on the one side capable of monitoring network trafficlike network based IDS and on the other side to monitor different application and system logs like host-based IDS at the same time. Prelude is an example of a hybrid IDS [116].

2.4.3 Detection Strategies

Commonly, detection can be classified in *misuse based detection* and *anomaly based detection* [119].

Signature- or Misuse-based Detection

In order to be able to detect attacks a signature-based IDS requires specific attack descriptions. These can be as simple as a specific pattern matching a portion of a network packet or as complex as a state machine or neural network description that maps multiple sensor outputs onto abstract attack representations. If an appropriate abstraction can be found, signature-based systems can identify previously unknown attacks which are abstractly equivalent to known patterns. Signature based IDS are inherently unable to detect truly novel attacks and suffer from false alarms when signatures match both intrusive and non-intrusive sensor outputs. Signatures can be developed in a variety of ways, ranging from hand translation of attack manifestations to automatic training or learning using labelled sensor data. Because a given signature is associated with a known attack abstraction, often names (such as Smurf or Ping-of-Death) are assigned to attacks.

Anomaly-based Detection

An anomaly based IDS monitors network traffic and compares it against an established baseline. The baseline identifies what is "normal" for that network, i.e. what sort of bandwidth is generally used, which protocols are used, which ports and devices generally connect to each other and alert the administrator or user when supposedly anomalous traffic is detected, or significantly different in comparison with the baseline. Baseline characterisation approaches range from statistical models of component or system behaviour, neural networks and other AI techniques, to approaches inspired by the human immune system. The primary strength of anomaly detection is its ability to recognise novel attacks. Its drawbacks include the necessity of training the system on noise with the difficulties of tracking natural changes in the noise distribution. Changes can cause false alarms, while intrusive activities which appear to be normal activity can lead to detection failures. Anomaly-based systems have difficulties classifying or naming attacks.

Anomaly based detection was first proposed by Dorothy E. Denning. In her work ([120]), she presented the idea of creating profiles out of the audit data for every subject on a system, where a subject may be a user, a process or a host. After profiling the behaviors, the subsequent behaviors of those

subjects are compared with the profiles using statistical metrics in order to find the anomalous ones.

2.5 Relevant Tools

We describe in this section the most relevant tools we use in our work.

2.5.1 Netfilter / IPtables

Netfilter and iptables [121] are building blocks of a framework inside the Linux kernel. This framework enables packet filtering, Network Address Translation (NAT) and other packet mangling. Netfilter is a set of hooks inside the Linux kernel that allows kernel modules to register callback functions with the network stack. A registered callback function is then called back for every packet that traverses the respective hook within the network stack. Iptables is a generic table structure for the definition of rule sets. Each rule within an IP table consists of a number of classifiers (iptables matches) and one connected action (iptables target).

2.5.2 SER

The SIP Express Router (SER) [61] is an open-source SIP server, published under the GNU Public License. SER's architecture has been designed for high scalability to serve thousands of calls per seconds, and flexibility due to a modular plugin-approach and a highly configurable routing language. SER is already in widespread use at many VoIP providers.

SER was designed in a highly modular manner as depicted in Figure 2.23, it consists of an efficient core that is responsible for receiving, parsing SIP and forwarding messages.

The core is also responsible for invoking certain procedures that are provided as extension modules. These modules are dedicated to providing certain features. Such modules include:

- Registration and user location: This part is responsible for handling registration requests and managing the user location database.

- Transaction management: When acting in a stateful mode SER must maintain per transaction state, generate replies, match replies to requests and deal with forking.

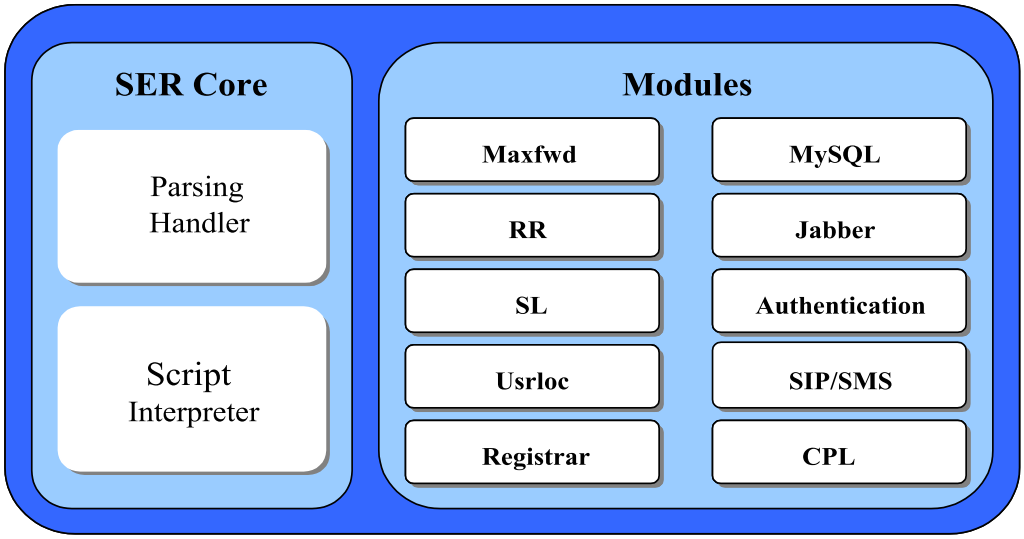

Figure 2.23: General architecture of SER

- User authentication: SIP utilising HTTP digest for authenticating user requests. This part of SER interacts with the database, which maintains the users identity and password and is responsible for checking the credentials of the users.

- SIP handling: While the core deals with message processing, the transaction management deals with state handling and taking the appropriate SIP actions, this type of modules deal with additional processing logic such as handling of record-route headers, loop detection or support for ENUM.

Each module exports a set of functional procedures and utilises procedures exported by other modules. The integration between the different modules and the core is realised through a configuration language. The SER Configuration Language (SCL) is a rich and flexible scripting language. In this context SCL is similar to shell languages with support for regular expressions and allowing the administrator to specify certain rules whenever a specific event occurs, in the manner of

```
If (event) then (action);
```

In the figure below, the administrator specifies that after receiving a registration request, the core should invoke the authentication module before handing the registration by the registration module

```
If(method==REGISTER)
    www_challenge(sip.org/*realm*/, 0);
break;
```

To enable the extension of the functionality of SER it provides an open programming interface to utilise the current and novel functionalities of SER and add novel ones.

This is achieved by implementing a new module that provides these new features. Such a new module would use the exported procedures of the other modules as well as functionalities of the core. The new module in its turn exports new procedures that could be invoked by the core and other modules as well.

The SCL is then used for integrating the new functionalities in the general functionality of SER. Using SCL, the administrator of the SER platform can define the exact behaviour of the platform based on available modules and needed actions.

2.5.3 PROTOS Suite

PROTOS [122] describes a method to assess the robustness of SIP implementation by describing a general test suite to find vulnerabilities. The test suite sets a baseline to determine vulnerabilities in SIP products focusing on SIP parser abilities, and can be used during in-house testing or as part of regression testing. PROTOS has proven to be efficient in testing for software vulnerabilities by using a black-box testing method based on syntax testing. In syntax testing, test cases are generated from the corresponding input specification. The test suite is written in Java and is freely available.

2.5.4 SIPp

SIPp [123] is an open source SIP traffic simulation and testing tool. Multiple scenarios can be defined in XML scenario description files, both for UAS and UAC. SIPp can handle advanced scenarios with multiple calls, IPv6, TLS, SIP authentication, injecting of variable data and media playback. Some simple scenarios are already hard-coded into the application.

SIPp can be used to test SIP equipments like SIP proxies, B2BUAs, IMS CSCF and user terminals.

Chapter 3

SIP Security Threats with Focus on DoS

With SIP's multiple application possibilities combined with the complexity of the protocol as introduced in Chapter 2, several security weaknesses have already been discovered. Listing and analysing a full SIP threat model would already justify another separate document. Hence, in this chapter we start with a general SIP threat introduction and introduce three main SIP security threats - message fraud, spam and Denial of Service. We elaborate partly on the first two topics and give pointers to further relevant work for the interested reader. In the remainder of this chapter we introduce the SIP DoS security threat in great detail. We introduce the three main SIP DoS forms message payload tampering, message flow tampering and general message flooding.

3.1 General Threats and Related Works

Security and *privacy* requirements[1] in a SIP communication environment are expected to be equivalent to the currently predominant communication platform, the PSTN. Contrary to the PSTN, SIP networks are commonly deployed over the open Internet, which complicates the provision of security and privacy in SIP networks.

SIP messages may contain information that a user or provider wishes to keep private. For example, message headers may reveal confidential information about the communicating parties. The SIP message body may also contain user information (addresses, telephone number, etc.) that must not be exposed to third parties. The open and distributed nature of the SIP

[1]Results of this chapter have been published in [19, 20, 64].

Table 3.1: Network and application security issues

Issues	Impact
Eavesdropping: unauthorised interception and decoding of signalling messages	Loss of privacy and confidentiality, message fraud
Viruses and software bugs	DoS / unauthorised access
Replay: retransmission of genuine messages for reprocessing	DoS, SPIT
Spoofing: impersonation of a legitimate user	Unauthorised access, fraud
Message tampering / integrity	loss of integrity, DoS, SPIT
Prevention of access to network services e.g. by flooding SIP proxy servers / registrars	DoS
SIP-enabled IP phones: Trivial File Transfer Protocol (TFTP) eavesdropping, Dynamic Host Configuration Protocol (DHCP) spoofing, Telnet access	Loss of confidentiality, Unauthorised access, DoS

networks, especially SIP VoIP architectures, in conjunction with the variety of subsystems (like DNS, AAA, Application Servers) further complicates the task to secure the network.

The potential SIP threats can be divided into external and internal ones. External ones are attacks launched by someone who does not participate in a SIP-based communication and usually occur when signalling and data packets traverse untrustworthy network realms. Because of this, SIP suffers from all known attacks associated with any Internet application or subsystem. Internal threats originate from within the SIP network. Table 3.1 illustrates some of the identified threats and attacks, and their impact on the overall SIP security.

The most severe threat in a VoIP environment is probably the easy access to the communication channel. For instance, the existence of several internet tools for analysing VoIP traffic makes eavesdropping a simple task for any potential perpetrator. Furthermore, the text-based nature of SIP messages gives more opportunities for attacks like spoofing, hijacking and message tampering as opposed to closed network communication. The results are three classes of SIP attacks, which are

1. SIP message fraud
2. Unsolicited messages, i.e. spam
3. Denial of Service

The content of a SIP message can be altered during its way through the Internet. This is mostly done in a harmful way to gain an advantage over the party whose messages are tampered with, which is generally called **message fraud** [49]. This could be done to gain free access to monetary systems, e.g. hacking into the PSTN gateway. Another possibility is to steal confidential information by eavesdropping on the communication channel. Research in this direction is still in its infancy. Tools have been developed to probe for vulnerabilities in SIP devices that can be exploited for message fraud attacks [124]. Wang et al show that leading US VoIP services are vulnerable to multiple fraud attacks [125]. Also, possibilities to launch billing attacks are evaluated [50]. It is expected that message fraud attacks will be increasing in the following years. The upcoming European research project SCAMSTOP [126] will be addressing SIP message fraud.

In general, **spam** refers to any unsolicited communication and in telephony, traditional or IP-based, it usually refers to unsolicited bulk call attempts. In that case the denomination Spam over IP Telephony (SPIT) is common.

SPIT is considered also a problem for SIP [127]. Multiple handling strategies have been defined. For example, Croft et al. [43] present an approach to prevent voice spam by extending the call setup procedure. They include an anonymous mediator into the call path that manages the session setup. Mathieu et al. [44] present an approach to detect and mitigate spam through a network-level entity. The basic function of the specific entity is to capture, filter and analyse the network traffic, passed from the equipment located in the edges of the network, in order to detect and mitigate spit calls. Quittek et al. [128] propose a method to apply hidden Turing tests to detect spam calls. The IETF proposes a strong authentication system to make life harder for potential SPIT-senders. [129, 15]. A novel approach is the detection on Mailbox spam using audio analysis [12]. These and other approaches have also been addressed in the European research project SPIDER [65].

The SIP **DoS Threat** is outlined in more detail in the following sections.

3.2 SIP DoS Introduction

The goal of a Denial-of-Service attack is to *render the service or system inoperable*. Hence an attack can be directed toward different entities in the

network, depending on the attacker's intent. If the aim is to render the service as a whole inoperable, the main target will be the core servers in the SIP infrastructure. These can be all SIP proxies, but also other servers on which the SIP infrastructure depends: DNS, RTP proxies, gateways to other networks, etc. Direct attacks on the user agent are also possible, however they will have a lesser impact, i.e. the attack's effect will only be noticed by the user agent itself.

We will distinguish between three different types of SIP DoS attacks. They are *SIP Message Payload Tampering*, *SIP Message Flow Tampering* and *SIP Message Flooding*. A classification, to be illustrated in the following text, is depicted in Figure 3.1. The most relevant class and main topic of this document (message flooding) is highlighted.

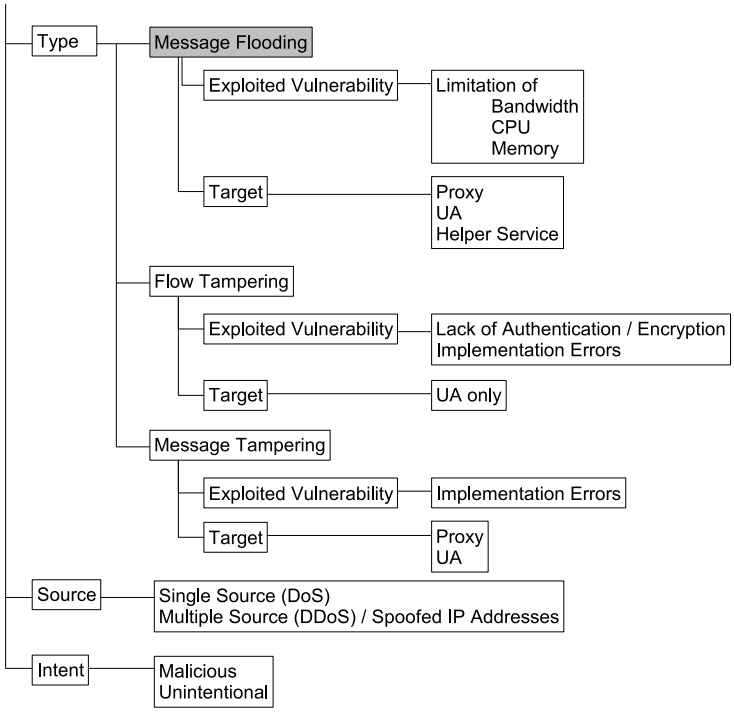

Figure 3.1: Classification of SIP DoS attacks.

3.3 SIP DoS by Message Payload Tampering

The first class of attacks is based on tampering with the actual SIP message or more specifically, the SIP payload. SIP is a text-based protocol and messages are transported usually with clear text. Attackers can try to inject harmful

content into a message, e.g. by entering meaningless or wrong information with the goal of creating a buffer overflow at the target. Also, such messages can be used to probe for vulnerabilities in the target. Harmful code that will be executed in an unforeseen context can be introduced into the payload. An example is SQL code injection [130], which allows the attacker to execute SQL code in a database.

Such attacks can target both proxies and UAs. Unintentional attacks are possible due to poor SIP implementations. Especially with probing requests it is likely that these messages will have been launched from different sources.

3.3.1 Example: SIP Message Tampering with SQL

SQL Injection is a well known attack in the internet world. The goal of this attack is to inject harmful SQL code in web page data entry forms, that will be executed by the authentication database [131].

Similarly, SIP servers rely on SQL databases (e.g. MySQL [132]) and utilise SQL statements in order to store and administer users credentials and appropriate data for providing services. For example, the SER proxy provides build-in modules in order to support MySQL and other databases. Here, *Subscriber* and *Location* tables provide relevant service-related information. In detail, the Subscriber table stores the user-related data such as user name, domain, password etc. (see Table 3.2) for SIP authorised users, while the Location table stores all the data representing the current available contact addresses for all legitimate subscribers.

SQL injection in SIP can be triggered every time a SIP network entity (e.g. SIP UA, SIP Proxy) is asking for authentication. So, in case a SIP network element requests authentication, the UA on behalf of the authorised user computes the appropriate credentials based on the HTTP Digest mechanism [80]. The result of this computation (credentials) is included in the messages authorisation header. Then the message is forwarded to the proxy server, which has to authenticate the received message. Thus, it recalculates user credentials using the user's password stored in the Subscriber table as presented in Table 3.2. To accomplish this task, it generates an SQL statement of the following syntax:

```
SELECT password FROM subscriber
WHERE username='joe' AND realm='black.net'
```

In case a malicious user tries to launch an SQL injection attack in the SIP architecture, he spoofs the SIP message and inserts malicious SQL code in its Authorization header, like the following:

Table 3.2: Example subscriber table entry

Username	Domain	Password	First	LastName	URI
joe	test.net	12345	joe	user	joe.user@test.net

```
Authorization:Digest username="joe';
  UPDATE subscriber SET first='malicious'
  where username='joe'--",
  realm="black.net", algorithm="md5",
  uri="sip:black.net",
  nonce="41352a56632c7b3d382b5f98b9fa03b",
  response="a6466dce70e7b098d127880584cd57
```

This message can be any SIP message requiring authentication by a SIP server. The code can be embodied either in the username or in realm fields in the Authorization header. As soon as the proxy receives such a tampered SIP message, it generates and executes the following SQL statement:

```
SELECT password FROM subscriber WHERE username= 'joe';
UPDATE subscribe SET first'malicious' WHERE username='joe'--
```

As a result, message authentication fails, but the second command manages to change joe's 'first' to 'malicious'. It is also possible for a malicious user to attempt to employ other SQL commands, with the aim of making the database useless and cause a DoS to the provided service.

3.4 SIP DoS by Message Flow Tampering

A special case of DoS attacks in real time communication networks are attacks that disturb the ongoing communication between users. Common internet services like web browsing or email communication have an asynchronous time model, e.g. a requested web page is delivered directly to a user. The user will read it without further communication with the web server. The same applies to email – a user downloads the email and studies it independently of a server connection. In contrast, in SIP real time communication networks two communicating users establish a constant connection with each other whereby content is transmitted continuously between both parties.

An attacker can now target this connection by introducing fake or modified signalling messages into the communication channel. Several different SIP signalling messages can be misused for this task.

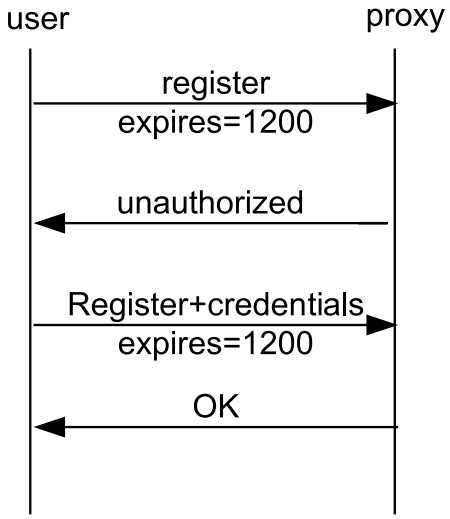

Figure 3.2: Normal register flow

The attacker needs to know the session parameters in order for these attacks to function correctly. The parameters can be sniffed from the network. Tests have shown that multiple implementations do not follow the SIP specification correctly, thus proving the feasibility of such attacks [133].

Flow tampered attacks are mostly targeted at SIP UAs.

3.4.1 REGISTER Attack

REGISTER requests add, remove, and query bindings. A REGISTER request can add a new binding between an address-of-record and one or more contact addresses. Figure 3.2 depicts a common registration procedure: A legitimate user builds a REGISTER message and sends it to its proxy server. The SIP server generally requires authentication for incoming REGISTER messages, thus it sends a response to the user with an unauthorised message including a random number that must be used by the UA to create the corresponding credentials. The legitimate user re-constructs the previous REGISTER message by including the appropriate credentials and re-sends it again. After this, the user is registered with the service.

The registration procedure can also be accomplished by a suitably authorised third party, which has the appropriate privileges to perform registration on behalf of a particular address-of-record. A client can also remove previous bindings or query which bindings are currently in place for an address-of-record by setting the `expires` header-field tag to zero.

An attacker may now de-register all existing contacts for a URI and then register his own device as the appropriate contact, thereby directing all requests for the affected user towards the attacker's device. An attacker may for example register his own device as the contact address of the victim and deregister all old contacts, thereby preventing these users from being invited to new sessions.

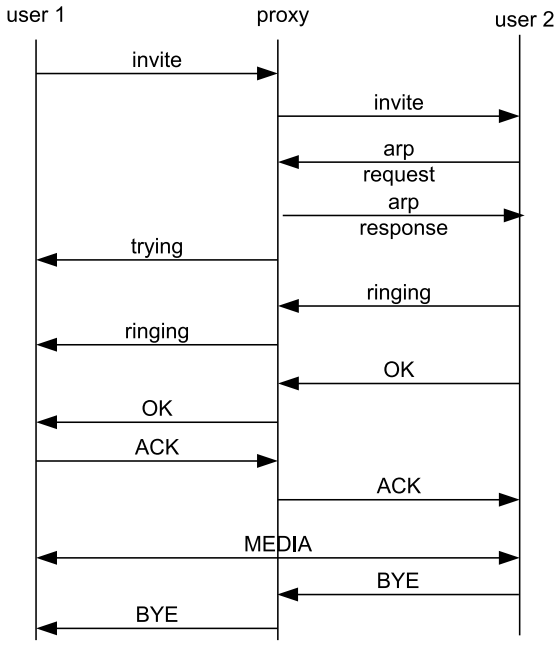

Figure 3.3: Normal session termination

Moreover, it is possible to launch a DoS attack by the modification of a register request that contains more than one Contact address which utilises "q values" to classify the corresponding priority of these addresses. The "q value" indicates a relative preference for the particular Contact field. If these values are set to zero, it is possible that the user can not receive an incoming SIP INVITE. Equivalent to the previous attack is the malicious modification of the Expires header whenever a malicious user sets the value of this header to zero. In addition, a malicious user may modify not only the "q values" but add his contact address and changes the corresponding "q values" to receive all incoming calls.

3.4.2 INVITE Attack

Once a dialog-session has been established by initial messaging, subsequent requests can be sent that attempt to modify the state of the dialog-session. Thus, any unauthorised modification with a forged re-INVITE of a dialog-session by a potential attacker may cause a DoS.

3.4.3 BYE Attack

Another attack that can tear down a session is the BYE attack. The BYE request is used to terminate an established session as depicted in Figure 3.3.

An attacker in possession of the correct session parameters is now able to tear down another session, as depicted in Figure 3.4. Depending on the implementation, the BYE acknowledgement responses might not even be routed to the original session participants. In a phone setup, this could be very annoying for the participants. The end of the conversation would thus only be discernible because no voice answers are routed back.

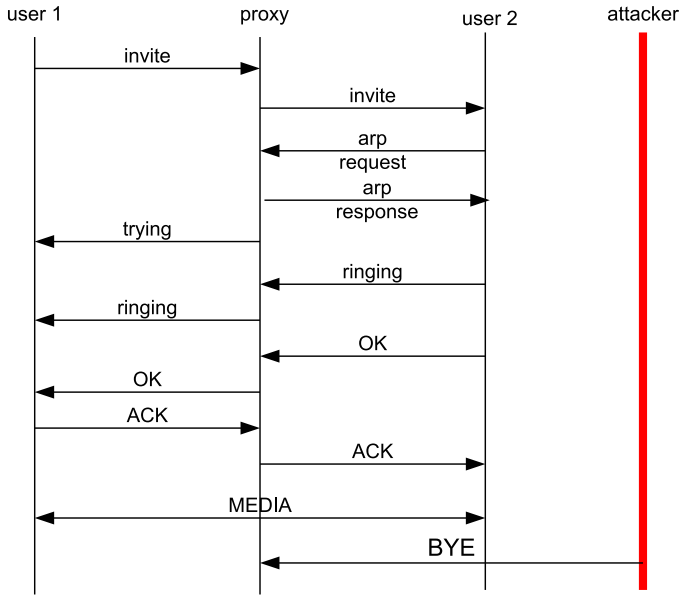

Figure 3.4: Spoofed session termination

3.4.4 CANCEL Attack

A CANCEL request is sent to prematurely cancel an ongoing INVITE request. More specifically, it asks the UAS to cease processing the request

and to generate an error response designating that request, as depicted in Figure 3.5.

The attacker may utilise the CANCEL method to cancel a legitimate third party INVITE request as shown in Figure 3.6.

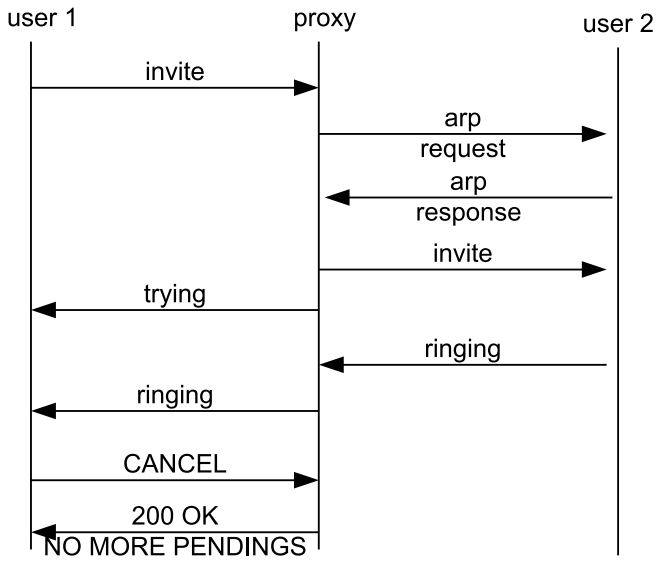

Figure 3.5: Normal CANCEL flow

However, CANCEL requests can only be used in conjunction with INVITE requests. Thus, when a SIP proxy receives a CANCEL request for any other message type (than INVITE) it must not process this message but rather produce an appropriate error response. Also incoming CANCEL requests must not be processed if the original request has already generated a final response. This limits the time window to place a CANCEL attack.

However, contrary to most other SIP messages, CANCEL messages are transmitted hop-by-hop and can not be re-submitted. As a result, they can not be challenged by the server in order to get proper credentials in an Authorization header field. Because of this, CANCEL attacks might still be a suitable method for attackers.

3.4.5 UPDATE Attacks

The SIP UPDATE method gives to end users various capabilities such as muting incoming calls, identification of QoS service and negotiation of other session attributes. Forged UPDATE messages can thus be utilised e.g. to degrade QoS, so that a user cannot communicate properly with other parties.

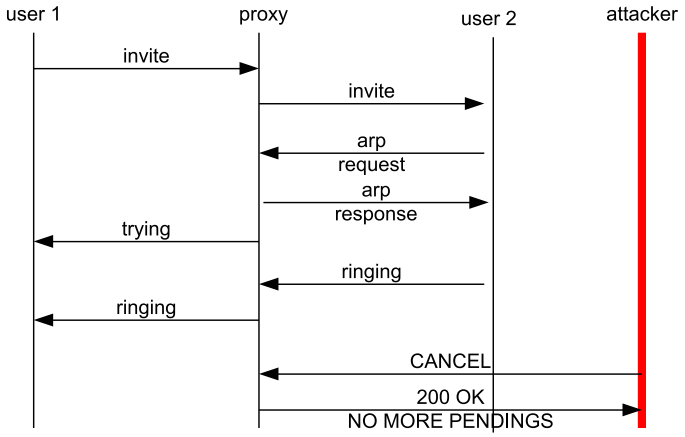

Figure 3.6: CANCEL attack

3.4.6 REFER Requests

The REFER method [134] is specified in an extension that provides a mechanism where one party (the referrer) provides a second party (the referee) with an arbitrary URI to reference. Assuming that this URI is a SIP URI, the referee will send a SIP request (usually a SIP INVITE) to that URI (the refer target). As a result, REFER can be used to enable new applications, including call transfer. RFC 3892 [135] extends this method, allowing the referrer to provide information about the REFER request to the refer target using the referee as an intermediary. The refer target can use this information to decide whether to accept the referenced request from the referee. This scheme enables the referee to act as an eavesdropper, giving him the ability to launch man-in-the-middle attacks. For example, the referee can forge the Referred-By header or/and eavesdrop on the referred-by information. The referee may also copy all the related information into future unrelated requests. Although the specification uses an S/MIME based mechanism to enable the refer target to detect possible manipulation of the Referred-By header data, this protection is completely optional.

3.5 SIP DoS by Message Flooding

The most common incarnation of a DoS attack is where an adversary sends a huge amount of messages to a target with the goal to overwhelm the target's processing capabilities, and hence rendering the target inoperable. Such attacks are generally hard to detect, as the utilised attack messages are usually valid messages and thus not easily distinguishable from regular messages (see

Figure 3.7).

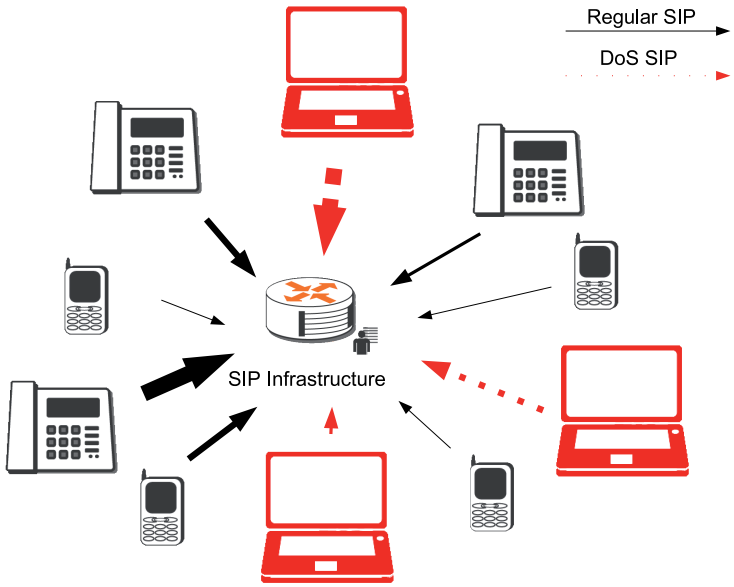

Figure 3.7: Distinction problem in a DDoS attack scenario

In the first place, the main SIP proxy (especially the *registrar*) would be the predominant target for an attack. However, the whole SIP ecosystem can be severely jeopardised if another component of the network (e.g. STUN server, RTP proxy or DNS server) is being attacked.

A DoS attack can focus on the depletion of three different resources, which are *memory*, *CPU usage* and *network bandwidth*. Many depletion attacks can already be defeated by intelligent implementation (e.g. parallel processing, avoiding blocking conditions) combined with powerful hardware for the proxy and the coherent usage of the SIP authentication mechanism. However, sophisticated attacks can still breach such a prepared system. Here begins the hard work of defence, as such intelligent attacks generally cannot be detected by the proxy itself, but only by an intelligent network monitoring solution. This detection mechanism must operate at network line speed to handle the enormous traffic generally involved in a DoS attacks. Also it needs to have clear metrics to distinguish between legitimate traffic and potential dangerous (DoS) traffic. If the monitor fails here, false alarms will be raised or in the worst case legitimate users will be falsely denied access to the service.

A Denial-of-Service is not necessarily triggered by a malicious attack. Unintentional attacks can additionally occur by misconfigured end devices or wrong user implementations.

3.5.1 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 into 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 AAA, 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 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 is sent out, the server can delete the buffered data.

Stateful servers: In general we can distinguish between two types of state in SIP:

1. 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 users interaction is involved.

2. 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[22].

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 which can cause blocking. Such operations may be misused to quickly paralyse the server's operation.

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.

- Security check: SIP might use secure protocols providing state-of-the-art security (TLS, S/MIME, IPSec). However, very few implementations actually do that. The most widely deployed security protocol is still digest authentication. Possibly, one of the reasons is the cost-factor: Vendors attempt to keep prices of end-devices low and build devices with limited memory capacity (typically less than 1 Megabyte) leaving themselves few space for cryptographical protocols. Aspects of DoS attacks on SSL and IPsec are independent of SIP itself.

- Supporting services: In the context if this document a supporting service is a service that is used by a SIP server in order to fulfil 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 a CPL server or some other form of an application server. This interaction is realised in general over some form of inter-process communication and involves in general the generation of appropriate requests, exchanging those requests over

the network and analysing the reply. 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.

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 servers 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 work.

3.5.2 Overview of SIP Flooding 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 is broken digest authentication; invalid clients calculate wrong digest values, they are challenged to submit a correct value and continue re-submitting 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.

Besides attacks on SIP itself, as will be described in this section of this document, 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 signalling 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 [136] 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, which can be misused by malicious parties to change the discovered values and corrupt SIP signalling indirectly.

Manipulating DNS entries is a similar 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.

While in general attacks can be mounted against 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.

DoS Attacks Based on Exhaustion of Memory

State maintenance in SIP servers is one of the 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.

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. Different SIP messages can be used for message flooding, e.g. REGISTER, INVITE, or OPTIONS. An example using REGISTER messages is shown in Figure 3.8.

Brute force attacks from inside the SIP infrastructure can be launched with messages that contain unrecognised content like e.g. requesting exotic extension or message bodies of unknown type. A SIP entity has to process these requests and generate error messages, but there exist other, more resource-consuming attacks.

Brute force attack with multiple varying messages can be even more harmful for the target if the attacker uses a special variation procedure: The attacker send mostly the same message with only minimal in the Call-ID header. Here, he can put multiple different SIP URIs that all map to the same device, e.g. by adding the port number to one request and leaving it at other requests, or by using the direct address of the domain or the IP decimal address, ... In such cases receiving SIP elements would still handle each request independently, increasing the processing overhead. Such attacks can be launched especially with REGISTER, INVITE and OPTION messages.

DoS attackers might modify the Route header field values found in requests that identify a target host and then send the forged request to a large number of SIP network elements. Record-Route could also be used to produce similar situations when the attacker is certain that the SIP dialog initiated by the requester will result in numerous transactions originating from the backwards direction.

Relaying attacks can be launched by attackers when they create bogus requests that contain a fake source IP address and a corresponding Via header field that identifies a targeted host as the originator of the request, and then send this request to a large number of SIP network elements (i.e. SIP phones).

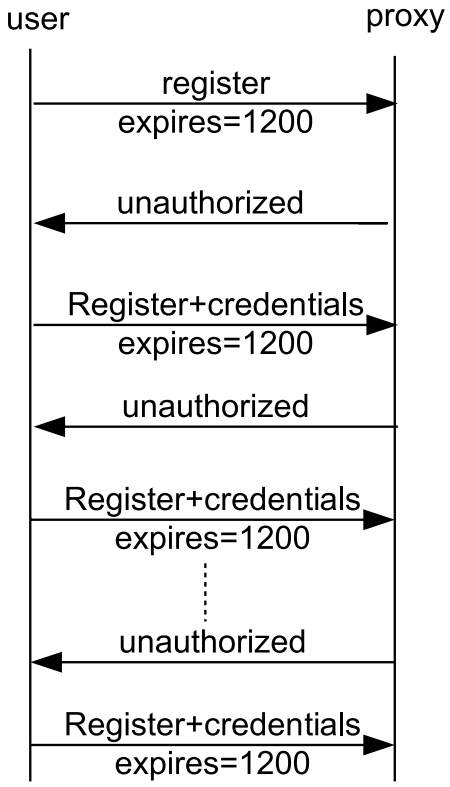

Figure 3.8: SIP message flooding using REGISTER messages

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 need to maintain the state for at least three minutes, where it still tries to retransmit the message.

A SIP element is most vulnerable if it has to keep state for a longer time, which is the case in the INVITE process. After forwarding an INVITE, 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 re-transmits 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

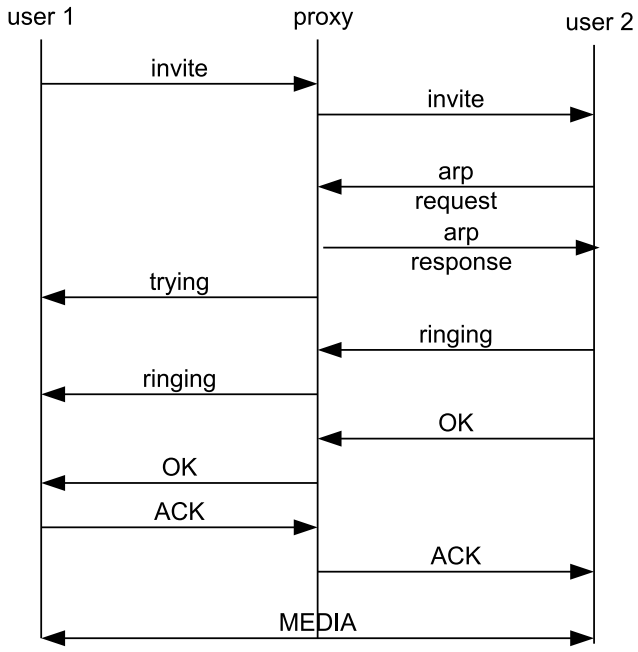

Figure 3.9: Normal flow of INVITE message

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 itself. In this case the server will need to maintain the state until the redirect transaction is replied as well. The UAS has to re-transmit a 2xx response periodically as it cannot guarantee an all-reliable connection. Also, during the INVITE transaction, UA capabilities are exchanged. Generally, the UAC sends its capabilities first using an SDP payload included in the INVITE message. However, the client can also wait for the UAS to offer its capabilities, delegating this work task to the other side. This can be utilised by an attacker.

General UA behaviour for keeping state is as follows (the timer names are those taken from RFC 3261):

- UAC clients have to re-transmit unreliable INVITEs if they do not receive an answer within a maximum of 32s (Timer C). They also need to keep state for another 32s after they have forwarded a 3xx-6xx (Timer D).

- UAC will continuously re-send non-Invite messages for up to 32s (Timer

F), even after receiving a 1xx response. After receiving a 2xx response, they still need to keep state for another 5s (Timer K).

- UAS has to re-sent any 1xx messages for INVITEs if a re-request arrives. Any 3xx to 6xx responses need to be re-send up to 32s (Timer H).

- UAS after receiving an ACK need to keep state for another T4 sec before abandoning.

- UAS only re-sends responses for non-INVITES after a query for up to 32s (Timer J).

As an example, consider end user devices that have been designed mainly to respond under normal conditions. This means that they are able to process a few incoming messages simultaneously. Figure 3.9 presents a normal flow of INVITE message.

Considering the situation that an attacker impersonates himself as a valid user 1 he will possibly generate numerous INVITES as depicted in Figure 3.10. Under this situation the attacker builds up INVITEs only, without any respond message.

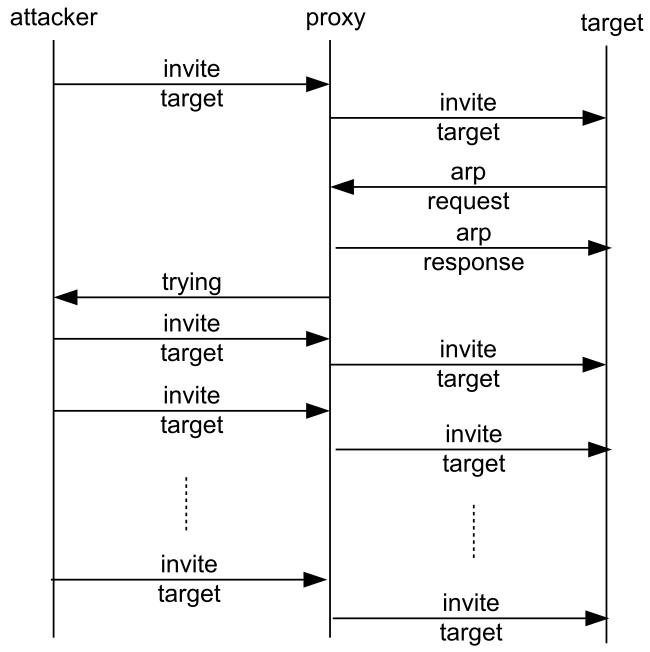

Figure 3.10: Flood with INVITE messages

Another scenario that an attacker could possibly exploit is depicted in Figure 3.11. In this scenario the attacker tries to "behave" like a legitimate UA. It is likely that the attacker will deploy different INVITE scenarios to cause a DoS in the proxy or in the end-terminal device.

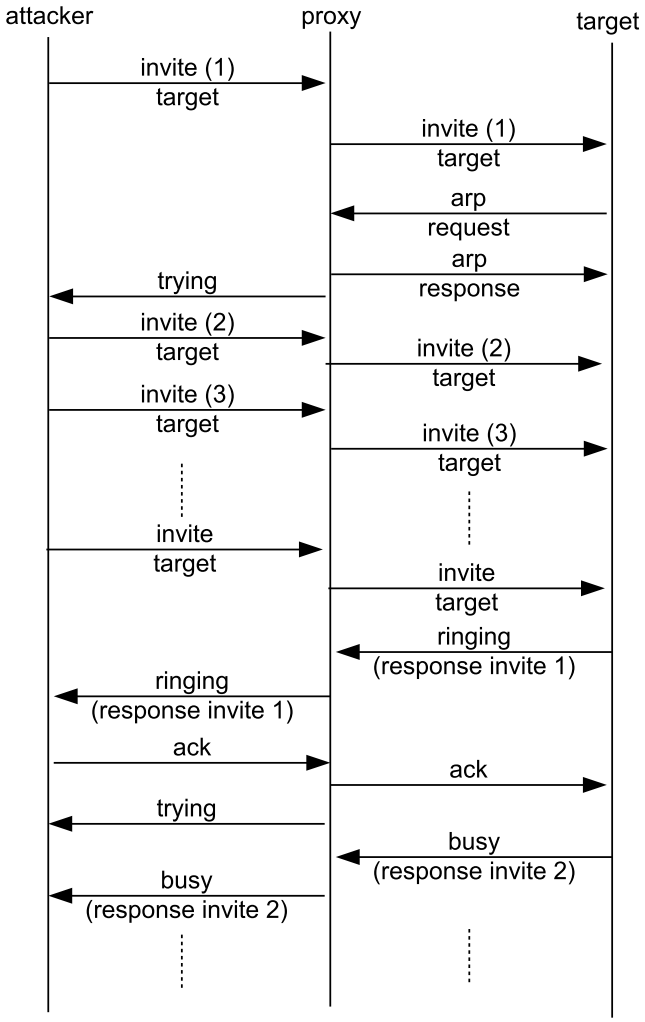

Figure 3.11: Alternative flood with INVITE messages

CPU Attacks

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 small 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 messages also increase network utilisation and memory usage. For an attacker to be effective with this method, he has to utilise 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 MTU, or if it is larger than 1300 bytes and 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 to the SIP specification.

- Poor parser implementations can be rendered inoperable by including message bodies of a size that does not match the one 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 that each only contains one value. If multiple message headers of the same field are included in a message where theses headers are spread all over the message, this will further complicate the parsing. Figure 3.12 illustrated 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.

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

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

Figure 3.12: Multiple header placement possibilities

- Some message headers are more vital for processing than other ones. Vital header fields are all routing-specific fields, like To, Via, Route, etc. So, messages with these fields placed towards the end of the message require more processing power to parse. One way to accomplish this is by inserting multiple informative header fields, e.g. Allow or Supported, in front of the routing fields.
- SIP as defined in RFC 3261, is a refined version of the previous standard as defined in RFC 2543 [70]. 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 "z9hGbK" the message is considered to be pre-RFC 3261. This would indicate a fallback to the more complex RFC 2543 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.

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 relatively 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 remote communication between the SIP server and the application server is needed. Thereby attacks on the application server result in blocking of 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.

Interaction with External Servers As already indicated a SIP proxy might need to contact an external server to fetch some information or realise 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, e.g. through constantly querying the user's credentials at an AAA server, or incurring increased load by issuing requests that require the execution certain complex application logic at an application server.

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 that server's 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 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 (for example, an invitation to a destination served by a temporarily failed DNS server) may temporarily disable servers operation for several seconds.

By adding addresses that might be irresolvable or unresponsive, a SIP proxy can be forced to forward a message to an unresponsive address by adding the unresponsive address to one of the following headers as depicted in Table 3.3.

3.5.3 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 loops by using a header named Max-Forwards, which is decremented for each traversed proxy. Similar to IP's Time-To-Live, the packet is dropped after reaching the value of 0. Further 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

Table 3.3: Forwarding to non-existent TCP receivers

Client puts an irresolvable address in	Header description	Server action
Request-URI in a SIP request	Indicates the destination the request is being sent to.	Server attempts to forward requests to unresponsive address indicated in R-URI.
Route header-field in a SIP request	Determines the route a request should take. Only proxies must understand this header. After receiving a request with such a header, the proxy is supposed to forward the message to the address indicated in this header.	Server attempts to forward requests to unresponsive address indicated in Route.
Contact header-field of REGISTER messages	Servers might collect the information included in the contact headers of request and reply messages. Proxies might then use this information as entries for a location information cache that could be used to speed up future searches.	Server attempts to forward requests for registered user to previously registered address.
Topmost Via header-field of a request	The VIA headers indicates the path taken by a request so far.	Stateful servers attempt to send a 100 reply to the address. Both stateful and stateless servers relay replies from downstream nodes to this address.
Second topmost Via header-field of a reply		Server attempts to relay the reply to address in second topmost Via.

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 stateful. Thereby, 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 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.

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. A 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 its 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 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 victims. 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 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 addresses 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.

3.5.4 SPIT and Denial of Service (DoS) Attacks

In this section we briefly discuss the difference between SPIT (as introduced in Section 3.1) and DoS, as they have several properties in common.

Both SPIT and DoS attacks have a negative impact on SIP networks, however they express completely different goals and objectives. SPIT aims

at passing some sort of information (related to money, medicine, jobs, etc.) to the recipients without their consent. On the other side, DoS attacks are generated with the intent of disturbing legitimate traffic, potentially rendering any given network-based service useless. As SPIT is also sent (or generated) in bulk, it might be possible that this huge volume of SPIT might reach a level where it becomes also a kind of DoS attack.

Although SPIT has not yet been dispersed in professional environments with financial and marketing motives, it is likely that in the future variations of DoS attacks are encountered, caused by SPITters. One likely candidate could be the early media phenomena: Sending high volume of pre-recorded messages to the destinations will result in high bandwidth consumption, choking network resources, and then expeditiously causing loss of service.

Early media attacks can be either targeted directly at the recipients or at proxies, utilising wrong server configurations or implementation faults.

Another possibility we might encounter could be SIP transactions being manually crafted into templates, and then used for automated outbound calls. If we make the assumption that not all these transactions are correctly implemented according to the SIP standard and applicable extensions, CPU cycles in end-points or proxies might experience a considerable increase, again causing loss of service.

Chapter 4

DoS Protection Requirement Analysis

In the previous chapter we have shown the vulnerabilities of SIP against DoS attacks and other threats. A SIP infrastructure needs dedicated protection features to reduce the risk of falling prey to any of these attacks. In this chapter we list protection requirements[1] for the three main classes of DoS attacks as established in Chapter 3.

We assume that protection is first and foremost required for server components. This analysis is to a limited degree also applicable to UAs.

4.1 Requirements for Payload Tampering Protection

SIP is a text-based protocol, thus messages are human readable. A sophisticated parser is necessary to translate the human-readable message payload into a machine-readable representation. As experience has shown, flaws in such an implementation like buffer overflows or missing integrity checking can result into serious security breaches. It is therefore highly important to protect against payload attacks.

SIP is now a mature standard and *the* technique to prevent payload attacks is a well-tested and robust implementation. To help developers, there are different tools to check SIP implementations for correct operations [122, 124], thus any well-established SIP agent should generally be hardened against payload attacks. However, many different parser implementations exist and with SIP's popularity new implementations are constantly becom-

[1] Results of this chapter have been published in [19, 4, 17].

ing available. As it is difficult to check each implementation for correct operation, a viable option for the network operator would be to add another payload attack prevention system in the form of a well-specified and tested message integrity checker within an external monitoring tool or as a security extension for the server software, like the examples proposed in [29, 137] (see Chapter 8). Such a setup is also necessary if network operators are aware of implementation flaws in their devices, but there is no software or firmware update available to fix these flaws. The overhead of an additional message check is generally low as no state information needs to be maintained.

4.2 Requirements for Message Flow Tampering Protection

Several message types can be used to disrupt individual SIP sessions, these attacks have been introduced in Section 3.4 and are known as Re-INVITE, CANCEL or BYE attacks, depending on the utilised message type. Many researchers have addressed this problem with dedicated protection methods, like [53, 54, 138] (see Chapter 8). However, effective protection can also be achieved through SIP's dedicated security mechanisms, i.e. by message encryption:

Flow tampering attacks are only possible if an attacker can sniff necessary network parameters. If the signalling flow is encrypted it is nearly impossible to launch this type of attack. SIP already defines mature and established encryption methods, like Transport Layer Security (TLS) [81] or IPSec [82]. Instead of countering the effects of an attack, encryption would actually prevent the attack itself. Note that while encryption is an advisable option against flow tampering attacks, it does not help against payload attacks or flooding attacks.

Regarding the security of the defined protocols, *IPSec* assumes pre-established trust among the communicating parties and it can only be utilised in a hop-by-hop fashion. Since IPSec is implemented at the operating system level, most SIP clients do not implement this protocol yet, however support is increasing. For this reason, IPSec can only protect the traffic between the corresponding network servers. Moreover, the SIP specification does not suggest any framework for key administration, which is required by IPSec. In contrast to IPSec, *TLS* does not assume any trust relation among communicating parties. TLS can be utilised either for one-way or mutual authentication schemes and maybe it is more suitable for inter domain authentication. Of course, there is always the risk that the message can be

intercepted inside the recipients network if the last hop is not encrypted. Additionally, TLS is used by the SIPS scheme to offer an end-to-end security. However, TLS fails to deliver end-to-end security and protects only connection-oriented protocols.

4.3 Requirements for Message Flooding Protection

From the three DoS attack types, protection against flooding attacks is the most difficult task.

4.3.1 Possible Countermeasures Against Memory Exploitation Attacks

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 behaviour, 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 messages would prevent malicious behaviour 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, 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 could have been the usage of predictive nonces [139] that 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. 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. However, it was never officially standardised for SIP.

Stateless proxy: There is a certain computational expense associated with each SIP transaction at each proxy server and this cost is much greater for stateful proxy operation. Thus, an obvious protective measure for re-

ducing the risk of memory exhaustion attacks is to perform as much of the server's functionality in stateless mode before going stateful. The "stateless barrier" should be used to perform as many security checks as possible – these may include

- Stateless authentication of users, forgoing the normal re-transmission algorithm. (By re-transmitting the 401 (unauthorised) or 407 (proxy authentication required) status response it amplifies the problem of an attacker using a falsified header field value, like Via, to direct traffic to a third party.
- Checks of unauthorised 3-rd party registrations,
- Detection of replay attacks,
- 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. For this purpose, the concept of a stateless UAS can be utilised. In this mode, a server generates a reply to a request, sends it out and immediately forgets the request, its computational result and resulting reply. Request retransmissions are processed as brand-new requests. After all the stateless checks, transactional state may be established. Some functions such as static forwarding (e.g. least-cost gateway routing) may be easily executed statelessly. The following list iterates well-known functionality which inherently requires 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.
- Re-transmission buffer, particularly important if a SIP path is known to include a wireless hop to absorb too rush re-transmissions.
- Some services such as "forward_on_event" forward to voice mail 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.

4.3.2 Countermeasures Against CPU Attacks

Server design: The first line of defence against any DoS attack is achieved by using well-dimensioned hardware with fast CPUs and large memory and high speed network connections. 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:

1. **Clean and efficient implementation**: Implementers need especially to use efficient and fast memory allocation schemes, event handling and parsing mechanisms.

2. **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 (see Figure 4.1). 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.

4.4 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.

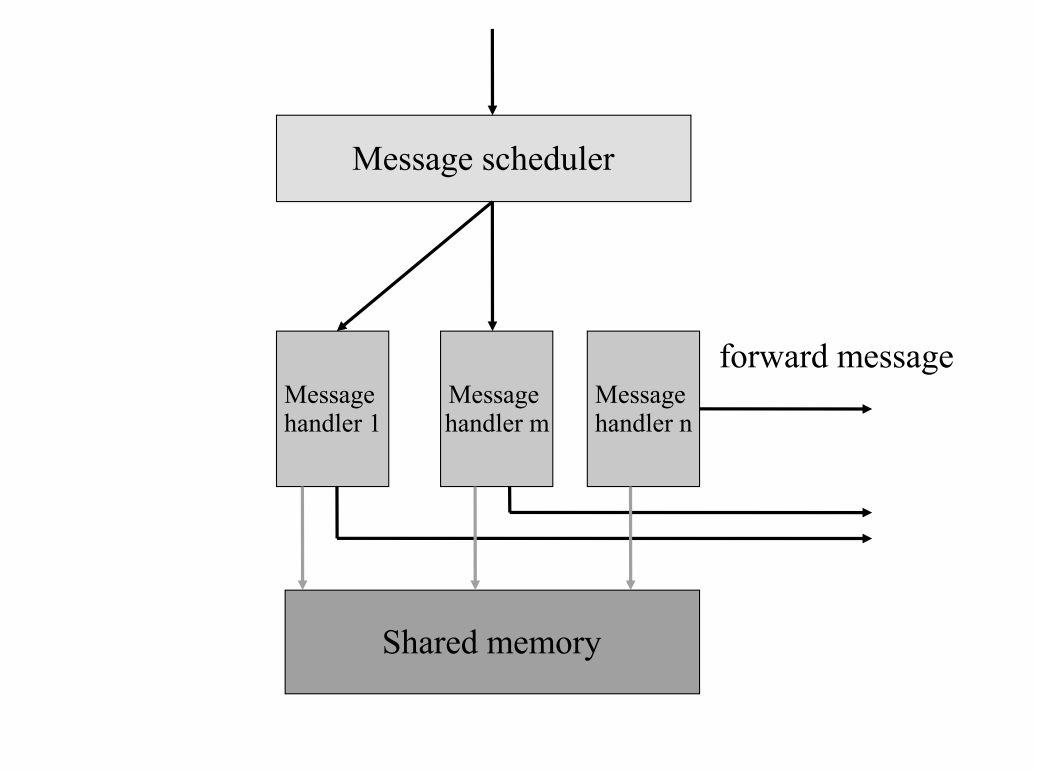

Figure 4.1: Design scheme of a parallel SIP server.

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 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 including parser attacks

   (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, 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. Consider using a technology like 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 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).

4.5 Requirements for External Monitoring

An integral part for DoS protection is thus a well-tested and effective implementation plus additional monitoring. Monitoring can be deployed directly at the server itself or at a dedicated security device. Integrated monitoring at the server has the drawback that the additional monitoring checks might well lead to a self-induced DoS attack by itself: In case of only a light flooding attack, the server's resources are quickly exhausted due to the additional overhead of message monitoring [56].

Thus, we have here defined strict requirements for an external monitoring point to achieve the goal of fast, reliable and effective protection of the service.

1. **Transparency:** The monitoring point must be completely invisible from the networking point of view. It must not be possible to gain knowledge about the existence of it by analysing the network traffic. If the monitoring point is visible by some means, it can be identified and possibly circumvented or even attacked.

    - *No IP routing.* If the monitoring point acts as a router it modifies IP packets, its existence can be guessed from the outside.
    - *No SIP proxying.* A very easy way to intercept SIP traffic is to act as a proxy and use the Record-Route feature from SIP to stay in

the message path. This, of course, announces the monitoring point to every SIP user, regardless whether it is doing proxying or acts as Back-to-Back (B2B) User Agent. Even here, the monitoring point can be circumvented by addressing the protected proxy directly, e.g. by its IP address. Common SIP SBC protection solutions are generally implemented as a B2BUA. From a security perspective, this is not a wise move.

2. **Line speed:** All traffic must be dealt with in real time. This means that no buffering is allowed. Incoming packets must be handled immediately after they are encountered. Otherwise it would be possible to detect the monitoring point by analysis of network delays.

3. **Scalability:** Flooding attacks cause a high bandwidth utilisation. In order to analyse this stream, the architecture must be capable to handle this stream. Therefore it must be possible to deploy new capacities into the monitoring point easily. DDoS attacks can easily reach into the gigabit per second area of bandwidth.

4. **Independence:** To be as usable as possible, the monitoring point cannot rely on a special SIP proxy implementation. Hence, no direct communication channel is established between the monitoring point and the SIP servers. As a consequence, the monitoring point needs to implement a subset of the SIP logic to successfully follow the operation within the network (e.g. session state changes). As such, the monitoring point needs to monitor not only incoming traffic from the outside world, but also responses from SIP servers.

5. **Extensibility:** As it is possible that new types of attacks are discovered, the architecture must be open to allow new algorithms for detection to be integrated rapidly. New functionality should be added or dropped without changing the monitoring point's architecture.

Chapter 5

Security Solution Specification

In Chapter 3 we have introduced the DoS threat on SIP architectures, with the most severe threat being DoS flooding attacks. In Chapter 4 we have introduced the basic requirements for protection. While to some degree the threat can be minimised by deploying a robust and hardened implementation (efficient parser, parallel processing, consequent authentication, ...), this would not be able to cover the full scope of of the DoS threat. Eventually, there is no other way than intelligent, external monitoring of the SIP traffic flow. In the end, this is how we want to target the DoS threat with our **solution approach**: we develop individual *monitoring algorithms*[1], with each algorithm concentrating on a narrow scope of the DoS problem, and thus developing a strategy how this problem can be mitigated by monitoring network traffic flows. All algorithms analyse the network traffic to detect a certain DoS pattern. Whenever a DoS is detected, an alarm is raised and, if possible, malicious traffic is blocked. The algorithms are deployed in security test beds: Two algorithms require high performance capabilities, and are deployed in a scalable security architecture called VoIP Defender. The third algorithm, which is not dependent on high throughput performance and deals with SIP DNS requests is deployed as an enhancement to an available DNS cache.

5.1 General Protection Framework

5.1.1 Overview

In Section 4.5 we have defined general requirements for DoS protection in SIP networks. For our work, there was no solution available that met all defined

[1]Results of this chapter have been published in [17, 7, 13, 3, 16].

requirements. We therefore introduce a general security framework called *VoIP Defender* that complies to those requirements and which is the basis of our protection work. The VoIP Defender architecture has the following characteristics:

Multi-layered architecture For scalability reasons, the task of traffic monitoring has been split into individual and independent components. At the lowest layer, the filter- and scanner node ("Filter") intercepts all raw traffic and outputs re-assembled SIP messages. The actual analysis of the traffic for malicious requests is done in the analysis layer ("Analyzer"). It includes a SIP parser and performs local operations. Multiple Analyzers can operate in parallel, with the Filter forwarding only a subset of the traffic data to each Analyzer. To combine the input from multiple Analyzers, the decision node ("Decider") takes the input from all local Analyzers and decides on a global action, e.g. a user notification or a change in the firewall configuration. The firewall is part of the Filter.

Delayed reaction Also, for scalability reasons, the Filter does not take immediate actions whenever a packet is encountered. Running through the full stack of all nodes, i.e. Filter, Analyzer, Decider and firewall update would result in increased processing delays. Instead, whenever a packet is encountered, it is duplicated. One instance is forwarded to the Analyzer, while the other instance is passed on in the network.

Infrastructure independence VoIP Defender works independent of the protected SIP infrastructure, i.e. it contains all intelligence by its own. It is placed in front of the entity to be protected and does gather all necessary information solely by sniffing from the network traffic.

A basic overview of the VoIP Defender setup is visible in Figure 5.1.

5.1.2 Filter- and Scanner Node ("Filter")

The Filter is probably the most crucial component of the whole architecture for delivering real time behaviour. Its primary purpose is to fork incoming and outgoing traffic towards the Analyzers (scanning). Its second task is to apply filtering rules to all incoming traffic. For outgoing traffic from the protected SIP proxy forking is also applied, with one copy as input for the Analyzers, and another copy send out to the internet, where it is routed normally. Filtering is not applied to outgoing traffic.

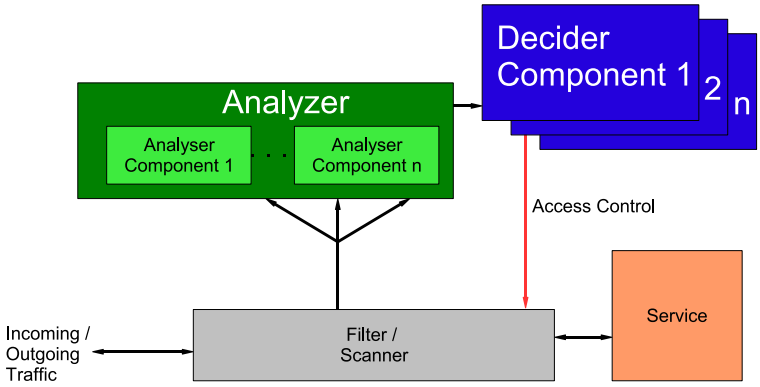

Figure 5.1: VoIP Defender overview

Traffic forking is a requisite to gain high scalability in combination with intelligent detection algorithms. Otherwise, if traffic would be completely analysed before being forwarded towards the protected SIP proxy, high delays would be introduced, especially during flooding attacks.

The Filter also contains scalability features. It might have multiple Analyzers assigned to it, to which it can forward SIP traffic. Analyzers rely on a consistent feed of SIP messages, therefore it must be assured that messages belonging to the same SIP session are processed by the same Analyzer. The Filter distributes them according to their transaction ID. Here the Filter node acts as a load balancer.

Passing messages also undergo inspection by the firewall. The firewall is controlled by rules generated at the Decider. Rules consist of conditions and an action to be taken, if the conditions are met. A condition can be any IP, UDP, TCP or ICMP property. Additionally, a condition may be a regular expression, which is applied to a SIP message. Thus, it is possible to decide about a message by its SIP properties.

5.1.3 Analysis Node ("Analyzer")

The Analyzer is the bottom half of the intelligence of the detection architecture. It analyses incoming traffic from the internet and outgoing traffic from the protected SIP proxy. It is necessary to keep track of both incoming and outgoing traffic because SIP is a stateful network protocol.

An Analyzer includes the low level functions for each detection algorithm in the system. A low level function is the part of a detection algorithm which must deal directly with the message flow, e.g. the SIP messages parser. Low level functions are expected to produce a result which can be used along with the results of other Analyzers to decide about the start of an attack, its status and its end. Analyzers are running in parallel to allow easy scaling of the analysis load, which depends on the number and complexity of the deployed detection algorithms, as well as on the expected network load situation. Results of the analysis run are forwarded towards the Decider.

5.1.4 Decision Node ("Decider")

The Decider is the top half of the intelligence of the security architecture. It gathers the output from all Analyzers and decides about the actual attack situation. It hosts an entity for each detection algorithm, which is capable of correlating the output of its specific Analyzer bottom half function. The Decider itself can also be scaled up, by deploying a dedicated Decider for each algorithm.

The Decider receives incoming reports from the Analyzers and delivers them to the corresponding algorithm-specific Decider modules. Modules decide whether an attack has been launched, and what can be done to counter it. Countering here means to create rules for the Filter, which will block all malicious traffic to the SIP proxy matching the rules created by a Decider module. In an attack situation, all traffic is still delivered to the Analyzers. Thus it is possible to decide when the attack is over, and to remove the previously created rules.

As an example for Analyzer and Decider co-operation, take DoS detection with the CUSUM algorithm [57]. In short, CUSUM monitors incoming sources, and detects if within a certain time frame a flow of unrecognised sources appear. For the implementation of CUSUM in VoIP Defender, only the source (IP address) and the reply type from the SIP proxy (Acknowledge / Deny) are of importance. The Analyzer module for the CUSUM algorithm only extracts this information from the stream of SIP messages and forwards only this essential information to the Decider. The actual CUSUM algorithm implementation is located at the Decider.

5.1.5 User Interaction

The administrative console is the central interaction point for the user. Here the status of each component is delivered to the user. Monitoring, detection

and prevention messages and alarms can be gathered at this point for user inspection.

5.2 General SIP DoS Attack Protection

In this section we present a method to detect high-traffic flooding attacks on SIP-based architectures using the introduced VoIP Defender framework. This method is based on an adapted model of the *SIP state machine specification* and is thus able to detect deviation from any normal operation. Through communication with the VoIP Defender firewall, this scheme allows blocking of offending traffic and thus keeping the service alive even under attack conditions. This method is especially effective for unintentional DoS attacks.

5.2.1 Background: The SIP State model

In the SIP communication context, two concepts are defined to describe a communication session between entities, that last a certain amount of time. These are *Dialogs* and *Transactions*.

A *dialog* is a peer-to-peer SIP relationship between two UAs that persists for some time. A dialog is established by SIP messages, such as a 2xx response to an INVITE request. Such a dialog can be terminated with a BYE message at a later time. Depending on the type of the session, dialogs can exist for a considerable amount of time, e.g. during a voice call or a video transmission.

A SIP *transaction* consists of a single request and any responses to that request, which include zero or more provisional responses and one or more final responses. As such, any dialog is comprised of individual transactions.

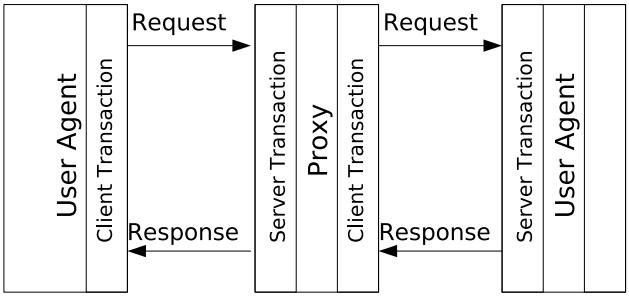

Figure 5.2: UAS / UAC transaction relationships

Transactions are modelled within the SIP RFC through a state machine. The state machine defines which events are valid and what and how the state

will change if a certain event occurs. State machines are defined both for a User Agent Client (UAC) and a User Agent Server (UAS). Additionally, they are different for INVITE messages and Non-INVITE messages, thus resulting in altogether four different state machines:

1. UAS INVITE state machine
2. UAS Non-INVITE state machine
3. UAC INVITE state machine
4. UAC Non-INVITE state machine

The relationship between UAC / UAS transactions is pictured in Figure 5.2.

5.2.2 Solution Approach: Finite Server Transaction State Machines

Specification

As the goal of a protection scheme is to analyse high traffic flows common to Denial-of-Service attacks, the solution being developed has to be resource-aware in order to not become a subject of a DoS attack itself [56]. Hence, to conserve memory, only SIP states which are absolutely necessary for analysis should be modelled. We refrain from dialog state modelling, as dialogs can potentially have unlimited durations.

We model the proxy's server transactions with the state machine *only*. With this modelling we can follow the protected server's operation for every incoming SIP message. We would not gain additional information by modelling proxy client transactions, as they are only triggered at the proxy by previous incoming messages. Furthermore, it would require additional resources for our implementation, which we try to avoid for scalability reasons.

Our SIP state machine model closely follows the SIP state machine for server transactions, including the same naming conventions. The two state machines (UAS INVITE and UAS Non-INVITE state machine) have six different states altogether - *Idle*, *Trying* (only for Non-INVITEs), *Proceeding*, *Completed*, *Confirmed* (only for INVITEs), and *Terminated*. State transitions are indicated by message events (SIP requests or responses) and timeout events (Timer *C*, *F*, *H*, *I*, and *J* according to RFC 3261). The schematic state machines are depicted in Figure 5.3.

However, the state model varies from the SIP state machine in order to be suitable for anomaly detection. The most notable difference is the

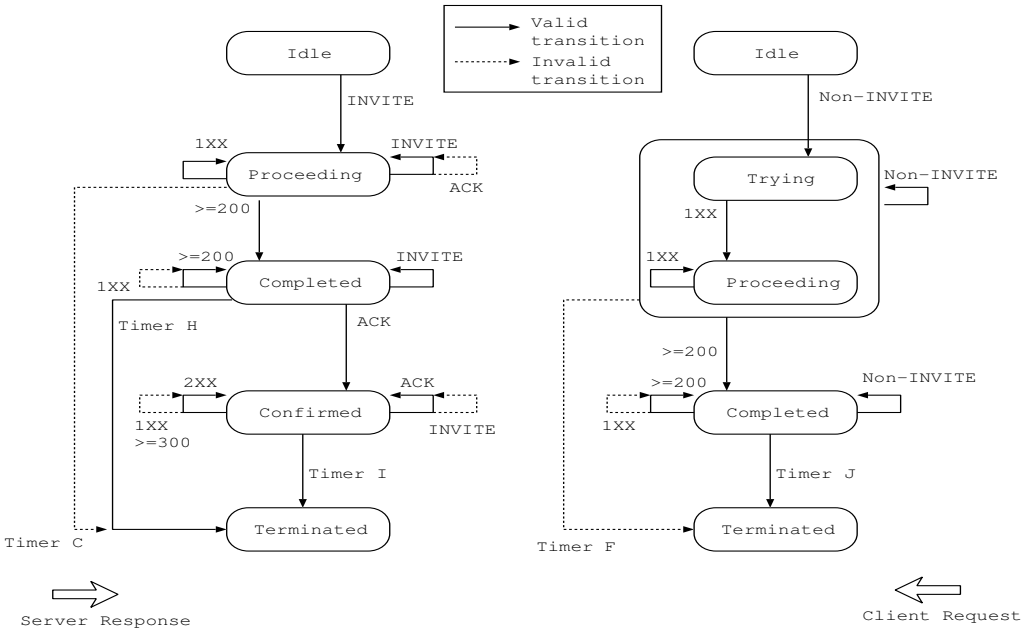

Figure 5.3: SIP transaction state models. Left: UAS INVITE, right: UAS Non-INVITE

inclusion of 2xx replies into the model. As specified in RFC 3261, an ACK is only considered part of the transaction if the final response was *not* a 2xx response. For our purpose, this differentiation is not necessary, hence we consider every ACK a part of the INVITE transaction.

As a second addition, state information is eventually released, hence each transaction will always be terminated. Otherwise, this could become a source for a self-inflicted DoS.

Finally, we explicitly model events that are not allowed within any state, e.g. an ACK should not arrive during the *Proceeding* state of an INVITE transaction and the server should not send provisional responses within the *Completed* state after sending the final response.

Application and Measurements Features

The modelled state machine implementation must have access to incoming and outgoing SIP messages as they are seen by the SIP proxy. This is achieved by VoIP Defender's passive network monitoring capabilities. Another possibility would be to implement this model directly at the SIP proxy, at the cost of increasing the proxy load.

With the application we are able to place each incoming message and corresponding proxy replies in context within the modelled state machine.

The state machine delivers detailed statistics of the current internals of the monitored proxy. By measuring different values we get an accurate picture of the ongoing traffic process. For each type of transaction (INVITE or non-INVITE) we maintain a set of features.

- *Active transactions.* We count the number of active transactions that are being processed at any specific time.

- *Active Replies.* A good hint as to the general status of the proxy are the types of replies it sends, especially the error replies $\geq$ 3xx. Combined in its response class (1xx, 2xx, 3xx, 4xx, 5xx, 6xx), we count the number of replies within each class.

- *New transactions.* We measure the amount of new transactions that are created within the proxy over time. A new server transaction begins when a new request from an outside client arrives (*Idle*$\rightarrow$*Proceeding* or *Idle*$\rightarrow$*Trying*). Client transactions started by the server are not taken into account. Each transaction has only one such transition. By differentiating new and currently active transactions, we can easily detect peak message flows to the proxy.

- *Transaction Termination.* It is similarly important to measure how a server transaction is terminated. Every transaction is terminated with a timeout event. We focus on timers C and H for INVITE transactions and timer F for non-INVITE transactions. Timers I and J are not considered, as they are always part of any regular SIP operation.

- *Intrastate events.* Different events that do not lead to a state change can occur within a transaction state. We measure the time required by the proxy to generate the first 1xx provisional response to an INVITE transaction, for example. The re-transmission messages within each state are also counted. Of special interest are out-of-specification events, i.e. events that are not valid for the SIP state model (as shown in Figure 5.3), including *Proceeding* $\overset{ACK}{\longrightarrow}$ *Proceeding*, *Completed*, *Confirmed* $\overset{INVITE}{\longrightarrow}$ *Confirmed* and *Confirmed* $\overset{1XX}{\longrightarrow}$ *Confirmed* state changes.

- *State transition.* Whenever the state machine proceeds from one state to another we measure how long the transaction has stayed in the previous state and which event causes the state transition. *Proceeding*$\rightarrow$*Completed* and *Trying*$\rightarrow$*Completed* are especially important as they result from final responses ($\geq$200) from the proxy. We distinguish

between them on the response class level (i.e. transition due to a 2xx or 5xx reply).

5.2.3 Attack Detection and Mitigation

Attack Detection

Through the gathering of statistical data, the state machine is able to detect anomalous behaviour, even though the monitored traffic complies to the SIP specification. The measurements are applied on three different levels i.e. each monitored message is evaluated within three different scopes.

- *Transaction Level.* Within this scope we follow the operation within each single transaction.
- *Sender Level.* We evaluate all the transactions that are from the same device based on its IP address.
- *Global Level.* With statistical summaries of all recent ongoing transactions, this allows us to give feedback on the current system state.

Transaction Level State Changes. The transaction state machine can follow and verify message flows generated within one single transaction. As it has information about the full SIP specification, it can detect non-conforming behaviour within each active transaction. Multiple transactions can be initiated by the same user agent.

The main target for this monitoring level is to detect unnecessary message flows within an ongoing transaction. Two cases are common:

- *Pending open INVITE transactions.* After a UA has sent an initial INVITE message to the proxy, it waits for provisional 1xx messages and / or a final answer from the proxy. An INVITE can take a considerable amount of time, depending on the callee's reaction. During this time the server transaction remains in the *Proceeding* state. While the UA is allowed to re-send his INVITE in this state in case it has not received a previous provisional 1xx-response, multiple re-sent INVITEs at this stage are generally a sign of an improperly configured UA or the beginning of a single-source message flow. Such messages can generate considerable traffic at the SIP proxy.
- *Closed transactions.* After a server transaction has moved to the *Confirmed* state, the proxy is required to maintain transaction state for several seconds (Timers I and J) to match any re-submitted messages

to previous transactions. Again, if multiple re-submitted messages are encountered, this is generally a sign of an improperly configured UA or a part of a single message flow, generating overhead traffic at the SIP proxy.

Sender Level State Changes. In the most basic case, a single IP address corresponds to a single user (one SIP phone connected to the proxy). There are exceptions to this rule, which are mostly PBXs which manage multiple users. Session Border Controllers (SBC) or NAT Gateways can also be a single source for multiple different senders, which are distinguishable by their port number. If a malicious user launches an attack from a single source, he or she is likely to generate different transactions during their flood to increase the processing overhead at the proxy, and thus the attack would not be recognised at the single transaction monitoring level. By evaluating the combined statistics of each single IP address, we are able to detect such anomalies.

Global Level State Changes. At the final level, we extend the monitoring of the local level to include all ongoing transactions. This becomes necessary if attacks are launched concurrently from different sources, which would be the case in a DDoS attack, for example.

At this level we can also gain knowledge of the health of the system. If, for example, the average measured time in the *Proceeding* state increases, this could be a sign that the SIP proxy is under heavy load. Combined with a high rate of measured session termination events through timer time-outs or a high response rate of 4xx error codes from the proxy, this is a good indication of a flooding attack.

Attack Mitigation

If a flooding attack is detected due to measurements at the transaction or sender level, the offending source can be identified as the transaction ID or sender (i.e. its IP address) are associated with the individual measurements. This information gained from the state model can be used to control a network firewall. Then we can set rules at the firewall to let some requests pass through or to reject them. Offending users can be distinguished on the transaction level or sender level, i.e. to prevent single messages of a certain type or all messages from one IP address.

Redundant requests can be selectively rejected by the filter using transaction level filtering, this is especially useful for an incorrectly configured UA sending redundant requests which disturb the proxy server (as outlined

in the previous section). All transactions with redundant requests can be identified using transaction-level monitoring. The traffic flow at the proxy is thus reduced while the ongoing transaction is undisturbed (see Figure 5.4).

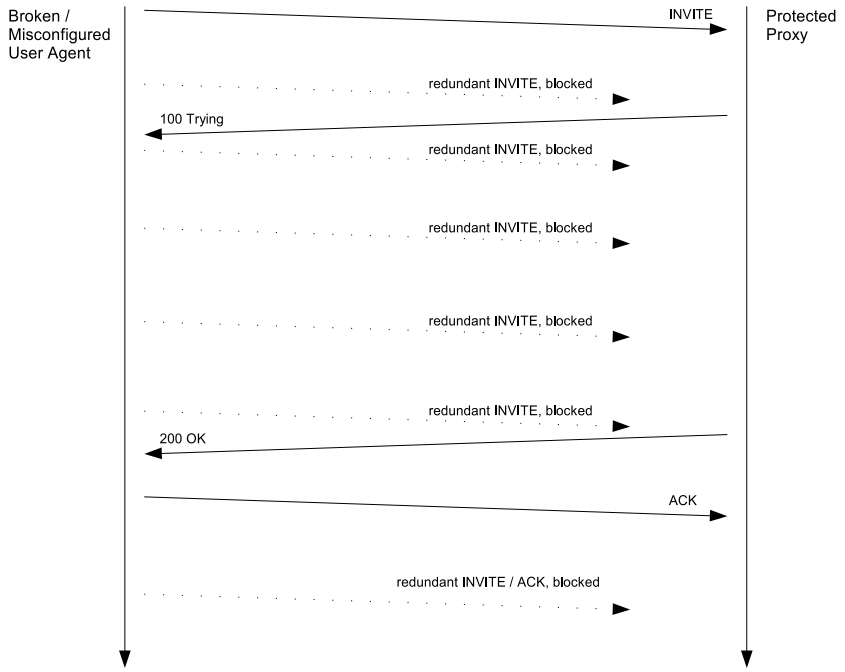

Figure 5.4: Detection and mitigation of attacks from misconfigured or broken user agents.

Global level state changes are monitored for attack detection. As it is not possible to directly distinguish between malicious and regular users at the global level, there is no direct way to use this information for attack mitigation. A possible mitigation possibility is outlined in the next section.

5.3 Distributed SIP DoS Attack Protection

From the current research point-of-view, there is no one definite solution to prevent all types of DoS attacks on SIP servers. In this section we propose a first step towards Distributed Denial-of-Service protection. We are introducing a lightweight security mechanism based on firewall *pinholing*, that effectively prevents many DDoS attacks on SIP servers. Pinholing is a common firewall technique, where the configuration of the border firewall of the protected network is dynamically updated depending on current network

traffic. The firewall is initially configured to block most incoming traffic, but allows some exceptions ("pinholes"), so that traffic with special characteristics can pass the firewall barrier. The proposed mechanism controls a firewall to generate pinholes that are necessary to effectively protect SIP servers. The approach shares some similarities with *greylisting* [140] used in email spam prevention, i.e. all incoming requests are initially held back by the firewall (they are "greylisted"), and only forwarded to the destination if the sending entity follows the SIP specification correctly. Hence, with our mechanism we can deny access to all distributed flooding bots that do not meticulously follow the specification.

The mechanism cannot handle all types of DDoS attacks, however it is especially effective against flooding bots that utilise spoofed IP addresses, a common technique that attackers use to evade detection. Spoofed addresses are difficult to handle with all current prevention mechanisms.

5.3.1 Background: Greylisting

Greylisting [140] is a complementary mechanism to white and black lists used in email spam prevention. When a message is received by an email server from a sender that is not listed on a white or a black list then the message is rejected temporarily. Senders that implement the Simple Mail Transfer Protocol (SMTP) [73] specification would hence correctly retry sending the message later. The re-transmitted message would then be accepted by the server and forwarded to the client. Thus greylisting is based on the assumption that SPAM software is rather simple and is optimised to send a lot of messages but does not care about re-transmissions. This way, messages from legal users would never be dropped unnecessarily and would always be forwarded to the receivers, albeit a bit delayed, while messages from spammers are dropped with very little effort.

5.3.2 Solution Approach: Firewall Pinholing

Our proposed mechanism works as follows: a firewall with pinholing capabilities is positioned in front of any SIP server that should be protected. The firewall is initially configured to block all incoming requests destined to the proxy, i.e. no pinholes are established.

Any arbitrary but regular UA sending a SIP request to the proxy (for example an INVITE or a REGISTER request) will have this request discarded at the firewall. However, afterwards the firewall establishes a pinhole so that further requests with a relation to this UA shall pass the firewall unhindered. Thus, further communication of this UA will not be affected.

Because of SIP's defined re-transmission algorithm, it is guaranteed that the UA will automatically re-transmit all SIP messages that have been initially blocked by the pinholing firewall. Consequently, the UA's re-transmission message will thus pass through the created pinhole to the SIP server, thus the communication channel is established. After some time of inactivity (i.e. no traffic passing through this pinhole) the generated pinhole is closed at the firewall. The schematic overview is depicted in Figure 5.5.

A DoS attack on the other hand strives for effective binding of all resources at the SIP server, e.g. through the establishment of as many different transactions as possible (for example by launching a memory depletion attack as described above). Hence, for effective resource depletion the attacker needs to initiate multiple different transactions, which will all be blocked by the access firewall. Only if the attacker UA conforms to the SIP specification and re-sends all previous requests (meaning, it also implements a message timeout detection scheme), will it be able to pass through the pinholing firewall.

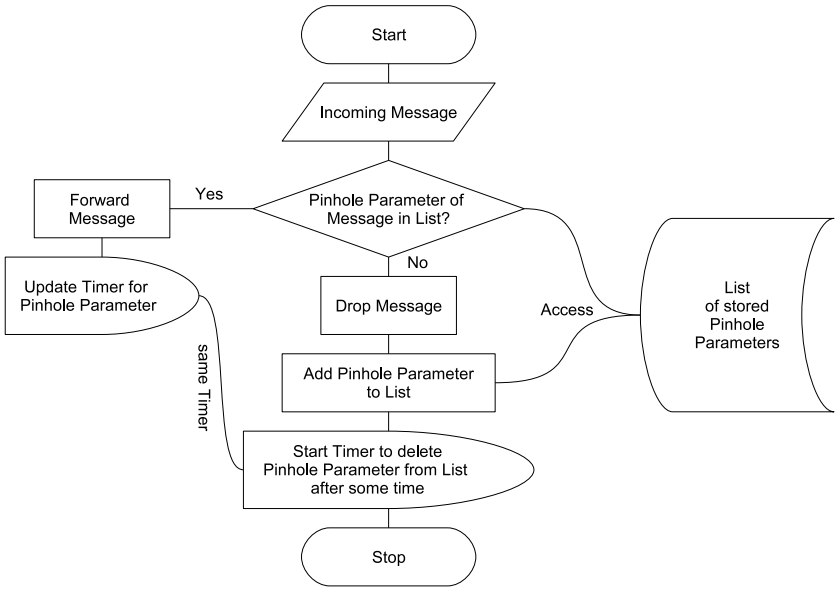

Figure 5.5: Pinholing process overview

5.3.3 Pinholing Parameters

A key component of this mechanism for successful prevention will be the method for creating pinhole rules at the perimeter firewall, e.g. which parameters should be considered for specifying the pinholes. Here we are considering different possibilities for the *pinhole parameter*.

If we consider a basic attack tool, it is likely that it will generate attack traffic with spoofed source IP addresses. This is necessary, as otherwise the attack will be filtered out by common security solutions. Most commercial SIP security products (e.g. Borderware SIP Assure [62]) use *threshold-based* prevention mechanisms against DoS. This is achieved by allowing only a limited number of requests from one source within a given time frame. However, such a mechanism is not effective against messages with spoofed IP addresses.

For the pinholing algorithm to be effective against such attacks, the pinhole parameter can simply be the source IP address, i.e. every first message from one given IP address will be dropped, while further requests from this same address will pass through (see Figure 5.6).

With this pinhole parameter, prevention is also possible for DDoS attacks with real IP addresses if the attack generation tools does not implement the correct SIP re-transmission method.

However, with the IP address as the only pinhole parameter, it would be ineffective if attackers were to initiate multiple requests with their real IP address or with random but fixed spoofed IP addresses. In this case only the first request would be blocked while all further requests could pass unhindered through the newly generated pinhole.

To cope with this situation, the pinhole parameter can also be modified to consider the transaction or session ID (i.e. evaluating the relevant `Via`, or `Call-ID` header fields and tags in the SIP message) as the pinhole parameter, thus only allowing messages from the same context to pass the firewall.

It is possible for an intelligent attacker to circumvent this method. However, to achieve this the attack tool needs to be more complex, and and thus becomes less effective: Instead of using the full given bandwidth capacity to generate different requests, it has to re-send previous messages. As an attacker will not know how many requests have to be sent to finally pass the prevention mechanism, its individual attack power decreases with every re-send: Assuming an attacker can generate 1000 attack requests per second, it would only be able to pass 500 individual requests if all of them have to be repeated. This number would further decrease if the attacker had to repeat requests more than two times.

The generated pinholes should later be removed from the firewall to free up resources and to speed up firewall performance. Note that for DoS protection it is not absolutely necessary to close the pinholes immediately: an attack can only "hit" an open pinhole by coincidence, and a more sophisticated attack would also pass an already closed pinhole. Also, if pinholes are closed too early, regular users are also affected, as they have to re-open the pinhole. As REGISTER messages are recurring messages sent by the UA, the pinhole should be at least open for the common register refresh time

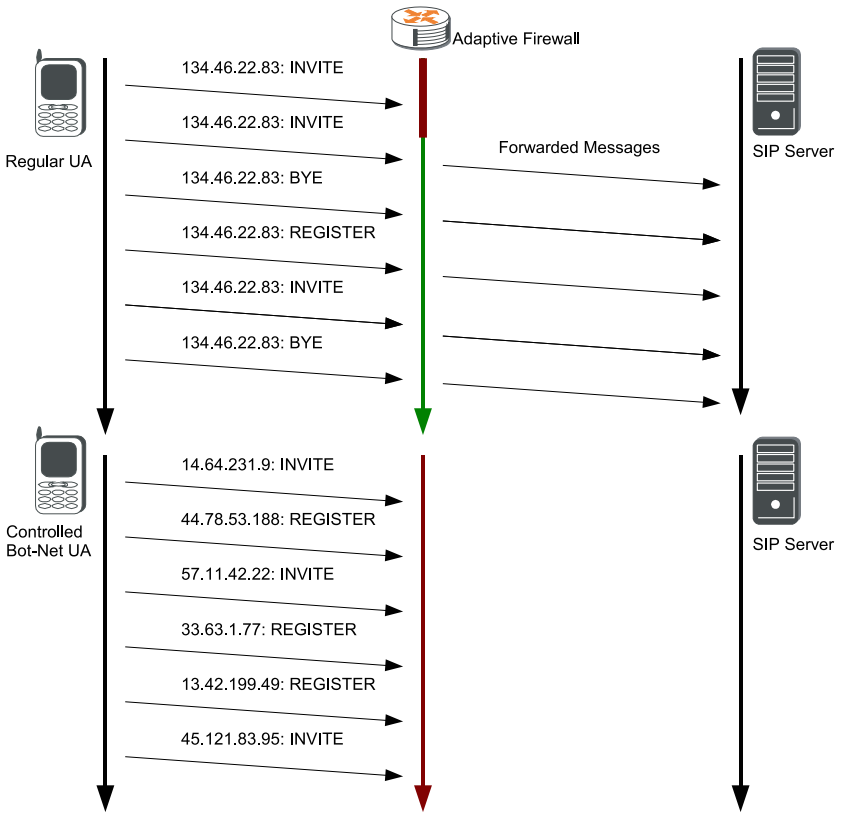

Figure 5.6: Pinholing overview, all new requests are blocked until message re-transmission.

(which defaults to one hour).

Note: The pinholing mechanism works for UDP transport, which is the dominant form of SIP transportation and likely to be used in attack scenarios. SIP does not use its own re-transmission feature when sending over a reliable protocol like TCP, here the re-transmission feature of TCP is used instead.

5.4 Combined SIP-DNS DoS Attack Protection

In this section we present a new kind of SIP DoS attack that is launched by utilising the Domain Name Service [141], on which SIP heavily relies on, which we call a *SIP DNS attack*. This a completely new type of attack, and it is easy to launch and slows down message processing at the target SIP proxy by a fair amount. We present a mitigation scheme specification which

is based on a non-blocking cache design.

5.4.1 Background: Domain Name Service

The Domain Name Service (DNS) is the basis for most current internet services available today, including web and email. It is a completely globally distributed and managed database, providing an essential service for Internet applications and users, i.e. name resolution, which is the mapping from human readable textual domain names (e.g. www.berlin.de) to a numerical IP address (e.g. 62.50.41.196). Whenever a user requests a domain resolve, there are generally two cases to distinguish:

1. *The DNS server knows the name mapping.* The name server might know the mapping because it is the authoritative name server for this domain. As such, all mappings for the domain are preconfigured for this domain server. The server might also know the name because it has resolved this address previously. Generally, in this case the mapping is still stored in the server's internal cache.

2. *The DNS server does not know the name mapping.* In this case the server will issue a recursive request to other name servers that might be able to provide an answer. The server will eventually receive a response, either containing the valid mapping or an error message that no mapping is possible. In the former case, the mapping will be stored in the server's internal cache for a limited period of time. The name server can also set a time limit for the query. If no answer is received within this limit, the address is considered unresolvable.

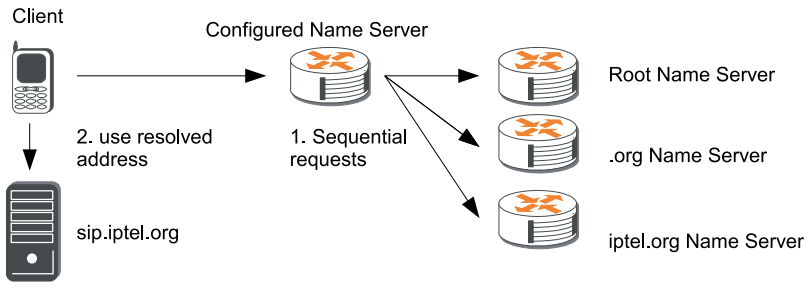

Figure 5.7: A procedure of DNS recursive request

In Figure 5.7, an example is given where an end user issues a DNS request of "sip.iptel.org". The name server will execute a series of recursive requests until finding the authoritative server of this domain (iptel.org). Finally, the authoritative name server will reply with the IP address of "sip.iptel.org".

5.4.2 DNS Usage in SIP Infrastructures

The Domain Name Service plays a key role in every SIP network at three following aspects [142].

1. Many of the header fields in a SIP message contain Fully Qualified Domain Names (FQDN) that need to be resolved for further processing from a SIP entity, especially when forwarding requests to other SIP entities. Relevant SIP message headers include for example, Contact, Request-URI, Via and Route.

2. To interconnect the PSTN with a SIP network, ENUM telephone number mapping [143] is used. In short, this allows the mapping of a PSTN telephone number (e.g. +1 234 567) to a valid SIP address.

3. SIP can utilise different transport level protocols (e.g. UDP or TLS). To find its right contact server in regard to the used transport layer protocol, a SIP entity will issue a DNS SRV request for the domain of the regarding SIP URI [144]. The response will contain one or more destination hosts that provide the required service.

In short, a SIP entity might query the DNS subsystem up to three times (ENUM mapping, server locations and address resolution) before it can actually process and forward a message.

5.4.3 Scope of the Attack

The goal of a DoS attack is to render a SIP server inoperable for as long as possible. While all kinds of SIP servers are affected by DNS attacks to some degree, such attacks are especially fatal for P-CSCFs in IMS networks and so called outbound proxies. In the context of IMS networks, a P-CSCF is responsible for receiving traffic from roaming users and forwarding it to the home domain of the users. While outbound proxies provide a similar functionality they are mainly used for NAT traversal reasons [22].

Whenever a SIP server encounters a fully qualified URI in a header field necessary for routing (e.g. Via or Route field), it issues a query to the local name server to receive a valid address mapping. On average it takes 1.3 DNS queries to receive an answer with the mean resolution latency less than 100 ms. However, due to configuration errors, these numbers can be considerably higher [145].

The SIP DNS attack targets this relatively high processing time. It is possible to disturb server operation with specially crafted SIP messages containing URIs that will cause an even higher processing time at the DNS

server by using a URI in a routing header (Via, Route and To headers or in the Request-URI) of which the attacker is sure that its mapping will not be in the cache of a name server or will trigger a request to an authoritative name server that has a common low response time, e.g. because of low bandwidth connection. The former case is easy to generate by adding random host names to the left side of a address domain. The latter case can be discovered by an attacker by querying different name servers and measuring reply times. As an example for such a crafted SIP message see Figure 5.8.

```
INVITE: SIP:u1@2d4fww.hard-to-resolve.domain SIP/2.0
Via: SIP/2.0/UDP unresolv.domain; branch=z9hG4bk29FE738
CSeq: 16466 INVITE
To: sip:u1@2d4fww.hard-to-resolve.domain
Content-Type: application/sdp
From: SIP: unknownuser@host.com; tag=24564
Call-ID: 116352524 3@10.147.65.91
Subject: Message
Content-Length: 184
Contact: SIP: unknownuser@host.com
<SDP Part not shown>
```

Figure 5.8: Example SIP message with unresolvable URIs

Such a message is a well formatted message that complies with the SIP standard in every respect and as such cannot easily be filtered out by a SIP server or an IDS. Issuing SIP queries with a variation of such URI's will stop operation at a SIP server depending on implementation and configuration for a considerable time, as the SIP server can only continue its operation after having received an answer from the DNS server. A DNS proxy searches for a limited time for a name mapping. When using BIND DNS server [146] this value of the time out is set by default to 5 seconds. If it does not receive any answer from the BIND DNS server within the timeout period, a negative reply is generated. The whole processing is shown in Figure 5.9. During the name resolve request, some of the SIP server resources will be blocked. Depending on the servers architecture this could be either memory or a process.

These attack can also be launched in combination with the SIP Authentication framework [129], which we show at another place [6].

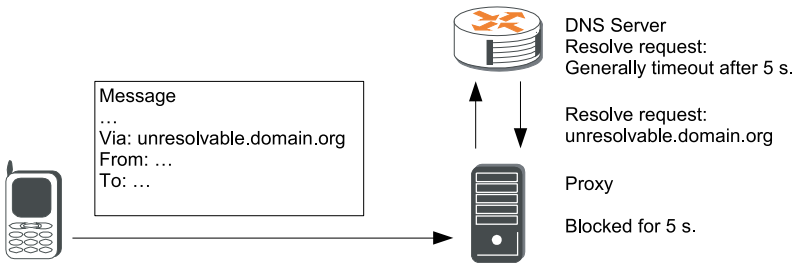

Figure 5.9: Attacking scenario by blocking SIP proxy with messages containing unresolvable URIs with a default BIND DNS setup

5.4.4 Basic Prevention Possibilities

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 indicate 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 multi-homed 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 behaviour 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 SUN's Solaris already include this feature. In case this is not included then the server should implement its own DNS cache.

5.4.5 DNS Implementations with SIP Servers

The basic options for using DNS in a SIP proxy are either synchronous or asynchronous usage:

- *Synchronous DNS implementation*: In this case the SIP proxy sends a DNS request and waits for an answer. While waiting the process that has issued the request would be blocked and would not be able to process new requests.

- *Asynchronous DNS implementation*: In this case the SIP proxy would issue a DNS request and continue processing new requests. Once the reply to the DNS request arrives or a timeout expires the proxy would be notified.

DNS and Synchronous SIP Servers

With a synchronous design, a SIP proxy would block in between sending the DNS request and receiving a reply to it. To reduce blocking effects, synchronous SIP proxies use parallel message processing by using multiple processes or threads with each process or thread responsible for processing one message synchronously. Such a design is depicted in Figure 5.10, similar to that presented in Figure 4.1. 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.

DNS and Asynchronous SIP Servers

Another option to design SIP servers is to use asynchronous processing. That is, after issuing a DNS request the server would not wait until an answer for the request was received but would queue the request in an event queue, save the data of the transaction, set the current operation on hold and move on to process the next request. When a reply for the request arrives the main process is notified and the waiting transaction is scheduled for continuation, thus

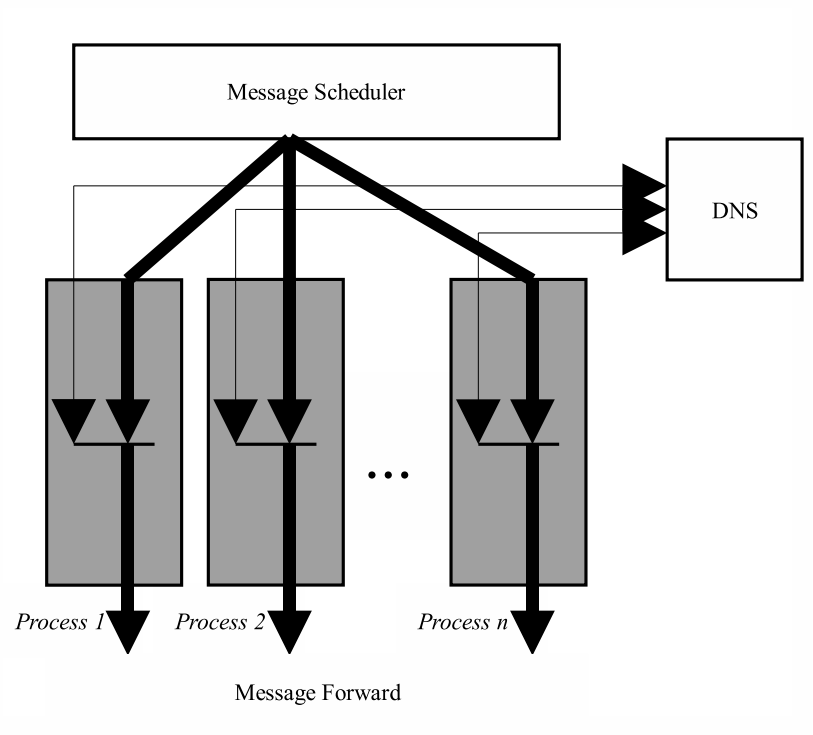

Figure 5.10: Parallel process design of the SIP proxy

eliminating a DNS blocking scenario. The procedure is shown in Figure 5.11. However, since the states of all unfinished domain name resolving requests need to be saved, the implementation complexity and memory requirements increase considerably. The server must support effective state suspend and resume capabilities, as each new DNS requests requires complete storing of the actual state into memory, and returning this state upon DNS resolve notification.

5.4.6 Solution Approach: Intelligent Unblocking DNS Cache

Neither design of the two design options can handle a SIP-DNS DoS attack. We therefore evaluate other protection methods.

The source of the described attack is the usage of fully qualified domain name addresses (FQDN) in SIP messages. Hence, FQDN should not be used unless necessary. To reduce the need to resolve DNS names in Via headers, RFC 3261 [23] suggests that each proxy that receives a request adds a received parameter to the Via entry of the request with the numeric IP address of the

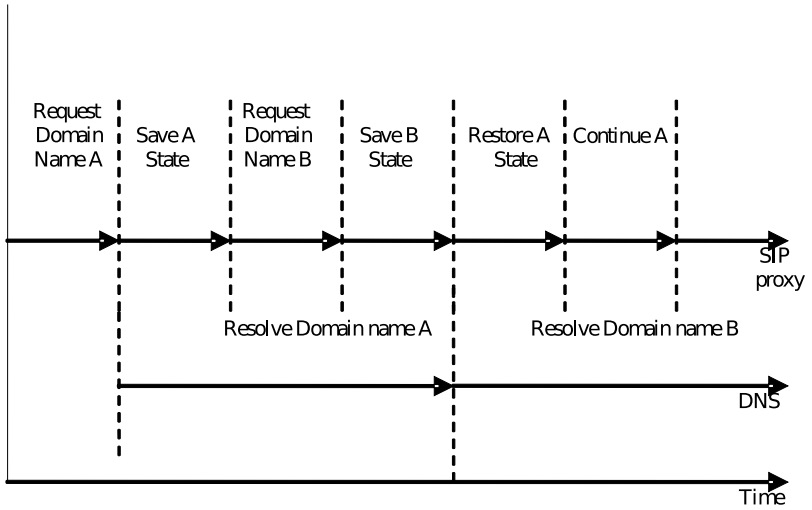

Figure 5.11: Procedure in an asynchronous scaling design

sending entity, thus eliminating the need to resolve this URI after receiving a reply.

However, the effectiveness of this mechanism is limited to reduce the usage of DNS when routing replies and can not be used for other SIP headers such as the Request-URI or Route headers.

Another approach for mitigating the effects of a DNS attack is to reduce the timeout value of DNS. While this would surely delay the collapse of the server it will not hinder it.

In the following we describe a scheme that is based on caching the results of successful DNS queries. This scheme is implemented for a synchronous working server. The choice of the synchronous servers stems mainly to the higher vulnerability of such servers to DNS attacks and hence the higher need for effective prevention mechanisms.

With the solution scheme we aim at minimising the time spent by a SIP server waiting for a DNS reply.

With this in mind, the SIP server is extended with a DNS cache. Whenever the SIP server requires the resolution of a DNS name it first checks if the name already exists in the DNS cache. If yes then it uses the cached information otherwise it issues a DNS request and caches the DNS reply with its TTL (Time to Live). According to the SIP environment, only successfully resolved A and SRV records will be cached. The records will remain in cache until their TTL expire or the capacity of cache is exceeded.

While different operating systems already provide DNS caches, they lack

dedicated features for optimal usage in a SIP network. As described, a SIP entity uses additional DNS records to locate other proxies, including NAPTR / SRV records [142], while a general operation system DNS cache does not consider such records for caching. Furthermore, a dedicated SIP DNS cache might need a different replacement policy than a general usage cache.

To ensure that a SIP server continues functioning even under a DNS attack, we define a blocking threshold. This threshold represents a certain parameter in the SIP server. Once the value of this parameter is above the blocking threshold, the SIP server stops issuing new DNS requests and relies solely on the content of the DNS cache for resolving DNS names. Hence, in this case, the SIP server will only be able to serve requests that contain already known DNS names. While this presents a limitation on the servers performance, it does ensure that under a DNS attack, the SIP server will continue to serve running sessions and new sessions destined to popular VoIP providers which are very likely to be cached. This threshold can be defined as follows: Assume a SIP proxy S processing messages in a synchronous manner, with n parallel processes as described in section 5.4.5. We define:

$$S_q(t) = \begin{cases} 1 & \text{a domain resolve call in process queue } q \\ & \qquad \text{but not returned at time } t, \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

We also define H as an indicator how many processes are concurrently resolving a domain name in time t, with

$$H = \sum_{q=1}^{n} S_q(t) \tag{5.2}$$

Hence the proxy will absolutely be blocked when $H = n$. To guarantee non blocking proxy operation, the following relation has to be met: $H < n$ at any time t. To achieve this we define a minimum operation threshold m, where m is reasonably small and $m < n$. While $H < R$, where $R = n - m$, the SIP server will function normally and issue a DNS request for each non-resolvable DNS name. Further, the SIP server will cache the results of successful DNS queries. Whenever $H \geq R$, the proxy is informed that further DNS resolve request will have a high possibility to cause a DoS due to proxy blocking. As a consequence, the proxy will not try to resolve any domain names that are not included in the cache. Instead, the proxy assumes these addresses to be unresolvable, and continues its operation. As soon as $H < R$, the proxy will again perform DNS lookups.

For the case of a SIP server designed to process messages in an asynchronous manner (see Sec. 5.4.5), the blocking threshold (R) could represent a percentage of the memory used by the server. Hence, once the percentage of the memory used by the server (H) exceeds a certain percentage of the overall available memory (R) to the server, the same behaviour as described above will be used. Combining the blocking threshold with a dedicated SIP DNS cache will effectively counter DNS attacks while keeping negative side effects for regular users to a minimum:

- As long as $H < R$ there should be no visible effect for regular users.
- In case of an ongoing attack, many regular users will not be affected: Current connections will be kept, REGISTER updates are executed without delay. Also, often new requests could still be served as long as the destination address is available in the cache.
- Only requests to destinations not currently in the cache will be dropped. These requests can not be handled at the moment.

As a result, this solution allows reduced operability under attack conditions. The amount of negative side effects for regular users mainly depends on the implementation of the caching replacement policy.

Chapter 6

Implementation

Following the specification of protection countermeasures as described in Chapter 5, we present in this chapter details how the specification is to be implemented in a prototype defence tool. The defence tool consists mainly of the VoIP Defender security architecture, which is used as a basis for two protection mechanisms: The state machine module to protected against general DoS attacks and the pinholing module to protect against DDoS attacks. The third module, the DNS cache operates individually from VoIP Defender; it is based on an available DNS caching solution.

6.1 VoIP Defender Architecture

The main security framework, VoIP Defender is implemented in C/C++ on top of the Linux Operating System on general PC hardware, i.e. it is a pure software-based solution. All three components (Filter, Analzyer, Decider) are self-contained and individually operating entities. Thus, all components can be deployed on one PC or distributed on different PCs.

6.1.1 Filter

The Filter is the core element of the VoIP Defender architecture. To improve performance, it is directly implemented in the Linux kernel. Its structure and the distribution of the entities over kernel and user space are depicted in Figure 6.1.

To achieve network transparency the Filter is acting as an Ethernet bridge, which intercepts all traffic on the MAC layer, and does not change network packets at all. If packets are allowed to leave the Filter, they leave exactly as they entered it. This also includes all MAC layer information.

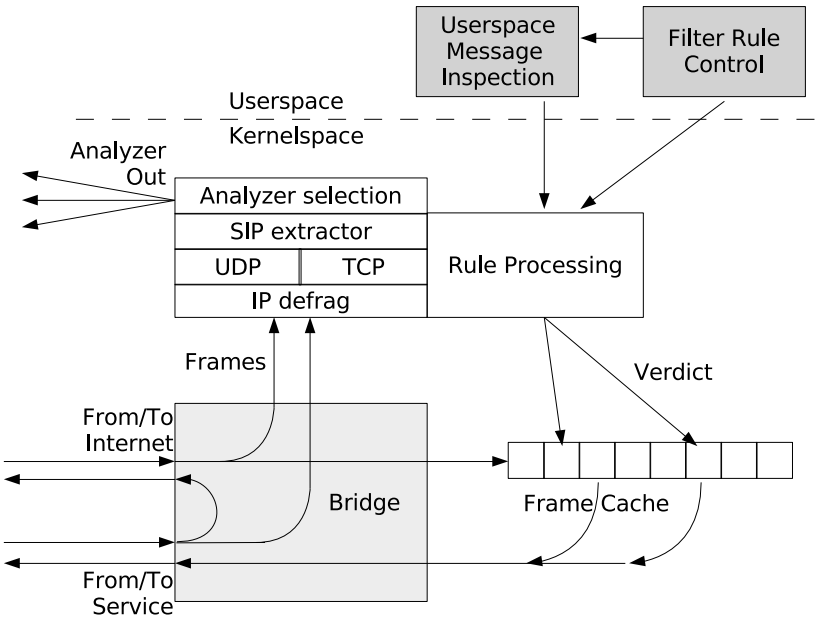

Figure 6.1: Filter entities

A packet which has been received by the bridge is stored in kernel space. It is then replicated and one copy is sent to the protocol reconstruction facility, another copy of the frame is stored in a frame cache, awaiting its verdict from the filter rule application.

As packets from the bridge are Ethernet frames, IP de-fragmentation has to be performed in order to gain full IP packets. After this step, UDP and TCP streams are reconstructed. During this process, references to the involved frames in the kernel queue are stored along with the reconstructed streams. From them, the actual SIP messages are obtained, which are fed into rule processing and sent to an Analyzer node as well, along with protocol meta data, like IP source addresses, number of fragments, timing information, etc.

Load balancing features are achieved as the Filter identifies the `Call-ID`, `To` and `From` fields from each SIP message and extracts the tags to form a unique session identifier for matching messages to sessions. In order to chose an Analyzer, it applies a hashing function to the `Call-ID` and the `From`-tag, computes a=*hash mod* n with n being the number of Analyzers available.

The extracted messages undergo inspection by the kernel space rule chains, as well as possible user space decision. For this purpose, a dedicated kernel/user space interface feeds a custom user space daemon with messages, which in turn sends back verdicts.

Rules are uploaded to the Filter via a user space daemon, which interacts with the kernel and the user space rule message inspection daemon.

Table 6.1: Condition types

	Kernel	**User Space**
IP	yes	yes
UDP	yes	yes
TCP	yes	yes
ICMP	yes	yes
regexp	yes	yes
scripts	no	yes

Table 6.1.1 opposes the condition types and possible execution points. The verdict of an applied rule is one of:

1. **Accept**: The message is harmless, pass it on.

2. **Drop**: The message is malicious, drop it.

3. **Continue**: Apply further rules until one produces a final decision about the message.

6.1.2 Analyzer

Figure 6.2 depicts the components of a single Analyzer node.

Incoming messages are centrally parsed by a high performance SIP parser. The parsed SIP message along with all received meta information and the original message buffer is then presented to all loaded detection algorithms. They can now apply their detection techniques and produce an algorithm-specific output. The Analyzer algorithm contains the scalable part of each algorithm. The output that an algorithm generates is relevant for the respective Decider module only, and has no meaning outside of this context. It describes the current attack situation, according to that specific algorithm. Therefore it is possible that an algorithm A detects a possible attack, while algorithm B does not. The results of each algorithm are then send to the Decider.

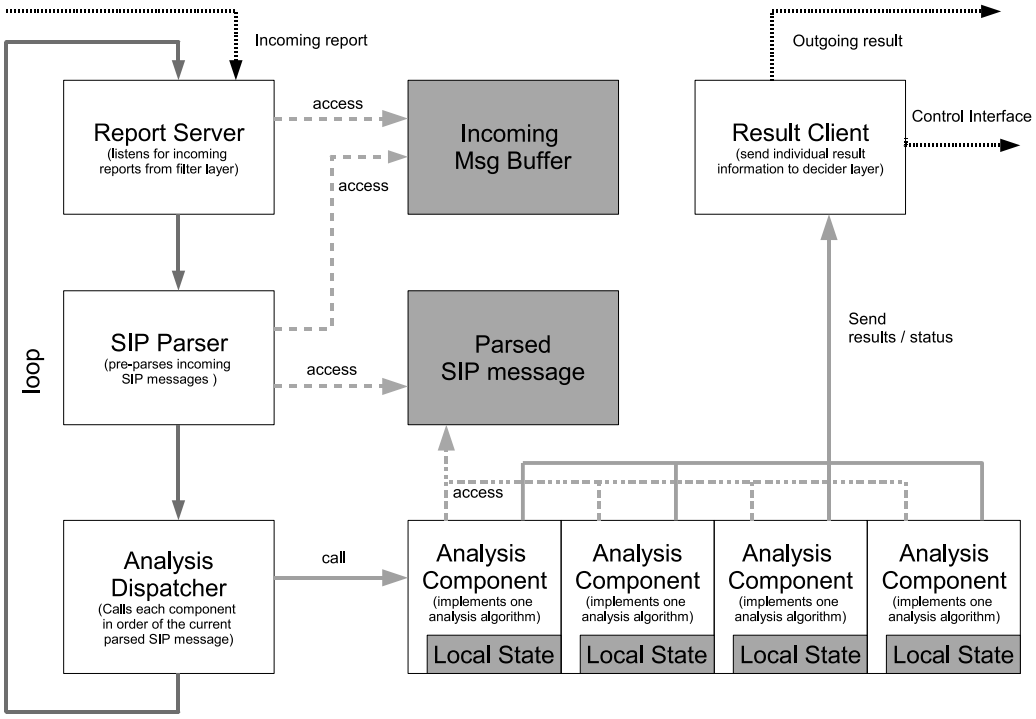

Figure 6.2: Analyzer architecture

6.1.3 Decider

Figure 6.3 shows the components of the Decider node.

The Decider receives incoming reports from the Analyzers and delivers them to the corresponding algorithm-specific Decider modules. These modules contain the non-scalable part of each detection algorithm. These modules decide, whether an attack has been launched, and what can be done to counter it. Countering means to create rules for the Filter, which will block all malicious incoming traffic toward the SIP proxy matching the rules created by a Decider module. In an attack situation, all traffic is still delivered to the Analyzers. Thus it is possible to decide when the attack is over, so the created rules can be removed.

The rule cache entity features static rules, which are created by human beings, or were taken from other systems, to suppress certain static types of attacks (pre-set rules). Rules consist of protocol-specific conditions, like addresses for IP, or port numbers for UDP/TCP, but can also be regular expressions for SIP-related content matching, or even complex scripts for user space based filtering engines.

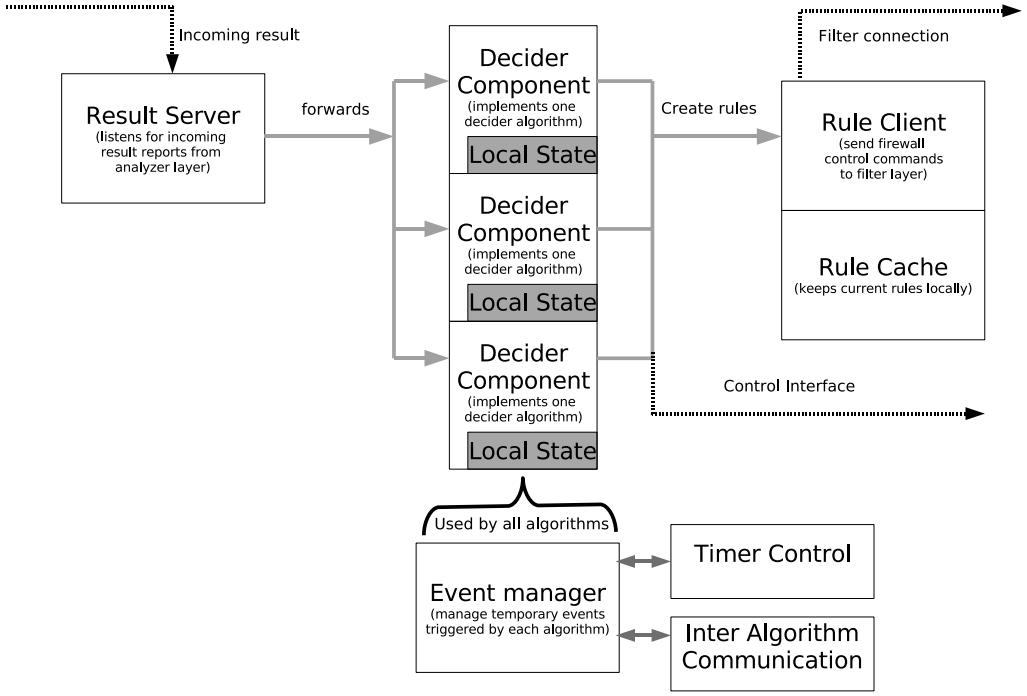

Figure 6.3: Decider architecture

6.1.4 Component Interaction

In this section we address the traffic flow between the individual components.

Filter Traffic reaches the Filter in the form of unmodified Ethernet frames. Incoming traffic is potentially filtered (packets are dropped), outgoing traffic is not suppressed. The observed SIP traffic is sent over multiple TCP connections to the Analyzers along with the following meta information:

- Time stamp when the first frame has been received
- Duration for reception of the whole SIP message
- IP information (source, destination, etc.)
- TCP information (ports, flags, etc.)

The traffic bandwidth sent out by the Filter is at least the sum of incoming and outgoing traffic, plus the meta information.

Analyzer It receives the Filter's SIP traffic plus meta information. Each Analyzer is connected to at least one Decider over a TCP connection. The information exchanged is specific for the installed algorithms. The protocol just allows dispatching of the algorithms.

Decider The Decider itself has an open TCP connection to the Filter node. The Decider may create firewall rules, consisting of conditions and a resulting verdict. These text-based messages are sent back to the Filter node as soon as a Decider module installs a rule. The same applies for rule removal. Protocol operations for rules are:

- Install/remove rules
- Set a default policy
- Query rules
- Wildcard delete rules

User Interface The Decider is able to send out REST messages [147] to give status updates for the user. Also, the central user console may be connected to each other layer through a telnet-based command interface to control VoIP Defender. Commands can be issued from a command line or generated through a GUI.

6.2 The State Machine Module

The state machine detection module is implemented using the VoIP Defender architecture. It monitors all incoming and outgoing SIP messages and updates its internal state with each intercepted message according to the SIP transaction model introduced in Section 5.2.2*f*. State changes can be triggered by an intercepted SIP message or by a timer event. The key part of successful attack detection and mitigation is the definition of appropriate measurement parameters, which are learned during test runs. Of particular interest are threshold values that differentiate attack traffic from regular traffic provided from the background traffic model. To define this, we have launched the system with the background traffic model only, and defined the attack threshold value for each measured parameter as 10 times higher than the normal measured value to minimise the false positive rate. In the end, this value should reflect the proxy's processing capabilities, i.e. flooding attacks need only be prevented if the proxy would not be able to process them otherwise.

Table 6.2: Measurement variables for the state machine module

Measurement variable	Meaning	Type
Active	Active transactions at sampling point (only at Global level)	integer counter
Replies_{class}	All replies of *class* (1xx, 2xx, ...)	integer counter
New	New transactions within one sampling interval	integer counter
Terminate_{timer}	Transactions terminated due to *timer* fired	integer counter
$\text{Intrastate}_{state,event}$	Events occurred which cause a transaction to stay in the same *state* due to *event* monitored, also differentiate between conform and non-conform events	integer counter
$\text{Retrans}_{state,event,(interval)}$	A special case of the previous Intrastate measurement variable. Counts the number of retransmissions within one *state* (this is independent of the sampling interval) (for global level measurements divide the accumulated sums up into an *interval* slot)	integer counter
$\text{Time}_{state,(interval)}$	Intrastate time the transaction stays in *state* (this is independent of the sampling interval) (for global level measurements divide the accumulated sums up into an *interval* slot)	float value
$\text{Transition}_{state_1,event,state_2}$	Number of transitions from $state_1$ to $state_2$ caused *event*	integer counter

According to specification, we gather and measure statistical data. Table 6.2 shows the measurement capabilities of our state machine.

For comparison reasons and to conserve memory, at the local and global monitoring level, measured integer counters are combined into a histogram. So, to get an answer to the question, how often an event x occurred, the monitor would produce an interval range instead of the exact answer. Defined interval ranges are $1, 2, [3, 4], [5, 8], [9, 16], [17, \infty]$. Likewise, interval ranges are $[0, 2^{-4}), [2^{-4}, 2^{-3}), \ldots, [2^3, 2^4), [2^4, \infty)$ for float values.

Sampling calculation is done at regular sampling intervals s, with the basic sampling rate to calculate statistics s_b is set to 1 s. To get a broader overview, statistical calculations are also individually done at $s_b * 10$ and $s_b * 100$.

Again, to not exceed memory at the monitoring entity, sampling data at user level is stored in a chained hash table with a fixed size. To free space in the hash table we have implemented a Least Recently Used (LRU) [148] replacement strategy.

We present an example of the meaning of the measurement features as introduced in Table 6.2: At the sender level for INVITE transactions, $\text{Intrastate}_{completed,request}$ would give us the number of requests during one sampling interval that occurred from the same sender while any of this sender's transactions are currently in the *completed* state. In this case only another INVITE method is allowed. At the global level for any transaction, $\text{Retrans}_{completed,reply,[5,8]}$ would show us how many transactions had 5 to 8 replies in the *completed* state before the transaction changes to another state.

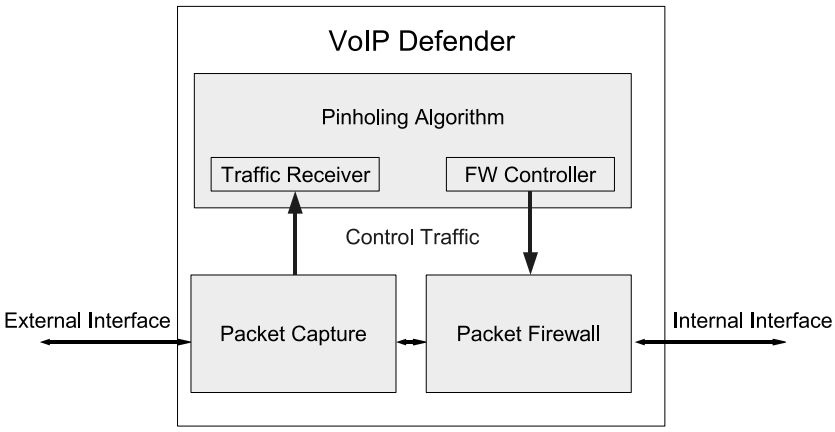

Figure 6.4: Pinholing setup

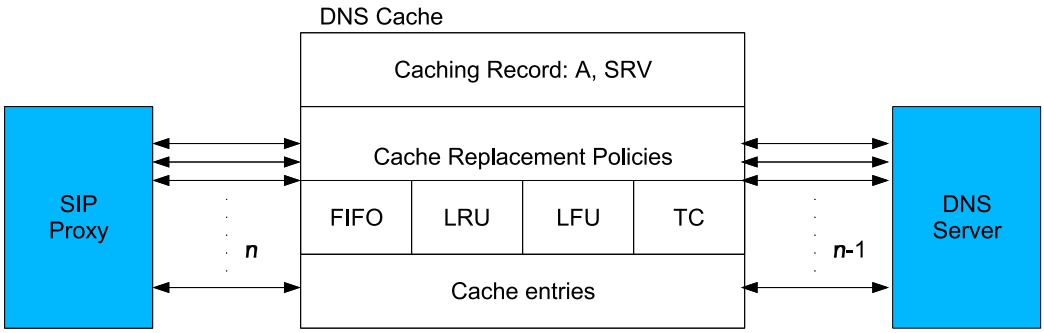

Figure 6.5: DNS cache implementation overview

6.3 The Pinholing Module

The Pinholing module is also implemented using the VoIP Defender architecture. For the pinholing mechanism, VoIP Defender captures all traffic to and from the SIP proxy and forwards it to the pinholing algorithm module. The pinhole database from Figure 5.5 is managed directly there. The VoIP Defender firewall controller is then responsible for installing and removing pinholing rules at the used firewall (see Figure 6.4). Rules are updated at the firewall as soon as the incidents occur.

Note, that while tests have only been conducted with VoIP Defender, the actual pinhole mechanism should also work in other security frameworks like IDS or SBC.

6.4 The DNS Cache

The Non-blocking cache design from Section 5.4.6 was prototyped on top of the Dnsmasq [149] DNS cache tool. The developed prototype operates with SER. As seen in Figure 6.5, the prototype supports:

- An "emergency process", which will only look up DNS records internally instead of forwarding requests to external DNS servers whenever $H \leq n-1$. In a pretest, we found that m did not affect the performance of the cache. Therefore, in the experiment, we set m fixed to 1. As seen in Figure 6.5, the cache could handle n requests in the same time while mostly, only $n-1$ requests could be forwarded to an external DNS server.

- Caching both regular DNS entries (DNS A records) and DNS SRV records.

- Different cache replacement policies such as first in first out (FIFO), least recently used (LRU), least frequently used (LFU) and time cost to replace old records [150, 148]. These algorithms are evaluated in the following chapter.

Chapter 7

Validation and Optimisation

In this chapter we show the feasibility of the specified methods. By running multiple tests under laboratory conditions with the prototype tools described in Chapter 6, we validate the correct operation of each proposed method.[1] Our focus lies on the validation of of the correct operation of each protection module, as specified in Chapter 5, however, correct operation alone is not enough to successfully counter the DoS attacks introduced in Chapter 3. We therefore begin this chapter with performance tests of the basic VoIP Defender architecture. We conclude with optimisation possibilities in the case where test runs show lower than expected performance. Unless noted otherwise, all performance measurements show the mean values of multiple test runs.

7.1 VoIP Defender Validation

The limiting factor for performance in the VoIP Defender architecture is the Filter, as this is the point where massive load in terms of network traffic (SIP streams) and processing power (rule application) is expected. We therefore concentrate on testing the limits of the Filter node.

7.1.1 Test Bed Setup

VoIP Defender is deployed as a prototype implementation in a private security network. We do performance measurements deliberately in a private network to ensure that only generated testing traffic is processed and no other traffic falsifies test results. The performance test bed setup is depicted in Figure 7.1.

[1] Results of this chapter have been published in [17, 16, 11, 7, 3].

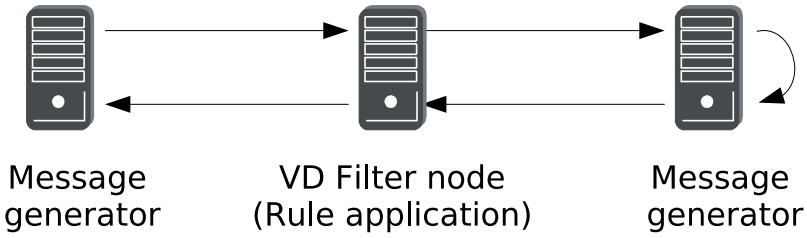

Figure 7.1: VoIP Defender performance test bed setup

A message generator can generate UDP messages of preset contents and measure the delay between sending them out and receiving them back from the target machine. For ICMP measurements we use the standard UNIX command `ping`, which uses ICMP echo requests and echo responses (message type 0 and 8) for delay measurements. The testing machines are connected with a Gigabit Ethernet network with switches between them.

7.1.2 Round Trip Time Delay

Table 7.1 lists round trip times for ICMP echo/response (`ping`) and UDP ping cycles. Measurement times are first given without VoIP Defender installed as reference, followed by setup times with the VoIP Defender Filter deployed with different numbers of installed rules at two different sending speeds. The rules are chosen not to trigger ever, thus it is made sure that for each packet the whole list of rules have to be applied. This test shows that even high numbers of rules influence the delay of network frames through the Filter node only marginally. Opposing that the number and complexity of regular expression, applied to UDP packet content, raises latency dramatically, as regular expression processing consumes much CPU power.

7.1.3 Throughput

Table 7.2 lists the recorded throughput rates along with the observed CPU load at the Filter. The data source sends out a 170 Mbit/s SIP stream, which passes the Filter on its way to the SIP server.

It turns out that simple, IP properties based rules are processed with minimal overhead, and yielding a high throughput rate. On the other hand regular expressions are turning down throughput much more than IP based rules, as they require much more CPU power. We cover regular expression based rules and performance improvements in more detail in Section 7.5.2.

Table 7.1: Measured round trip times (in ms)

Setup	1 Packet/s		10 Packets/s	
	ICMP 8/0	UDP ping	ICMP 8/0	UDP ping
bare wire	0.2	0.28	0.19	0.25
Filter, 0 rules	0.31	0.33	0.26	0.34
10 rules IP	0.28	0.36	0.31	0.29
100 rules IP	0.31	0.4	0.34	0.39
1000 rules IP	0.4	0.51	0.37	0.47
5000 rules IP	0.54	0.55	0.51	0.54
10000 rules IP	1.03	0.92	0.95	0.81
17000 rules IP	1.7	1.64	1.6	1.52
2500 rules IP/UDP	0.47	0.59	0.47	0.57
5000 rules IP/UDP	0.55	0.91	0.53	0.94
		regexp = ".*23[a|b].*"		
10 regexp	n.a.	107.03	n.a.	112.73
100 regexp	n.a.	1131.4	n.a.	-
		regexp = "abc[a|b].*"		
10 regexp	n.a.	0.73	n.a.	0.8
100 regexp	n.a.	2.92	n.a.	2.8

7.2 The State Machine

7.2.1 Test Bed

The state machine runs in the VoIP Defender security architecture. For this test, VoIP Defender operating with the state machine module are deployed within a security test bed consisting of multiple Dell Xeon servers running the Linux operating system. For the tests the components share workstation space inside Xen virtual machines. Incoming traffic and internal connections are routed over Gigabit Ethernet lines and switches. For the target proxy we again use SER. See Figure 7.2 for a schematic overview.

We use an enhanced version of the SIPp traffic generator tool to create background or "normal" SIP traffic to the proxy. Our enhancements to SIPp include the possibility to create random events in the scenario and diversifying reply messages using regular expressions. The background traffic generator consists of a caller group, a callee group and a register group. The register group is responsible for registering the other groups. The caller group

Table 7.2: Filter throughput

Setup	Throughput	Average CPU Load
no rules	170Mbit/s	0.00
100 rules IP	170Mbit/s	0.50
1000 rules IP	130Mbit/s	1.06
1 regexp rule	170Mbit/s	0.03
10 regexp rules	110Mbit/s	0.96
20 regexp rules	70Mbit/s	1.11

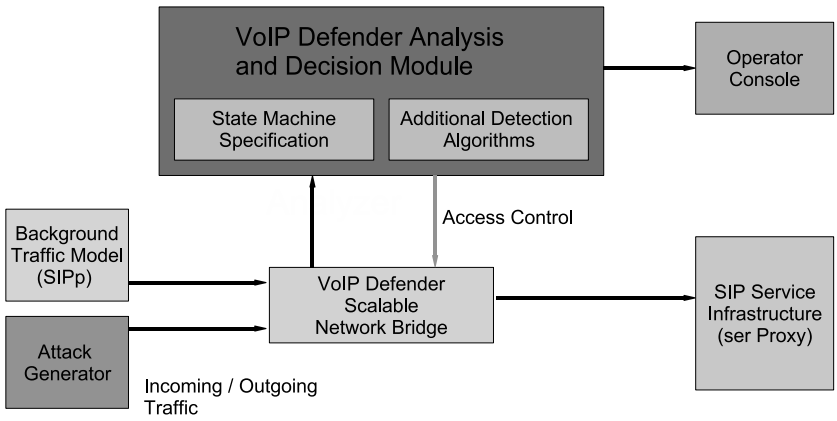

Figure 7.2: VoIP Defender state machine test bed setup

initiates random calls to users in the callee group. Calls are either accepted after a random time period by the callee group and finally terminated by one entity, cancelled by the caller or failed for some random reason.

We use both SIPp and a developed tool that is able to generate SIP messages with spoofed IP addresses to simulate DoS attacks. This traffic is also directed towards the proxy with the aim of disturbing proxy operation.

7.2.2 Testing Scenario Setup

To validate the specification-based SIP anomaly detection and mitigation approach, we create three testing scenarios. Within each scenario, the background traffic generator is configured to launch and respond to 40 REGISTER messages and 80 INVITE messages per second. Then we launch three different attacks against the protected SIP proxy.

1. Simulated broken UA attack (Attack 1). Within this scenario, a valid

user continuously generates and re-sends a valid and fixed INVITE message. As such, the transaction ID does not change with new messages. The responses are not acknowledged.

2. Single source flooding (Attack 2). This attack overwhelms the target proxy with multiple invitation requests to an unregistered user, aiming to degrade server operation. As such, the transaction ID does change with each new message.

3. DDoS INVITE attack (Attack 3). The same as the previous attack, however, through our IP spoofing tool each new message seems to originate from a different sender.

Table 7.3: State machine measurements showing maximum value for sampling interval, B = background traffic (80 msg/s), A = attack traffic (80 msg/s)

	Sender Level				Global Level			
	B	A1	A2	A3	B	A1	A2	A3
Active					258	1	2560	4177
New	82	1	82	1	82	1	82	79
Replies_{4xx}	32	1	82	1	32	1	82	74
Replies_{5xx}	0	0	0	0	0	0	0	0
$\text{Terminate}_{timer_c}$	0	0	0	1	0	0	0	31
$\text{Terminate}_{timer_h}$	0	1	161	1	0	1	161	144
$\text{Intrastate}_{completed,request}$	0	82	0	0	0	82	0	0
$\text{Intrastate}_{completed,reply}$	1	82	648	1	1	82	648	564
$\text{Retrans}_{completed,request}$	1	2727	0	0	0	2719	0	0
$\text{Retrans}_{completed,reply}$	1	2727	8	8	1	2727	8	8
$\text{Retrans}_{completed,reply,[5,8]}$	0	0	161	1	0	0	161	144
$\text{Time}_{proceeding,(0,2^{-4})}$	0	1	82	1	0	1	82	74

We run the tests in several different configurations. First, we launch background traffic and attack traffic individually (at 80 INVITEs/s) to show how they both differ (see Table 7.3).

Then we perform flooding tests *in combination* with the background model running at 80 INVITEs/s. Flooding scenarios are conducted with a significantly higher network load (500 and 2000 INVITEs/s) to reflect a real DoS threat (see Tables 7.4 and 7.5). To differentiate between normal and attack conditions, we deactivate the firewall filter and alarm logic at first, then activate it for attack detection and mitigation.

Table 7.4: State machine measurements showing maximum value for sampling interval, B = background traffic (80 msg/s), A = attack traffic (500 msg/s)

	Sender Level				Global Level			
	B	B+A1	B+A2	B+A3	B	B+A1	B+A2	B+A3
Active					2358	2327	16761	27041
New	82	52	532	84	82	82	616	608
Replies_{4xx}	32	28	136	16	32	28	139	386
Replies_{5xx}	0	0	423	78	0	0	490	506
$\text{Terminate}_{timer_c}$	0	0	1	1	0	0	0	0
$\text{Terminate}_{timer_h}$	0	1	1009	1	0	1	1009	893
$\text{Intrastate}_{completed,request}$	0	509	0	0	0	509	0	0
$\text{Intrastate}_{completed,reply}$	1	509	946	1	1	509	946	973
$\text{Retrans}_{completed,request}$	0	16999	0	0	0	16999	0	0
$\text{Retrans}_{completed,reply}$	1	17007	8	8	1	17007	8	8
$\text{Retrans}_{completed,reply,[5,8]}$	0	0	245	1	0	0	245	870
$\text{Time}_{proceeding,(0,2^{-4})}$	0	1	531	78	0	1	594	536

7.2.3 Results

At first, we run each attack independently, in order to see its "signature". Table 7.3 shows clearly that regular SIP traffic (B) generates a completely different pattern at the state machine model than malicious traffic (A1-A3), even when the traffic is generated in both cases with the same amount of messages. Several measurement variables show zero or very low values for normal traffic, while the same variables show high values under attack conditions.

Thus, attacks can easily be differentiated from normal traffic by monitoring these variables. As seen in Table 7.3, normal traffic does not generate an abundance of timeouts (*Terminate* events), redundant messages that keep a transaction within one state (*Intrastate* events), redundant retransmissions (*Retrans* events) or different processing times (*Time* events). On the contrary, each tested attack produces a large value of at least one measurement variable, either at the sender or global level. In particular, the clearest indication for attack 1 would be the high amount of request retransmissions which trigger a high flood of server reply retransmissions (sender level, $\text{Retrans}_{completed,request}$: 1 vs 2727 occurrences). A clear indication for attack 2 is the large amount of server replies, due to the open transactions generated by this attack (sender level, $\text{Intrastate}_{completed,reply}$: 1 vs 648 occurrences). Also a lot of time-outs are generated here (sender level, $\text{Terminate}_{timer_h}$: 0 vs 161 occurrences). The pattern for attacks 3 is similar to that of attack 2, albeit only at the global measurement level (global level, $\text{Intrastate}_{completed,reply}$: 1 vs 564 occurrences; global level, $\text{Terminate}_{timer_h}$: 0 vs 144 occurrences). If more measurements are taken into account, the

Table 7.5: State machine measurements showing maximum value for sampling interval, B = background traffic (80 msg/s), A = attack traffic (2000 msg/s)

	Sender Level				Global Level			
	B	B+A1	B+A2	B+A3	B	B+A1	B+A2	B+A3
Active					2358	2328	64674	105981
New	82	82	2142	105	82	82	2227	2722
Replies_{4xx}	32	32	1351	6	32	32	1354	947
Replies_{5xx}	0	0	2052	101	0	0	2134	2349
$\text{Terminate}_{timer_c}$	0	0	0	1	0	0	0	567
$\text{Terminate}_{timer_h}$	0	1	4003	1	0	1	4003	3557
$\text{Intrastate}_{completed,request}$	0	2044	0	0	0	2044	0	0
$\text{Intrastate}_{completed,reply}$	1	2044	1360	1	1	2044	1360	1699
$\text{Retrans}_{completed,request}$	0	67994	0	0	0	67994	0	0
$\text{Retrans}_{completed,reply}$	1	68005	8	8	1	68005	8	8
$\text{Retrans}_{completed,reply,[5,8]}$	0	1	1865	1	0	0	1865	1602
$\text{Time}_{proceeding,(0,2^{-4})}$	0	1	2136	101	0	1	2218	2397

accuracy of the detection would increase further.

Thus, both attacks 1 and 2 can be detected at the sender level, while attack 3 can only reliably be detected at the global level. As outlined already in Section 5.2.3 mitigation is possible for attacks A1 + A2 by temporarily blocking offending senders or by just blocking single messages from one sender. Comparing the measurements from the transaction level with those at the sender level, we can even differentiate between offending malicious hosts and regular high-traffic hosts like NAT Gateways or SBCs.

Table 7.3 only shows figures where either normal traffic or attack traffic is run separately. The combination of background traffic with high volume flooding at the same time confirms the signature (Tables 7.4 (500 msg/s flooding), 7.5 (2000 msg/s flooding)). Again, a huge amount of intrastate and retransmission events clearly distinguish offenders from normal traffic at the sender level for attacks 1 and 2. With the VoIP Defender firewall, these offenders can easily be blocked after detection. Particularly interesting is the high amount of $Replies_{5xx}$, showing an overload at the SIP proxy. While the proxy refuses to process further messages, our scalable state machine keeps track of the current network condition even under load.

This range of tests have a focus on DoS flooding attacks. It is likely that our proposed method will be able to detect other kinds of intrusions, too. For example, probing attacks will also generate open sessions at the proxy. These will be visible at the state machine, albeit with a lower intensity as with flooding attacks.

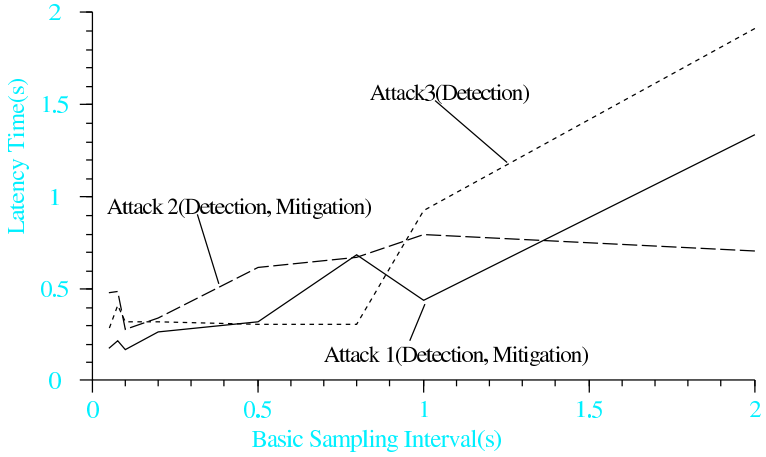

Figure 7.3: State machine latency time for detection and mitigation at different sampling intervals (flooding rate: 500 INVITE Msg/s)

7.2.4 Latency Time and CPU, Memory Usage

For performance evaluation, we also measure latency time, CPU load and memory usage of the state machine implementation. Latency is defined by the time after the attack is launched until the VoIP Defender state machine has detected the attack and blocked the offender, if possible. Latency times for attack detection and attack mitigation are shown in Figure 7.3. For latency calculation, the beginning of the attack has to be determined by the Filter node, while the reaction to the attack is determined at the Decider node, which leads to synchronisation problems. Thus the measurement values might not be completely accurate.

Obviously, latency time depends on the setting of the basic sampling rate s_b. With the default value for $s_b = 1$, the latency time is always below 1 s. Only in the case that the attack is launched exactly after a new measurement period is started, will the latency be close to 1 s. This shows the immediate response capabilities of our implementation. After experiments with different values for s_b, we see that attacks 1, 2 and 3 are detectable with a minimum latency time of under 0.3 s if s_b is set to 0.1 s. Mitigation for attacks 1 and 2 becomes effective within 0.3 s. With $s_b < 0.1$ s we are usually not able to gather enough data to reliably detect an attack in one sampling period, hence the latency time increases. With the increase of $s_b > 1$ s, the detection latency time also increases.

While a low sampling interval s_b is thus advisable for fast attack detection,

it increases the IDS load considerably as more calculations have to be made during the same interval.

For resource utilisation testing, we set up a simple INVITE flooding model, to test the limits of our implementation. CPU load and memory usage are shown in Figure 7.4. From the figure we can see that the CPU processing power is the limiting factor, and not RAM usage. The state machine in this set up can process around 2800 msg/s. Note, that the testing system is inside a virtual machine, and performance is expected to be higher if physical hardware is used.

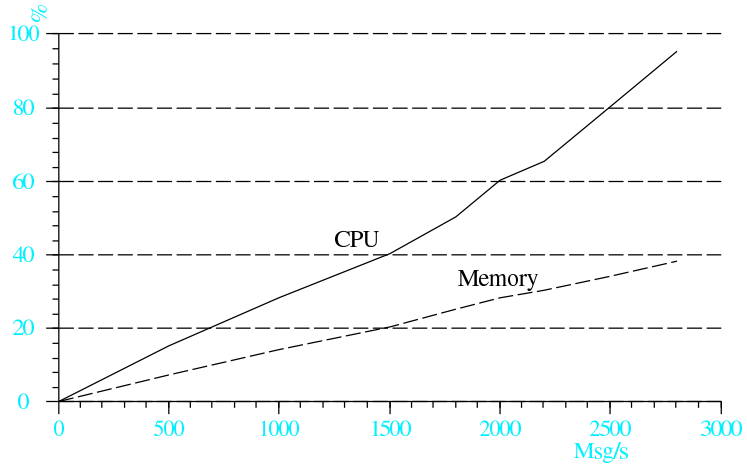

Figure 7.4: State machine CPU load and memory usage (Intel Xeon CPU 3.2Ghz, Memory 512 MB, inside Xen Virtual Machine).

7.3 Pinholing Validation

The pinholing testing is done using the same test bed as described in the previous section. We create an operation evaluation scenario and a performance evaluation scenario. In the operation evaluation scenario we proof the general operation of the mechanism, whereas in the performance scenario we push the implementation to its limits.

7.3.1 Operation Evaluation Scenario

For the operation evaluation scenario, we launch the SIPp background generator from multiple hosts to register different UAs at the SIP proxy and

initiate random SIP INVITE requests. Then we start an attack with the attacker tool, by generating random requests with spoofed IP addresses starting at a rather low rate of 20 calls / second. During the entire attack the goal is to establish all user-initiated requests from SIPp (i.e. no *false positives*) and to block all attack-generated requests (i.e. no *false negatives*) at the firewall. Rules are generated at VoIP Defender's firewall controller in real-time: As soon as a new SIP request is encountered, a new firewall pinholing rule is created and forwarded to the firewall to be installed there.

The test shows that all regular users are able to successfully pass the protection solution. In the next step we measure the delay in processing a regular user's call, i.e. by taking the time between the initial request from the UA and the received answer from the proxy. Without the protection solution established, this delay is on average 0.11 s in the test bed. With the protection solution established, this delay increases to an average value of 0.61 s. This delay is of course due to the first request being blocked by the firewall, and only the second re-transmission request being able to pass the firewall pinhole. The SIP specification states that the first re-transmission message should be generated after $T1$ s, where $T1$ is the calculated RTT or 500 ms, which is exactly what we witness in the test (Note, that SIPp does not seem to do an RTT calculation and just uses the default value of 500 ms for $T1$).

7.3.2 Performance Evaluation Scenario

In the second test we measure the performance of the mechanism and its influence on latency. From the previous test we know already that the system works correctly. Hence, it is sufficient to launch the attack alone to generate a high load of requests. We evaluate if the firewall can handle all requests, i.e. dynamically update the firewall table in real-time as requests arrive.

In this test, we configure the attack generator to emit 10000 SIP requests at a rate of 500 msg/s, as such a rate would already put a provider system under high stress. Note that the limiting factor here for the proxy is mainly processing speed and memory consumption, and not bandwidth usage. All tests are repeated 10 times, to minimise measurement errors.

The results from this scenario can be seen in Figure 7.5. The figure shows that VoIP Defender indeed generates all 10000 firewall pinholes as expected, i.e. there are no false negatives.

However, as more rules are added to the firewall, rule adding latency increases considerably. In a real-time scenario, rules should be effective as soon as new SIP requests are encountered. In the introduced scenario with 10000 SIP requests at a rate of 500 msg/s, ideally after about 20 seconds

all rules should have been installed at the firewall. However, the tests show that the last rule is added nearly 3 minutes after the last attack request was generated. This is a considerable delay and would have grave consequences for network traffic. In the theoretical case that only one regular user would contact the SIP proxy just after these 10000 attack messages have been generated, it would take the client three minutes before a contact with the SIP proxy would have been established. This is however a theoretical value – according to the SIP specification, a calling UA terminates an INVITE request after around 30 s (depending on RTT, and given that the calling human operator would not have given up before). During this time the UA would also have generated ca. 7 re-transmission requests. Evidently, this is not an acceptable use case, and due to the additionally generated re-transmission requests, the load on the defence node would even be increased.

Upon examining the setup, we identify the iptables-based firewall engine controlled by VoIP Defender as the performance bottleneck. While rules are generated without delay at the firewall controller in VoIP Defender, iptables cannot process these requests in real-time. The standard Linux firewall iptables works considerably well with a small set of static rules, but has known performance problems with an increasing rule set size and rules that are dynamically updated [151, 152, 153]. We investigate optimisation strategies in Section 7.5.1.

Within our test runs, there is no great difference in latency performance between different iterations of the test, i.e. derivation from the average case was minimal. This further confirms that the encountered low performance is not due to network problems and indeed results from iptables' rule processing.

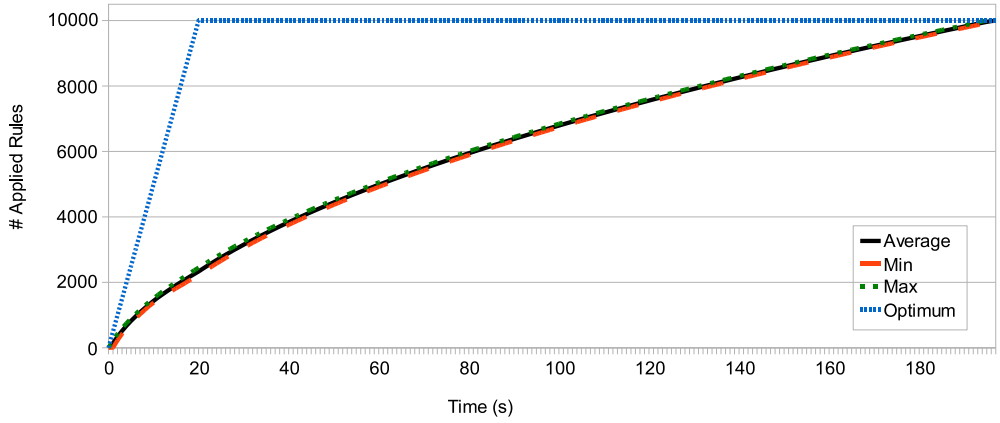

Figure 7.5: Time to install 10000 rules at the pinholing firewall

7.4 DNS Cache Validation

7.4.1 Test Bed Setup

We create a new test bed for the DNS Cache evaluation, as it is not an integral part of the VoIP Defender architecture. Within this test bed we show the effectiveness of the attack and evaluate countermeasures against it. The DNS Cache test bed consists of five main components.

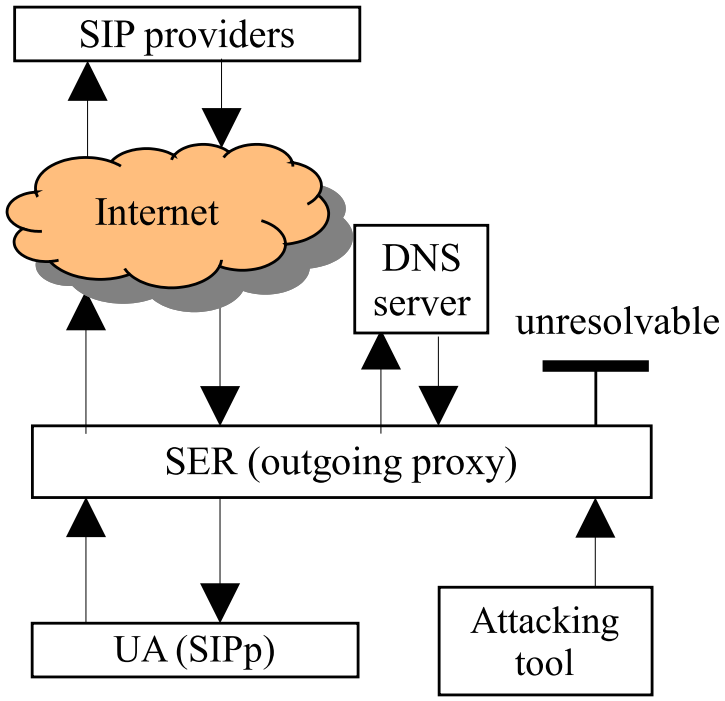

Figure 7.6: DNS cache test bed architecture

1. A SIP proxy as the main target of the attack. In the test scenario the attacked server acts as an outbound SIP proxy. Hence, all messages to or from a caller have to go though this proxy. We have again used SER for this task.

2. A local DNS server. The SIP proxy is configured to contact this server for DNS requests.

3. A self-implemented attack tool generating SIP messages containing unresolvable domain names. This tool can continuously send SIP messages with different hard to resolve domain names to the proxy.

4. User agents (UA) representing legal users that register themselves on remote SIP servers. The UAs are realised using the SIPp message generating tool. We use SIPp to simulate regular SIP REGISTER traffic, consisting of REGISTER requests with different kinds of responses from remote servers.

5. External SIP providers. We choose 100 different SIP providers from all over the world, mostly located in Europe and North America. The user agents are registered there. Every external SIP provider is located at a different domain.

The test bed (proxy, user agent, and attack tool) is established on Pentium D double processor machines with 1 GB RAM running on Linux Operating System, equipped with 100 Mbit Internet access.

The logical structure of the test bed is shown in Figure 7.6. We perform testing with the following steps.

- The outbound SIP proxy can be configured to have different parallel processing queues n, with $2 \leq n \leq 64$.

- We configure our UA to send continuously REGISTER messages from our local network to external SIP proxies. The external SIP register addresses are given to the UA in textual representation, hence the proxy has to resolve the domain before it can forward any request.

- The attacking tool is configured to send crafted messages containing hard resolvable domain names to the local outgoing SIP proxy. It is configurable by the attacking interval i (in seconds) between two attacking messages.

To measure the proxy performance, we send out 5000 REGISTER messages from the attack tool and count the number of responses (r) the local proxy can process. If we can get any kind of response from a remote SIP server, the domain name of the server is successfully processed by the proxy. Ideally, all 5000 messages can be processed at the local SIP server. 50 INVITE messages to different external SIP servers which are chosen randomly are sent in 1 second and every outgoing message from the UA is routed through the local SIP proxy. Table 7.6 shows the variables of the experiment.

All measurements are the average value of 10 runs of the tests.

Table 7.6: DNS Cache test bed parameters

Variable description	Variable symbol
Parallel processing queues of the proxy under attack.	n
Time interval between two attacking messages sent from the attacking tool to the local proxy.	i
Number of reply messages received by the UA	r

7.4.2 Feasibility of the Attack

To evaluate the performance of parallel processing under an attack, we perform an experiment based on our test bed with different parallel processing queues n and different attack intervals i.

The result is shown in Figure 7.7. With few parallel processing queues ($n < 8$) less than 20% of all potential messages can be processed, even with only one attack message per second. 64 processing queues are needed to adequately cope with the same attack speed of one malicious message per second. However, decreasing the attacking interval down to 0.001 seconds (1000 attack message per second), even 64 parallel processing queues are completely starved.

Generating 1000 messages per second is easily achieved with a DDoS attack, where an attacker controls hundreds of slave machines [28]. For example, Hussain et al. [154] demonstrated an attack scenario with 100.000 malicious messages per second. With that setup, even a proxy with 64 or more processing queues would be totally blocked. On the other side, configuring even more parallel processes costs more memory and CPU resources, possibly leading to system overload and hence another type of DoS.

We have found out that a SIP attack launched at a SIP proxy with asynchronous processing, running on a machine with 8GB of RAM all memory can be depleted in about 30 seconds, resulting in a complete lock-down of the machine [155].

7.4.3 Evaluation

Performance Evaluation

In order to verify the efficiency of the caching scheme, we run several endurance tests. In these tests 10,000 SIP REGISTER messages are sent over a

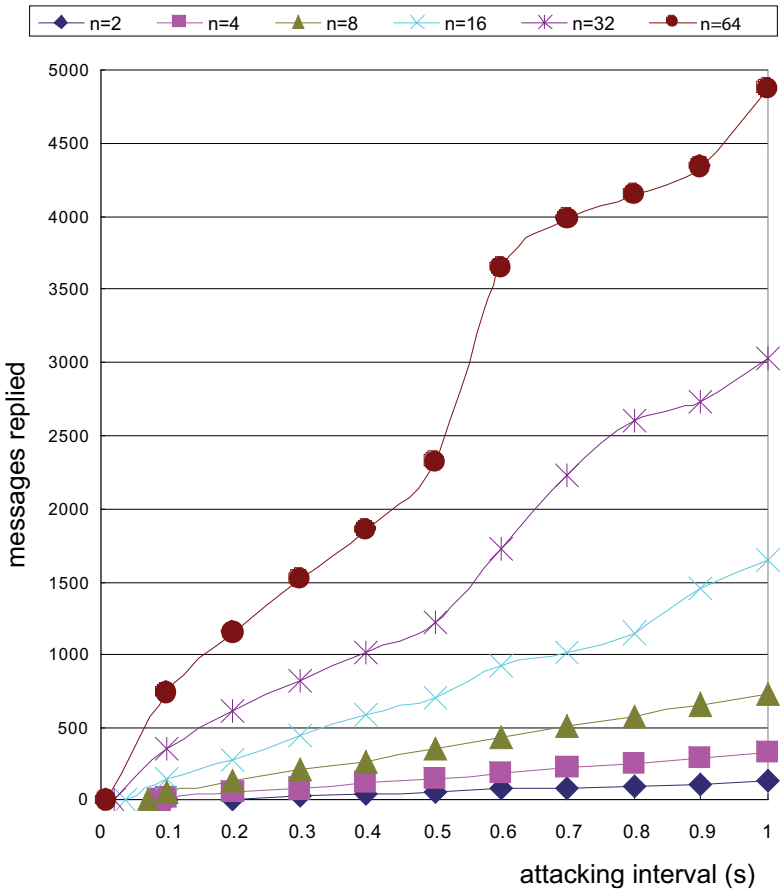

Figure 7.7: Processing performance of the local proxy for different processing queues n and varying attacking intervals

period of 140 seconds. These messages contain non-resolvable domain names with 10 ms delay between each message. We measure the number of messages the proxy can process during this test. In the optimum case, all 10.000 messages are be handled by the proxy. In case of an attack however, the proxy is not able to process all messages within time without dropping some.

Figures 7.8 and 7.9 show the number of resolved messages over time at the SIP proxy ($n = 2, 4, 16, 32$ (Figure 7.8), $64, 128, 256$ (Figure 7.9)) without a DNS cache, and also with the cache applied ($n = 2$ only, Figure 7.9). Every test case is run for about 140 seconds.

Looking at Figures 7.8 and 7.9, we can see that overall performance depends on the number of available processes for the uncached setup; with $n = 256$ showing the best results with about 4300 of the 10.000 messages processed. This value goes down to 50 successfully processed messages with

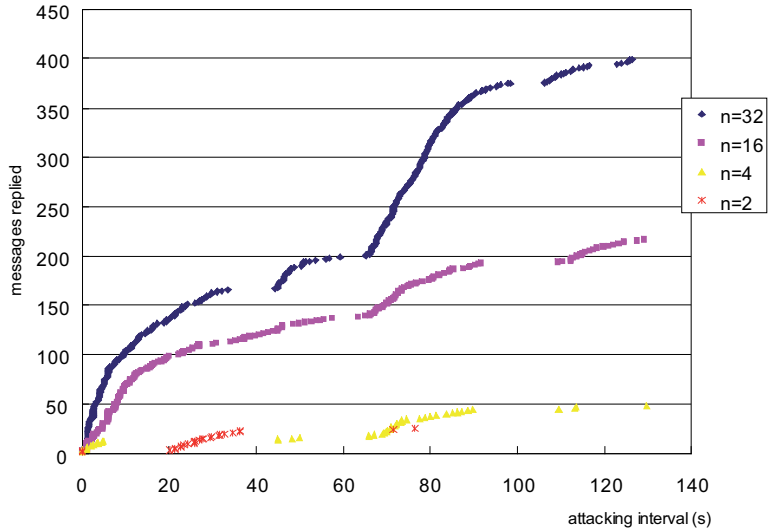

Figure 7.8: Message processing capabilities with different parallel processing queues ($n = 2, 4, 16, 32$)

$n = 2$. Hence, even with 256 processes, the performance of the proxy is still very limited in case of an attack. After deploying our cache solution the proxy answers nearly all 10,000 requests. We further observe that the result is independent of the number of activated processing queues n.

Furthermore, we can also see from the picture that without deploying the cache, the SIP server will stop working for several seconds during the test run, e.g. at $90s < t < 110s$ with $n = 16$. During this time the proxy is blocked completely, as all 16 queues are waiting for a response from the DNS server. When the cache is used, the SIP server is still functioning and can continue serving running sessions as well as new sessions destined to cached destination.

Cache Replacement Policies Evaluation

The efficiency of a cache is usually measured by its hit rate. A high hit rate indicates that the cache contains a high percentage of the entries the SIP server is trying to resolve. The hit rate is mainly influenced by the number of entries in the cache. The higher this number, the lower the probability of a cache miss and the higher the hit rate. However, the size of the cache will have to be limited. Otherwise, an attacker might deplete the system's memory by issuing a large number of DNS requests and hence continuously increase the size of the cache. Once the cache is filled some existing cache

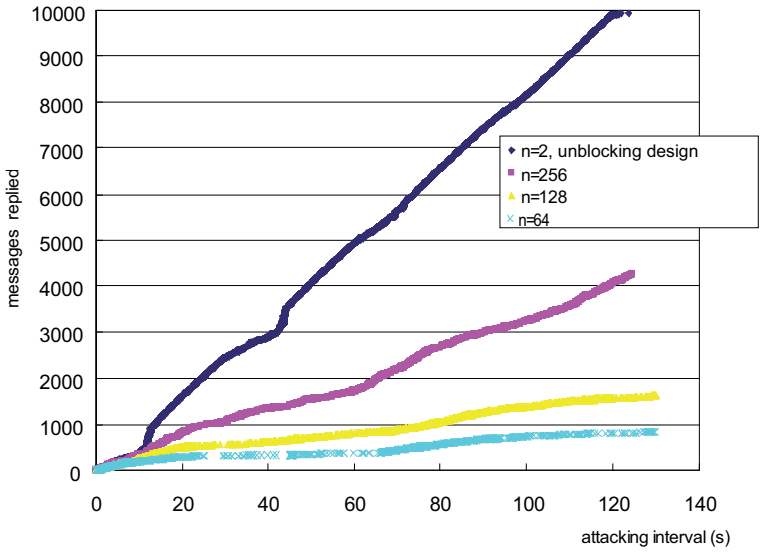

Figure 7.9: Message processing capabilities with different parallel processing queues ($n = 64, 128, 256$), and the DNS cache applied ($n = 2$).

entry will have to be deleted whenever a new entry is to be added.

Within our solution we cache only successful requests. If we choose to cache also unresolvable names, this might lead to a direct DoS situation on the cache with the described attack, as the cache will endlessly be flooded by unresolvable entries.

We consider four cache update policies; First-in, First-out (FIFO), Least Recently Used (LRU), Least Frequently Used (LFU) are well-known cache replacement strategies for paging and web scenarios. Considering that the time cost of looking up different domain names may be different, we consider also a Time Cost (TC) strategy (see Table 7.7).

Table 7.7: Cache replacement strategies and their operating key

Name	Primary Key
FIFO	Entry Time of Object in Cache
LRU	Time Since Last Access
LFU	Frequency of Access
TC	Request Time Cost

Generally, all replacement strategies are applied on a queue of cache en-

tries. The newest record is inserted into the head of the cache queue, while entries are deleted from the tail of the cache queue when the size of the cache storage is exceeded.

- For the FIFO policy, except for newest and oldest entities, all records are moved towards the tail by one when a new record enters the queue.
- LRU policy is similar to FIFO, with the difference that whenever one record within the cache is accessed, it is moved directly to the head of the queue.
- With LFU policy the DNS records are arranged by the frequency of their usage of DNS records. The higher the frequency, the closer are the entries located toward the head of the queue.
- With TC policy the queue is ordered by the time cost of the DNS lookup time of an entry. The goal is to keep entry with a higher lookup time available in the cache. The higher the lookup time cost of an entry, the closer it is to the head of the queue.

We repeat the experiment from Section 5.4.5 with these four caching strategies twice, one time with $n = 4$ parallel processing queues at the proxy and then again with $n = 32$. With a test run we observe that within our experiment we need $e = 270$ entries in our DNS cache to hold all domain names of the 100 contacted SIP proxies. The reason for this higher number is that sometimes root name servers have to be contacted first before the actual proxy name can be resolved. (e.g. ns2.mydyndns.org will be contacted automatically before looking up iptel.org). Cache replacement strategies can only be tested if the number of possible entries is lower than the total number looked up in the experiment.

While 270 entries for our test bed will require only a small memory overhead, in a real-life scenario the size have to be considerably higher.

To compare the effectiveness of the replacement strategies we arbitrarily set the entry number of our cache (e) to 80 which is reasonably lower than the possible 270 entries. The result can be seen in Figures 7.10 ($n = 4$) and 7.11 ($n = 32$).

For $n = 4$ we can see clearly that DNS caching with any caching strategy yields better performance than without DNS caching. The improvements vary depending on the attack interval and the used replacement policy, with Least Frequently Used (LFU) algorithm giving best results, which confirms to other performance tests with different setups [156]. The figure shows for LFU from 17% successful responses out of 5000 (at $i = 0.1$ s) up to 61% successful

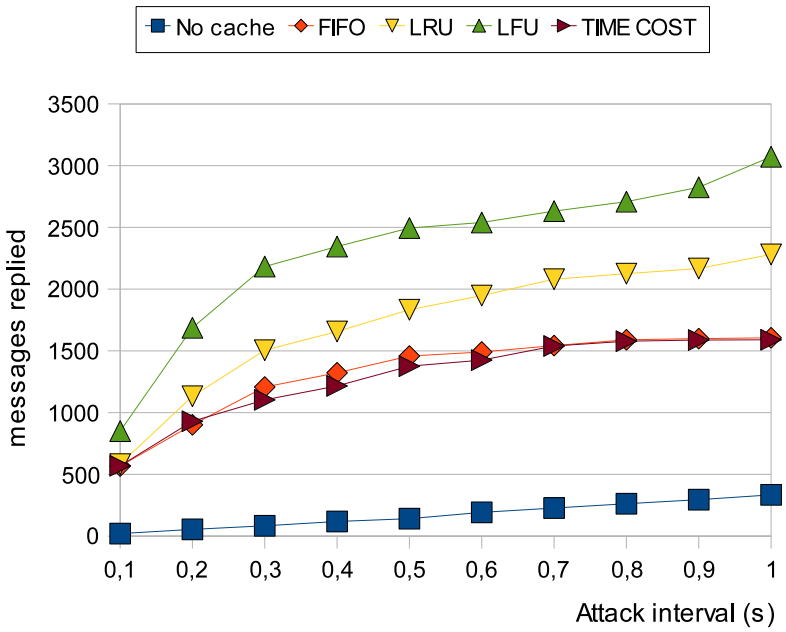

Figure 7.10: Performance of SIP proxies equipped with different cache replacement policies under attack ($n = 4$)

responses ($i = 1$ s). In comparison, figures for the uncached experiment are from 0.4% ($i = 0.1$ s) up to 6% ($i = 1$ s).

Considering $n = 32$, we can see a different result. Especially from attacking interval $i > 0.5$ s, the performance of all four algorithms are generally equal. This seems to be due to the increased processing capabilities of the proxy with $n = 32$, as also in the uncached experiment we see a clear increase in successful resolve requests. In this case, the attack simply does not consist of enough messages to slow down the proxy. However, with increased attack speed again LFU shows best results. We measure 44% successful responses ($i = 0.1$ s) up to nearly 100% successful responses ($i = 1$ s), in comparison to 6% ($i = 0.1$ s) / 60% ($i = 1$ s) for the uncached experiment.

Furthermore, we further increase the attacking speed to 0.02s, which totally blocks the proxy without cache (0% successful responses). In contrast, with the assistance of the LFU cache the proxy still manages to process 1936 (38%) messages.

In the last experiment, we survey the relationship of caching performance in relation to caching entries e. As mentioned, the number of cache entries is limited while the amount of the domain names in the real world is almost unlimited so that it an attempt to accommodate all possible domain names

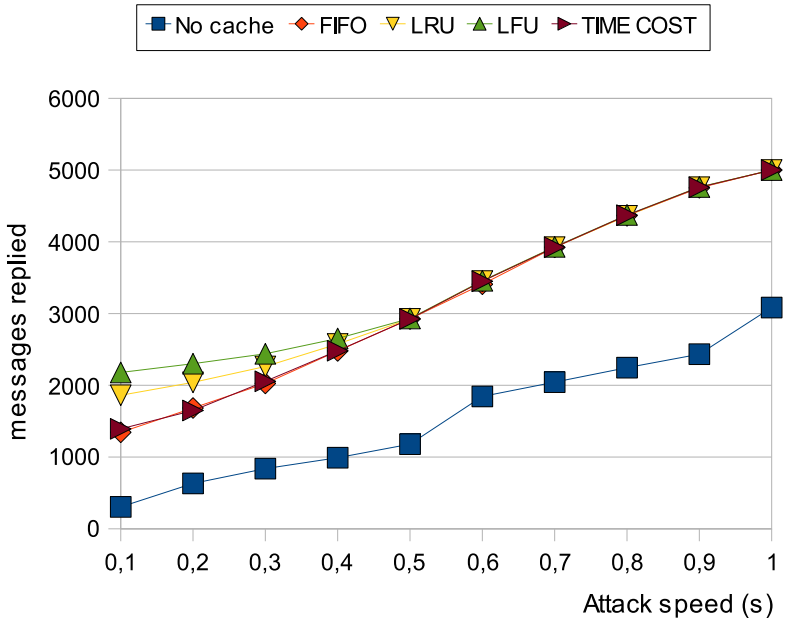

Figure 7.11: Performance of SIP proxies equipped with different cache replacement policies under attack ($n = 32$)

in the world will lead to a out-of-memory lockdown. To investigate how the number of cache entries affects the performance of the cache, we set $n = 4$ and the attacking interval $i = 0.5$ s, and evaluate with different numbers of cache entries based on the test bed. The result is shown in Figure 7.12.

From the figure, we can see that even if there are only 5 entries in the cache (minimum value measured in the experiment), the proxy still processes more messages than without a cache. The more cache entries, the better the performance of the proxy increases. When the number of cache entries is less than 150, the number of replied messages grows sharply with the increasing of cache entries while the growth becomes negligible when the number of cache entries is among 150 and 270. Finally, the curve reaches a steady state when the cache owns more than 270 entries, which matches the 270 different domain names involved in the test, Furthermore, we also increase cache entries to 500, 600 until 2000 and see that the result do not vary any more.

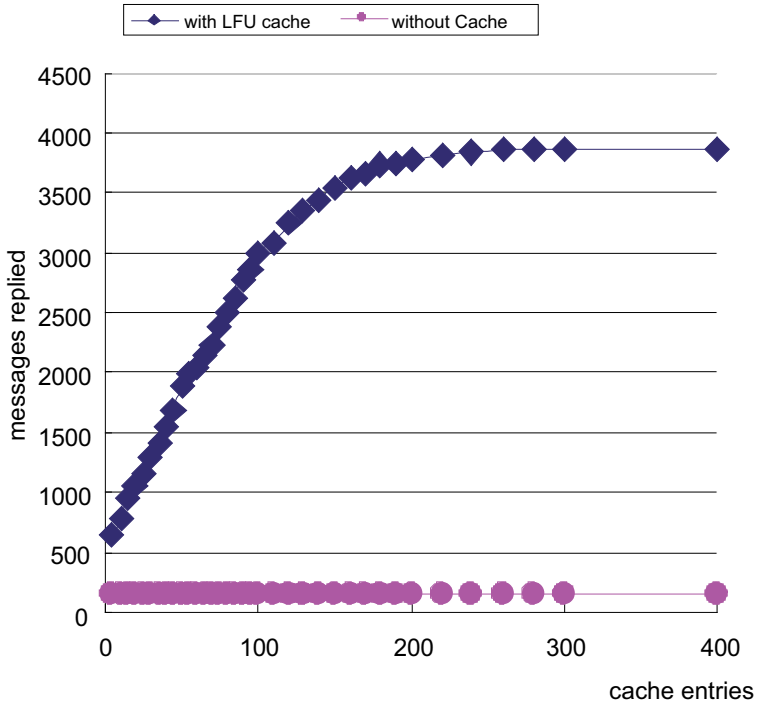

Figure 7.12: Proxy performance with different number of cache entries under attack

7.5 Performance Optimisation

Tests with the prototype, especially in Sections 7.1 and 7.3.2 show that performance of VoIP Defender degrades under certain conditions. We therefore evaluate the cause of this bottleneck and propose two methods how to overcome some of the limitations. One method is targeted especially at increasing performance of the pinholing module, while the other one is targeted at increasing performance in general.

7.5.1 Reducing the Number of Rule Updates in the Pinholing Module

The low performance of the pinholing module (see Section 7.3.2) is caused by the use of iptables as the firewall. Iptables shows limited performance mainly due to its costly memory copy operation. As already described, VoIP Defender sends a rule update request in real time, i.e. as soon as the pinhole mechanism decides that a rule update (rule insert / delete) is necessary.

However, for each new rule that is modified at the iptables firewall, the whole current firewall rule set has to be copied from kernel-space to user-space, where the rule set is updated. Then, the new rule set is copied back to kernel-space where it is installed in the actual iptables firewall to become effective. Kernel-space to user-space copy operations in Linux are a costly operation. Hence, such a process would be acceptable for static rule entries, however it is not feasible in our case with dynamic rule updates in real time.

We therefore propose another solution: Instead of updating rules immediately after they are generated by VoIP Defender, they are accumulated at VoIP Defender for a short time, so that a collection of rule update requests can be transferred to the firewall in one control message. These update requests are scheduled periodically, e.g. every second. As a consequence, the iptables firewall can avoid multiple costly memory copy operations, as it can update multiple rule set entries in just one memory copy operation.

This method however has an effect on regular users as the pinholes for regular users are not established immediately after the user's initial request, but only after a scheduled rule update push occurred. However, this effect only affects the signalling path and in effect might cause both a short and acceptable delay at the sender side before the caller hears the waiting tone.

We therefore modify the VoIP Defender firewall controller to accumulate individual rules and send out combined update control messages in 1 second intervals. New tests are conducted in the same way as introduced in Section 7.3. The performance increase with the same scenario is depicted in Figure 7.13. As can be seen in the figure, the update process is close to the optimum case – the best-case scenario actually matches the optimum case. Even the worst test run is only slightly behind schedule: While 10000 messages are generated within 20 s, all rules are installed at VoIP Defender after just about 21 s, in comparison to the nearly 200 s delay in the original setup.

As the optimised version works reasonably well, we run another attack with 50000 attack messages. The results are shown in Figure 7.14.

With this setup, we can see that iptables can handle up to around 18000 rules in real-time. If further rules are added, latency increases, resulting now in a rule installation delay of 68 s in the worst case. Clearly this test shows that iptables is not the perfect tool to be used as the firewall component, even after applying optimisation strategies. For comparison we summarise the rule-adding speed of the three tests in Table 7.8. This table shows iptables' rule-adding capabilities at the beginning of each test (without any previous rules established at iptables) and again at the end of each test. The figures are calculated from the lowest performing test run to indicate a worst-case scenario.

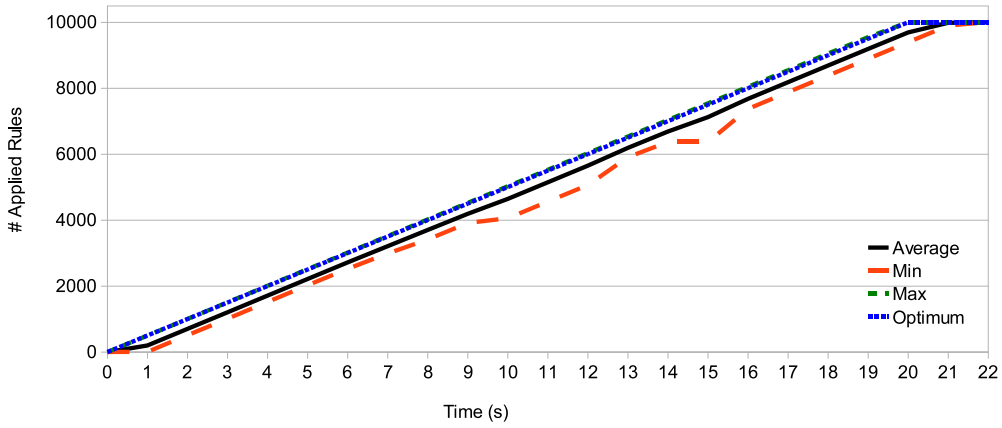

Figure 7.13: Time to install 10000 rules at the pinholing firewall (optimised version)

Table 7.8: Worst case rule adding capacities

	initial speed	final speed
real-time rule addition, 10000 rules	191 r/s	28 r/s
1 s delay addition, 10000 rules	500 r/s	433 r/s
1 s delay addition, 50000 rules	499 r/s	184 r/s

While we have shown that the mechanism works as expected, it is evident that iptables cannot cope with the generated traffic load. To circumvent this, one option could be to replace iptables with another firewall solution with better performance (e.g., software iptables replacements like ipset [157] or nf-HIPAC [158] or even a hardware-based solution). On the other hand, other optimisation options exist that could work even with an iptables based solution.

One option would be to create the pinhole not after the first request, but after the first re-transmission message is encountered. In this case, the amount of generated firewall rules would be significantly reduced; for the target case of spoofed message attack prevention it is very likely that no firewall rules would be generated at all. On the other hand, regular users would encounter a minimal delay in message processing. Given an arbitrary UA sending an INVITE to such a protected proxy, its first re-transmission message would be generated after $T1$ s. After this message, the pinhole would be generated, so that the second re-transmission message after $2 * T1$

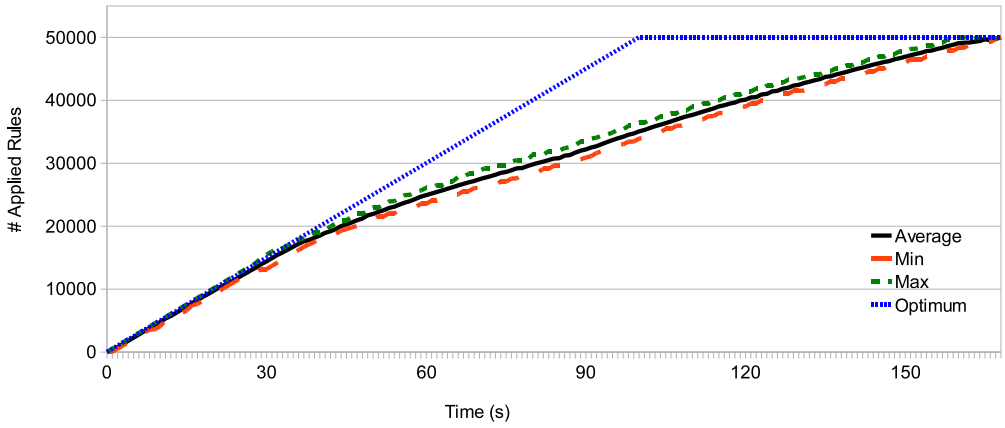

Figure 7.14: Time to install 50000 rules at the pinholing firewall (optimised version)

s could reach the proxy unhindered by the firewall. Hence, given a responsive proxy the user would experience a delay usually shorter than 2 s, before its request would be processed regularly. Such a delay is not uncommon for a voice scenario, e.g. in mobile call establishments, and thus would likely not irritate the user.

7.5.2 Generally Reducing the Number of Generated Firewall Rules

The used iptable firewall has an impact on performance in the VoIP Defender setup. Changing iptables against another, higher performing firewalls might alleviate the impact. However, as there are no endless resources, every solution will eventually come to its limits. We therefore examine a more general solution, which might not only be beneficial for our VoIP Defender setup, but also for other IDS solutions in general.

General Problem Space

In any IDS setup, there is the threat for a self-inflicting DoS. Given a high load of malicious traffic to the service, the IDS will continuously add new rules to the firewall to prevent this traffic from reaching the service. As the size of the firewall ruleset increases, the performance of the firewall will decrease, due to two factors: Searching for matching rules becomes more complex as the firewall ruleset grows. And, especially for SIP Application Layer Gateways (ALG), each passing SIP message needs to be parsed before

a decision can be made. As shown in Section 7.1, with VoIP Defender, the throughput rate of a firewall decreases with an increasing number of rules. Given a constant traffic flooding rate of about 170 Mbit/s, the rate drops to 130Mbit/s after the introduction of 100 IP-layer firewall rules, and drops further to 70 MBit/s after the insertion of SIP ALG rules. Other authors witness similar decrease in performance of protection systems [159] [56].

Hence, security solutions are faced by the following dilemma, which holds true especially in high-traffic scenarios: *If no flooding protection is applied, the service's performance will degrade due to the attack; however, applying IDS protection might also degrade service's performance due to limited firewall throughput rates.*

There a several possibilities to address this conflict in high-flooding scenarios:

1. Define more *intelligent protection algorithms* for the IDS. Ideally, the protection algorithm should generate as few firewall rules as possible but still provide effective protection. This would increase the complexity of the detection scheme and hence introduce a new performance penalty. Additionally, algorithm optimisations have to be developed separately for each new threat.

2. Design and implement a *high-performance firewall*. Different designs for firewalls exist [160] to increase firewall performance. However, adding additional features like SIP ALG processing, reduces performance again.

3. *Limit the number of rules* at the firewall, independent of the used IDS detection algorithm. This is the scope of this work in the context of flooding attacks. By applying a maximum size limit of the ruleset, a certain operating level of the firewall is guaranteed.

Each of the solutions have certain advantages and drawbacks. Most importantly, there is likely no ultimate solution that allows *optimum security* together with *maximum performance*. Hence, a solution to this problem is likely a trade-off between these two factors.

We have already proposed a possibility of an intelligent protection algorithm in the previous Section 7.5.1. Now, we evaluate the possibility of limiting the number of rules at the firewall.

Researchers have already proposed mechanism to reduce the overall ruleset size. Gupta [161] introduces forward or backward redundant rules that can be eliminated from the ruleset. A backward redundant rules is defined as an existing rule r appearing earlier than r', which is a subset of r. On

the other hand, if there exists a rule r appearing after r', which is a subset of r, it is forward redundant. Liu [162] proposes further methods to eliminate backward and forward redundant rules. Similar to our work, Yoon et al. [163] propose an aggressive reduction algorithm to find a group of rules and replace it with a smaller new group. However, their work does not to allow the space of the generated ruleset to differ from the original one, hence it is not guaranteed that the generated ruleset will be significantly smaller. All of these propositions work only at the network level and would also need further enhancements to work in SIP ALGs.

Firewall Ruleset Optimiser

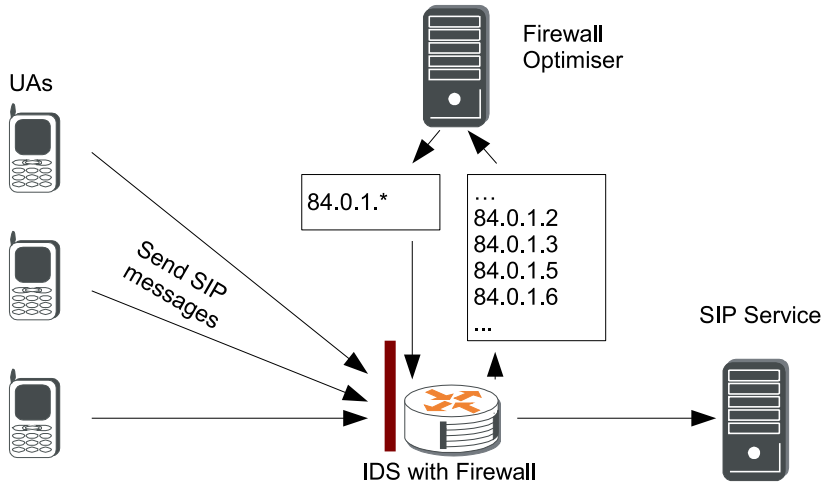

Figure 7.15: An optimiser re-constructs the firewall ruleset generated by an IDS to enhance throughput of the IDS

Algorithm Overview To reduce the size of the ruleset at the SIP firewall we define σ_{max} as an upper maximum bound for the number of rules the SIP firewall will accept. Its value depends on the performance of the used firewall and on the traffic scenario of the VoIP network. A higher performance firewall can tolerate a higher value for σ_{max}. The value can, e.g. determined by stress tests of the firewall. The value should be set in such a way that whenever $|Ruleset| \leq \sigma_{max}$ (where $|Ruleset|$ indicates the size of the firewall's rules), the firewall will yield acceptable performance for the defined setup and scenario.

We introduce a *ruleset optimiser* which constantly translates the input ruleset R_i, generated by an IDS, to the output ruleset R_o, which strictly

satisfies the requirement $|Ruleset| \leq \sigma_{max}$. R_o will then be applied at the firewall (see Figure 7.15).

By limiting the maximum ruleset size, the firewall's effectiveness will likely degrade by a certain degree e.g., rules might not be deployed at the firewall because the maximum size has been reached. We accept this lesser accuracy for flooding scenarios as an unwanted, but necessary side condition, as this at least provides partial service protection, in comparison to no protection at all. Also, for DoS attacks, it is acceptable to let some DoS packets pass through, as they will only increase the load at the proxy, but not degrade its performance, if most other packets will be blocked.

The basic idea of our algorithm depends on the three steps *ruleset merging*, *rule dropping*, and *accuracy calculation for optimisation*, as seen in Figure 7.16.

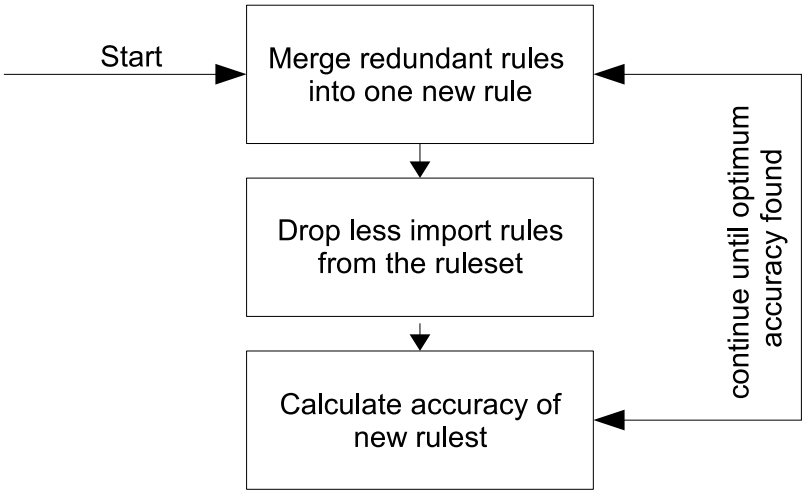

Figure 7.16: Overview of the ruleset optimiser

Firstly, we assume that most rules generated by the IDS will be single-mapped. For example, considering IP addresses, there will be IPv4 address field ruleset conditions for entries "81.1.0.1", "94.7.56.1", and "81.1.0.79". Looking at SIP vendor names there might be SIP user-agent header field conditions like "Cisco ATA 186 v2.16.1 ata18x", "Twinkle/1.1", and "Cisco ATA 186 v2.16.2 ata18x". We merge several single-mapped rules into one multi-mapped rule using a redundancy metric.

Secondly, if after this step $|R_o| > \sigma_{max}$, additional rules will be dropped until $|R_o| \leq \sigma_{max}$. We define a metric to select more relevant rules for merging and less relevant ones for dropping.

Using both ruleset merging and dropping, the accuracy of the ruleset changes during the transition. For example, by combining multiple single-

mapped rules into one multi-mapped rule, additional targets might be included in the matching ruleset. Contrary, by dropping rules from the ruleset, targets are excluded from the matching ruleset. Hence, we finally apply a metric to calculate a minimum change in accuracy from R_i to R_o.

To meet both performance and security requirements, the optimiser has to ensure, that

1. $|R_o| \leq \sigma_{max}$ during operation, and
2. the accuracy change through the generation of R_o from R_i is minimal.

Algorithm Details

Redundancy Metric Within the merging step, multiple single-mapped rules will be merged into one multi-mapped rule. The actual process depends on the scope of the firewall rule, and has to be defined for each examined field individually. We demonstrate this here with two examples, which can easily be extended for other fields of interest.

IPv4 Address Field The most common use in firewalls is the matching of IP address fields. We apply merging at the subnet level i.e., several single-mapped address rules are mapped into one multi-mapped rule covering one whole subnet. Ideally, if there are 254 single-mapped address rules covering e.g., the address range from 84.1.0.1 to 84.1.0.254 in R_i, we can replace these 254 rules with one multi-mapped rule 84.1.0.* in R_o.

Admittedly, this example covers an ideal situation: there are exactly 254 IPv4 addresses in the same subnet in R_i, so one can simply use the subnet address in R_o instead of 254 individual IPv4 addresses. However, merging with less than those 254 rules would change the accuracy of the resulting R_o. We introduce the variable χ to determine the threshold after which single-mapped rules are replaced by a multi-mapped one. Rules in R_i can be merged into a multi-mapped rule only if the number of mergeable rules is greater than χ. For example, if there are three address rules "84.1.0.1", "84.1.0.20" and "84.1.0.113" from the same subnet, and given that $\chi = 3$, they will be merged into "84.1.0.*". However, if $\chi = 4$, they will not be merged. The value of χ will be computed in the optimisation step.

SIP User Agent Header Field Many SIP header-fields are free-from text fields, e.g. the user agent string that denotes the generator of the SIP message. Its usage in SIP firewall would be highly beneficial to filter ill-configured user agents. Entries vary considerably due to the diversity of

vendors and versions. We have observed a common pattern how the string is generated, consisting of three parts, UA name, sub-name and version usually in left to right order, commonly separated by whitespace. Based on this observation, single-mapped UA string rules can be merged into one multi-mapped one.

As an example, we can separate a UA string into two parts by the last non-alphabet, non-numeric character. We call the left part *name part* and the right part *version part*. Firstly, we subgroup the rules which share the same name part. For example, "Zultys ZIP 2 3.43" and "Zultys ZIP 2 3.47" can be subgrouped because they share the same name part "Zultys ZIP 2 3.", however "sipsak 0.8.11" and "sipura/SPA" can not be subgrouped as their name parts are different. Again, χ indicates how many different UA strings can be merged into a new one. Only if more than χ rules can be found sharing the same name part, a new merged rule is created for it.

Significance Metric We define a significance value F_{rq} as the number of times a rule is matched within the IDS. For instance, given a rule that allows incoming traffic from IP address "10.8.0.1" which is matched at the IDS by incoming traffic 10 times since the last optimisation, we set $F_{rq} = 10$. Using a Most Frequently Used (MFU) policy, we are able to order the rules in R_i by significance. This ensures that at least the offenders which the most overhead traffic will be denied access to the network.

Accuracy Changes We introduce the term *space* to indicate how many entities are mapped to possible targets. For example, the space of a single-mapped IPv4 rule is always 1, while it is higher for multi-mapped rules (i.e., it is 254 for the above defined subnet rules). For UA strings we look at the size of the UA string. We can then define $S_{ruleset}$ as the space of all rules in a ruleset combined. Ideally, S_{R_i} should be close to S_{R_o} as this would indicate an accurate conversion.

Now we can calculate *false positives* and *false negatives* within the two rule sets. We define F_p as the increase of space through rule merging in R_o:

$$F_p = \sum i, \forall i \in S_{R_o}, i \notin S_{R_i} \tag{7.1}$$

Similarly, we can define F_n as the number of false negatives:

$$F_n = \sum i, \forall i \in S_{R_i}, i \notin S_{R_o} \tag{7.2}$$

Both merging and dropping will decrease accuracy in the target ruleset R_o, either by the introduction of false-positives (caused by merging) or false-

negatives (caused by dropping). Hence, with the conversion from R_i to R_o, the target space will change, causing R_o to become less reliable.

Table 7.9: Introduced variables

R_i	original ruleset consisting of a list of single-mapped rules
R_o	output ruleset after optimising, consisting of a list of single- and multi-mapped rules
σ_{max}	size constraint of R_o
F_p	false positive index introduced by optimisation
F_n	false negative index introduced by optimisation
χ	a threshold number to indicate the minimum number of single-mapped rules needed before they can be merged into one multi-mapped rule
η	a weighting factor to indicate the importance of F_p compared to F_n

For example, given R_i with 600 rules and setting σ_{max} to 20 and considering only single-mapped rules, only 20 rules with the highest F_{rq} can be considered for R, and information contained in the 580 lower significant remaining rules would be lost, so $S_{R_i} = 600$ and $S_{R_o} = 20$, and the false negative index $F_n = S_{R_i} - S_{R_o} = 580$.

The change in accuracy can be denoted by a (preliminary) *inaccuracy index*, as the sum of F_p and F_n. The higher this index gets, the more information is lost during the conversion from S_{R_i} to S_{R_o}.

However, F_p and F_n will generally not be considered equally in all situations. For example, if in a DoS scenario it would be acceptable to let a certain degree of malicious traffic pass, but regular users should rather not be affected, F_p should be rather low, while higher values for F_n are acceptable for a firewall in blacklisting mode. Other scenarios might require a different setup. To reflect this, we introduce a weighting factor η and define

$$\mu = F_p + \eta F_n \tag{7.3}$$

where μ is the final inaccuracy index, which should be as low as possible. All defined variables are listed in Table 7.9.

Formal Specification The formal specification of the algorithm is listed as Algorithm 1. Given the input values R_i, σ_{max} and χ the algorithm calculates the optimised ruleset R_o using a temporary ruleset R_{tmp}. R_i will be

Input: R_i, σ_{max}, χ
Output: R_o
1 $R_o = \phi$;
2 $R_{tmp} = \phi$;
3 SubGroup(R_i): $R_i = R_{i1} \cup R_{i2} \cup R_{i3} \cup ... \cup R_{in}$;
4 **foreach** $j (j \in \{1, ..., n\})$ **do**
5 | **if** $|R_{ij}| \geq \chi$ **then**
6 | | $r_{multi} = Merge(R_{ij})$;
7 | | $R_{tmp} = R_{tmp} \cup r_{multi}$;
8 | **end**
9 | **else**
10 | | $R_{tmp} = R_{tmp} \cup R_{ij}$;
11 | **end**
12 **end**
13 $SortByFrqAcend(R_{tmp})$;
14 **while** $|R_o| \leq \sigma_{max}$ **do**
15 | $R_o = R_o \cup Pop(R_{tmp})$;
16 **end**

Algorithm 1: Converting R_i to R_o

separated into subsets with each subset containing its own mergeable rules. Then, the size of each subset is calculated, and if the size is greater or equal than χ, all rules will be merged into a multi-mapped rule, which in turn is inserted into R_{tmp}. Otherwise, the rules in the subnet will not be merged, and will be inserted into R_{tmp} independently. F_{rq} of a multi-mapped rule is calculated as the sum of each single-mapped rule's F_{rq}. Finally, rules in the R_{tmp} are sorted according to their F_{rq} and the top σ_{max} rules with highest F_{rq} are selected to generate R_o. The methods "SubGroup" and "Merge" depend on the rule type (e.g. IPv4 address rule, UA header string).

We call the procedure in code lines 3 to 12 "merging" and the procedure in code lines 13 to 15 "dropping".

Finally, we formalise the optimisation model as follows.

$$min : \mu = F_p + \eta F_n \tag{7.4}$$

$$s.t. : R_o = f(R_i, \sigma_{max}, \chi) \tag{7.5}$$

$$|R_o| \leq \sigma_{max} \tag{7.6}$$

Here, (7.4) is the minimisation objective function to achieve, indicating the goal to find an optimal result with the inaccuracy index μ minimised. (7.5) and (7.6) are constraint functions, with (7.5) showing how to reach the

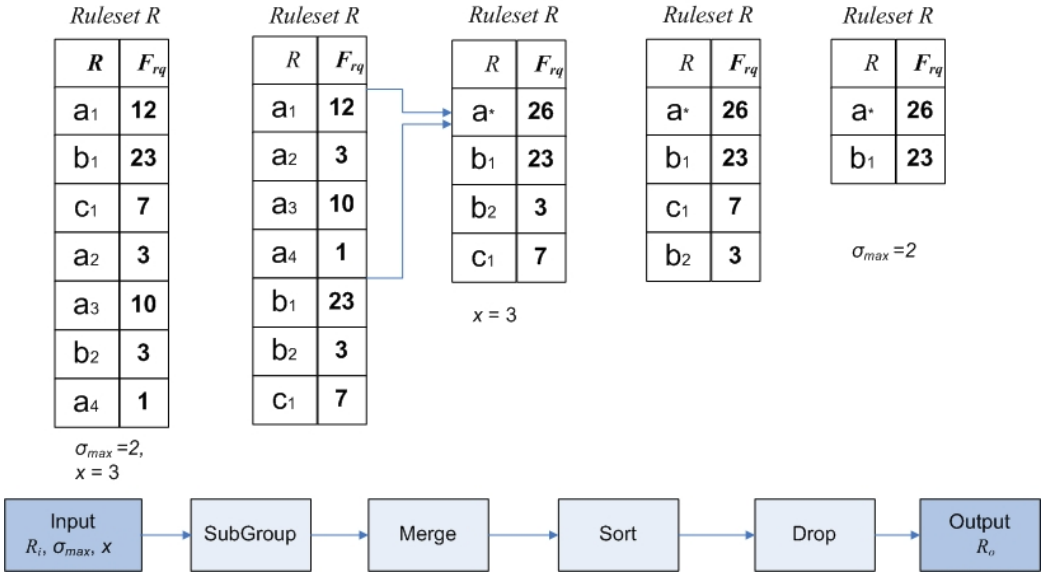

Figure 7.17: Example process of the optimisation algorithm - here, the four similar rules $a_1 - a_4$ are merged into one new rule and rules with lesser frequency are dropped.

output ruleset R_o using algorithm 1 as f, and constraint (7.6) indicating that the size of output R_o must be less than σ_{max}.

Example Run The general optimisation process using a concrete example can be seen in Figure 7.17. Here, different rules (a_x, b_x, c_x) with their frequency are shown. First, similar rules are subgrouped, so that, e.g. all a_x rules are put together. Then, similar rules $a_1 - a_4$ are merged into one new rule, as there are 4 individual a-rules here, passing the threshold value $\chi = 3$. However, the b-rules are not merged, as there are only two of them. The four new rules are sorted by their frequency, and finally the two rules with the lowest frequency are dropped to satisfy $\sigma_{max} = 2$.

Application

Testing with Real-Life Traffic For testing and verification, we apply the ruleset optimiser to test-traffic from different real-life SIP VoIP providers. Traffic is recorded for several hours, resulting altogether in about four GByte of tracefile data. As a simple test, we generate for each traffic sample an input ruleset R_i that matches every single packet.

We generate 6 different samples from that traffic to be used as R_i: ruleset A - C containing each 450, 600, and 1034 IPv4 address rules, respectively.

Furthermore, rule sets D - F with 51, 86, 107 SIP UA rules. We use a prototype optimiser written in Perl to input these 6 samples as R_i. The program optimises the ruleset and generates R_o, including the calculation of F_p and F_n. The optimiser runs on a 1 Ghz Linux machine with 512 MByte RAM.

For the tests, we arbitrarily set $\sigma_{max} = 20$ and $\eta = 20$. The result is shown in Figure 7.18. The inaccuracy index μ is calculated for each values for χ ranging from 1 to 254. Here, we can clearly see the optimum value for χ as a local minimum. For example, for ruleset A, the optimum value for χ is around 38. Setting the wrong value for χ, either by setting it to low or to high causes the accuracy to degrade. Moreover, the optimum value for χ changes for different rule sets.

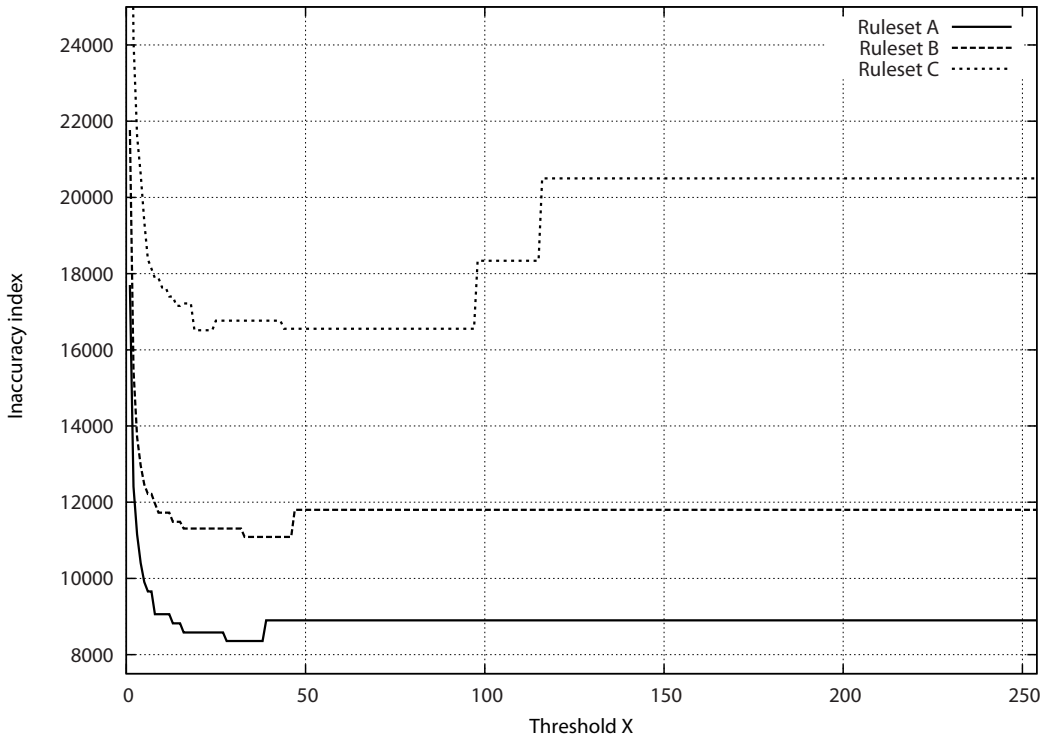

Figure 7.18: The inaccuracy index of outcome ruleset varies according to threshold χ for ruleset A and B, when $\eta = 20, \sigma_{max} = 20$

To calculate the false positive F_p for the SIP UA rules, we arbitrarily assume that each multi-mapped rule contains 50 entries at most. For instance, if a group of 5 single-mapped UA rules are merged into one multi-mapped rule, the false positive is 50 - 5 = 45. In this way, we can calculate F_p for the given R_o. Figure 7.19 shows our test result with ruleset D, E and F. The

result is similar to the previous one.

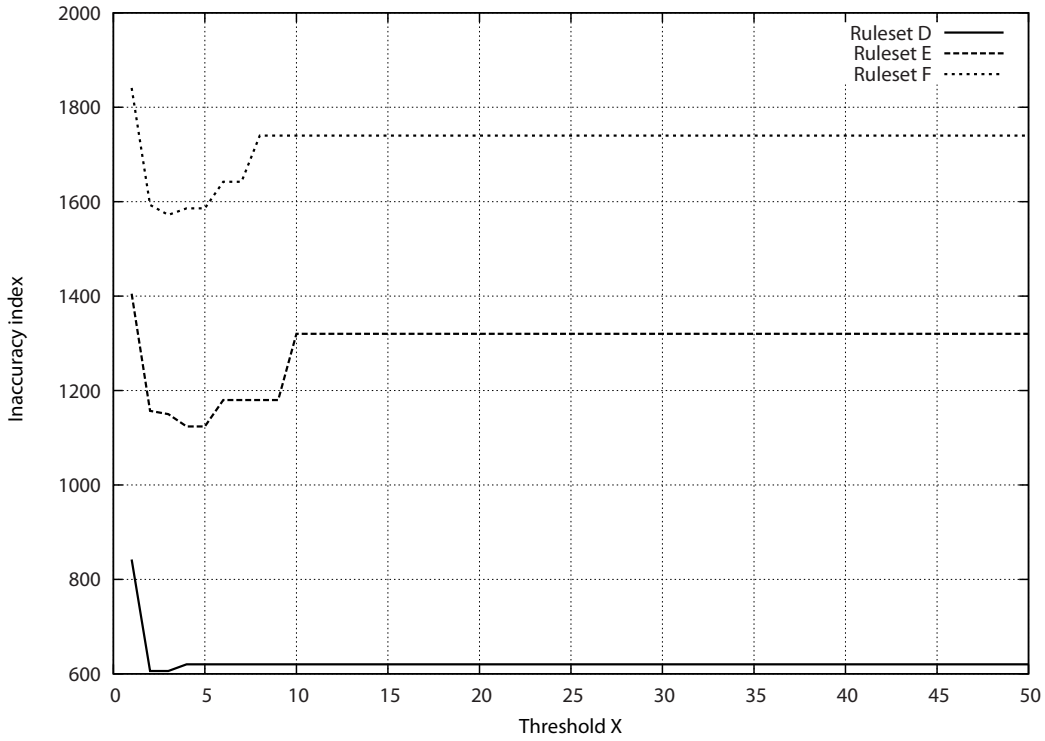

Figure 7.19: The inaccuracy index of outcome ruleset varies according to threshold χ for ruleset C, when $\eta = 20, \sigma_{max} = 20$

Optimisation

To find the optimum value for χ, we loop over each value of χ to find out the local minimum. This is a very time-consuming process and finding the right value for χ is an NP-hard problem [164]. We therefore apply a high efficient heuristic algorithm to calculate the best value for χ in reasonable time. Here, we employ Golden Section Search (GSS) [165] to solve this problem. GSS is a simple and efficient 1-dimensional optimisation method which does not require derivatives. It can deal with our unimodal objective by rapidly narrowing our search interval and still finding the optimum result.

We run several calculations of the optimisation process both with the full loop search and with GSS optimisation applied. Visible in Table 7.10, we can see that GSS can improve calculation time by around 90% with our script prototype. While reducing calculation time, the accuracy of the resulting μ is not influenced by the application of GSS.

Table 7.10: Calculation time comparison of the algorithms, T2 with GSS applied

Ruleset	T1 (s)	T2 (s)	Solution with GSS
A	51.51	3.31	$\chi_{optimal} = 38$
B	95.69	6.29	$\chi_{optimal} = 46$
C	229.04	17.71	$\chi_{optimal} = 97$
D	0.23	0.04	$\chi_{optimal} = 3$
E	0.49	0.13	$\chi_{optimal} = 5$
F	0.76	0.18	$\chi_{optimal} = 3$

Integration into VoIP Defender

The firewall ruleset optimiser is not yet integrated into VoIP Defender. However, the implementation and integration into VoIP Defender should not be difficult to achieve, and would be a further step in validating the correctness of the algorithm. The reason is is not yet integrated is solely the lack of resources.

Chapter 8

Comparison with Other Approaches

DoS protection for SIP-based networks is a wide research field. More than 20 different research works have already been published to address the threats introduced in Chapter 3. In this chapter we survey different DoS protection approaches from other researchers that have been published in scientific literature and compare them to our own protection solution as specified in Chapter 5[1]. We define different criteria on which we base the comparison, and summarise how well the research community has addressed the DoS threat.

8.1 Evaluation Criteria for DoS Defence Systems

Multiple countermeasure schemes have been proposed to target SIP related DoS attacks. They are necessary as general IP-based DoS protection systems do not address dedicated SIP DoS attacks. For example, SIP messages can use different transport protocols to deliver messages to the destination (like UDP, TCP or even SCTP). General IP-based DoS flooding protection mechanisms would not be able to detect an attack flow using different transport protocols, as they cannot take the actual message payload into account. SIP message flow tampering detection systems are also only possible if they have SIP knowledge. SIP DoS countermeasure systems can thus be seen as an additional layer of security to be deployed in conjunction with general IP DoS protection mechanisms.

[1]Results of this chapter have been published in [4].

For comparative purposes we will evaluate these methods using different criteria as they target different attacks, use different countermeasure algorithms or have performance differences. We define two main criteria groups: *algorithm-related* and *framework-related* criteria. The former is related to the theoretical idea of the countermeasure solution. It covers the mathematical principle the method is based on. The latter covers the actual implementation of the theoretical algorithm including setup, architecture and performance.

8.1.1 Algorithm-related Evaluation Criteria

Algorithm Principle This is the core of the defence method, the basic mathematical principle of how the author would handle the envisaged attacks. This could be a statistical model, a data mining approach, etc. As SIP is based on a complex state machine specification, many authors use this specification as a basis for their defence solution.

Attack Classes We indicate what kind of attack class is addressed. Here we consider the three main attack groups identified in Chapter 3, especially visible in Figure 3.1: payload tampering, flow tampering, and flooding. Especially for flooding attacks, we evaluate if the method considers single-source flooding or multiple-source flooding. If other, non DoS-related attack types are addressed (e.g. SPIT detection), they will be mentioned. However, these attacks are not the focus of this comparison.

Victim While most proposals are concerned with the protection of the main servers (which would be the SIP proxy or the P-CSCF in IMS networks), some are targeted at SIP client (UA) protection.

Protocol Our focus in this work lies on specific SIP-based attacks. However, RTP also plays an important role in SIP-based networks. We mention if the discussed method also considers RTP-relevant attacks. This would be mostly the case for RTP flooding attacks.

Reaction This defines what reaction is achieved by the defence method. All algorithms are targeted for attack detection, however not all algorithms can later classify the attack traffic, which is needed for attack prevention. Other ones might not be able to prevent the attack, but could propose a method for attack mitigation, i.e. a method to sustain the attack.

Detection Strategy To detect an attack, the algorithm has to have some knowledge of the attack. Basically, there are two possible principles.

Either the attack is described (e.g. using signatures), which is called pattern-based detection or by defining some type of "normal" network traffic. Attacks are then detected by deviation from this norm, which is called anomaly-based detection.

8.1.2 Framework-related Evaluation Criteria

Setup The security mechanism has to be employed in a security framework as introduced in Section 2.4. The common case would be a Network-based IDS (NIDS). A host-based IDS (HIDS) is deployed directly by the same host as the target SIP proxy, and evaluates information from this host, e.g. application log files. Another option would be to implement the defence solution as an extension module of the target proxy.

Implementation Details about the actual implementation, if any. This includes the programming language used and the framework used.

Placement The placement of the framework within the network has an effect on what information the framework will have for evaluation, especially in NIDS-setups. A common placement would be at the ingress point of the network, so that all incoming passing traffic can be seen. Some setups are based on distributed systems, with multiple different monitoring points.

Reactive Measure This means the reactive measure against a classified attack, if a method provides one. This could be firewall control to block certain requests, for example.

Performance Test Results If an implementation exists we will present performance results as described by the authors. Noteworthy features include detection latency, i.e. the time after attack launch needed before it can be accurately detected; processing latency i.e. the delay normal users encounter because this method is deployed in the network and processing capabilities i.e. how many messages the framework can process in one second. Note that these results cannot directly be compared between different implementations – there are too many variations between test setups, test configurations and testing hardware.

Scalability Especially for high-message flooding attacks it is important that the framework can scale for higher bandwidths. Here we will provide information if the authors have considered this, e.g. by limiting the memory requirements for their solution or introducing the possibility of using multiple detection machines.

8.2 Survey of SIP DoS Countermeasure Solutions

Here we present the actual countermeasure solutions that have been proposed by various researchers. The countermeasures are presented in order of publication. Each proposal is summarised according to the aspects of the evaluation criteria introduced in the previous section and separated into an algorithm-section and a framework-section.

We base this summary on available facts from the publication and do not judge any of the proposed ideas here. This is especially important with regard to measurements because of different testing conditions. Thus neither of the given performance measurements can be used directly for performance comparisons between different ideas.

A comparison and discussion of the presented ideas will follow in a later section.

With the exception of Inacu03, the survey covers proposals that have been described in scientific publications. There are already several commercial SIP security solutions (e.g. from Borderware [62] or AcmePacket [63]) targeting the same threats. However, information on and specifications of these systems are restricted, and thus cannot be evaluated here.

8.2.1 Iancu03

Algorithm

Iancu [60] developed a DoS flooding mitigation mechanism dubbed "Pike" that rate-limits incoming traffic on a per-host basis.

This method is listed as an example for the various rate-limiting software mechanisms available as add ons for SIP servers or in commercial security solutions. The algorithm counts all incoming requests per IP address in a defined time frame. Whenever a fixed upper limit is reached, further messages from the offending IP address are not processed for a limited time.

Framework

Pike is distributed with SER. It runs as an extension module within the proxy and operates only on that proxy. This is a common DoS mitigation algorithm that is also deployed in similar forms in other security frameworks, including in commercial setups like [62].

8.2.2 Reynolds03

Algorithm

Reynolds and Ghosal [57] propose a DoS flooding counter-measure mechanism by detecting open SIP sessions using the cumulative sum method (CUSUM) [166].

Attack detection is based on the observation that the ratio of connection setup messages (INVITE) and positive replies (200 OK) should be roughly equal at any given time. Hence, when this ratio suddenly changes, this is likely to be an indication of a DoS flooding attack, as such an attack would yield a lot of open connections that are not closed immediately. This principle was first proposed to detect general TCP SYN flooding attacks [167]. To determine the actual moment when the flooding begins (and stops), the non-parametric cumulative sum is applied. CUSUM is a sequential analysis technique which is used for monitoring change detection. This method is an anomaly-based mechanism, where threshold values have to be set to raise correct alarms.

This method is targeted to protect the end-user terminals (UAs) only; the authors do not consider the main SIP proxy to be the target of an attack. Under this assumption their method can also be used for attack prevention. In case of a detected attack against a UA, traffic to the identified UA can be throttled down or temporarily disabled by the SIP proxy.

Framework

The authors suggest implementing their mechanism within a NIDS (dubbed "Application Layer Attack Sensor") placed in front of the SIP proxy of the network to be protected. For attack mitigation, the sensor has a connection to the SIP proxy, to instruct it when it should throttle down or temporarily block requests. The framework has not been implemented, however its operation has been simulated with the author's "emulation toolkit" which is not further specified.

The authors simulate different flooding attacks to multiple UAs in the SIP network. As only UAs are considered to be victims, the attack rate was set rather low, ranging from 1-200 msg/min within the simulation. All attacks could be detected with the simulation, however due to the low attack rates it took up to 8 min to detect the attack with less than one attack message per minute.

8.2.3 Wu04

Algorithm

Wu et al. [53] present a stateful data mining IDS dubbed "SCIDIVE" to detect SIP message flow tampering and DoS flooding attacks by correlating SIP and RTP network traffic events.

Both SIP and RTP traffic is monitored and individual events are generated from the monitored packets. Events are predefined characteristics that can be extracted from received messages e.g. a session tear down event (when a BYE message is intercepted), or an RTP jitter event (when two out of order packets are observed). Detection signatures can be applied for these events. For message flow tampering this would be, for example, a tear down event *preceding* the corresponding RTP stream stop event. Generally, it would be the other way around. For flooding detection this would be to detect a large number of unchallenged 401 reply events from the proxy. Thus, this is a stateful approach: to determine if an attack is under way, the previous state if the system is considered alongside the currently encountered messages.

While the authors provide some hints for proxy protection, their focus lies on UA attack detection. This mechanism does not provide mitigation features.

Framework

For cross-protocol correlation to work properly, SCIDIVE has to be deployed at a place where it can monitor both SIP and RTP traffic. As RTP is generally routed end-to-end, the IDS cannot be deployed in the SIP provider network, except if the provider forces RTP traffic to pass its network (e.g. by enforcing the usage of an RTP proxy). However, the authors do not assume this scenario. Instead, they propose to place the IDS directly at all relevant UAs. It is unclear if it is their intent to place the IDS in front of users that are likely to be the target of an attack or in front of malicious users. In the test setup the IDS is placed in front of all users i.e. also in front of malicious users. They hint at the possibility of extending the framework that would allow multiple IDS instances to communicate with each other.

The framework has been implemented as a non-specified prototype NIDS. The need for efficient state handling is mentioned for scalability reasons but not evaluated any further. A theoretical performance projection is given.

8.2.4 Geneiatakis05

Algorithm

Geneiatakis et al. [168, 29] present a signature-based solution to protect SIP network elements from message payload tampering attacks.

They suggest the employment of signature patterns (based on the SIP grammar) to distinguish well formed messages from malformed ones, similar to computer virus signature descriptions. Specifically, any SIP message which does not correctly conform to the SIP grammar is identified as malformed. Two types of signatures are defined. The first type describes a general signature structure, to be applied to any SIP message, whereas the second type defines signatures that are applied only to specific SIP messages. The signatures are created using Perl Regular Expressions [169]. Any message classified as malformed is dropped and thus will not be processed by the target entity.

Framework

The authors have outlined an implementation either in any SIP network element as a pre-filtering mechanism before incoming messages are passed to the actual SIP parser or the solution can be incorporated into a General NIDS setup. For testing purposes the authors have implemented their solution in the core of the SER SIP proxy. They performed measurements for the introduced processing overhead in various testing scenarios and the false alarm ratios. The results show that the delay introduced on the server side is about 120 μs, while no false alarms have been raised.

8.2.5 Markl05

Algorithm

Markl et al. [155] propose a signature-based message integrity checker and DoS flooding preventing mechanisms based on the Snort IDS [115].

Passing SIP messages are checked for known malicious content, e.g. SQL code injection.[130]. Similarly to Iancu03, single-source message flooding is detected by a threshold message counter, and further messages below this threshold are dropped. Additionally, signatures for general IP-based attacks not related to SIP are applied.

Framework

This method is supposed to be a lightweight complimentary prevention system to be deployed together with the previous prevention system (Geneiatakis05) [14]. The attack signatures are fed into the Snort IDS. Snort works as a network bridge and captures all passing traffic and is thus best placed at the network ingress point. It controls the network firewall through the use of the SnortSam [170] firewall controller, different firewalls can be used to block offending senders. Prelude [116] is used to gather intrusion alerts from multiple sources and present alerts to the operator. Tests have been conducted with flooding rates of 3000 msg/s. As no modification to Snort has been done, performance and scalability are dependent on Snort's abilities.

8.2.6 Chen06

Algorithm

Chen [171] proposes a SIP state machine specification to detect multiple source message flooding attacks.

The author models the four defined transaction state machines specified in the SIP RFC (INVITE and non-INVITE transaction state machine, both for the client and server part). For each transaction, the according state machine is updated, whenever a new SIP message is encountered. This idea was first introduced for TCP/IP intrusion detection [172]. For flooding detection Chen adds an error state to each state machine and defines how this error-state can be reached. An attack is indicated if the number of error states in one sampling interval surpasses a threshold. The threshold for attack detection is network dependent. This method is for detection only.

Framework

This is a theoretical concept. The author proposes to place his mechanism in an external IDS at the network ingress point.

8.2.7 Niccolini06

Niccolini et al. [137] present a multi-layered IDS to counter different types of attacks.

Algorithm 1

The first countermeasure mechanism is a SIP message integrity checker to prevent SIP message payload tampering, similar to Geneiatakis05.

The mechanism checks that all incoming SIP messages are well-formatted by ensuring that all header-field sizes are correct, for example. Non-conforming messages can be discarded.

Algorithm 2

The second countermeasure mechanism is a basic SIP dialog state machine to detect out-of band message flow tampering and DoS message flooding.

A basic SIP Dialog state machine guarantees that messages within one dialog have the correct order, e.g. that a BYE message follows after the appropriate INVITE message. Out-of-order messages can be discarded. A rate-limiting counter is applied to throttle the number of transactions one user can initiate during one sampling interval.

Framework

The countermeasure modules have been implemented as C extension modules to the Snort IDS [115], which is supposed to be placed at the network ingress point. The authors see the implementation as a prototype and suggest the deployment of these countermeasure mechanisms in higher performing systems for real life setups. The prototype can process up to 860 malformed requests/s and introduces minimal delay. However, it crashes at higher flooding speeds in the test bed.

8.2.8 Sengar06-1

Algorithm

Sengar et al. [173, 59] propose a statistical flooding detection method dubbed "vFDS" based on Hellinger Distance calculation [174].

This work extends the detection principle of Reynolds03 by not only correlating the amount of INVITE and 200 OK messages in one sampling interval, but also extending this to ACK and BYE/CANCEL messages. The distribution of these four message types in normal traffic is compared to a distribution under attack conditions. For the comparison and computation of similarity, the Hellinger distance is calculated. It is an intrinsic way to estimate the distance between probability measures, and closely related to total variation distance. The attack threshold, i.e. which calculated Hellinger

distance value indicates an attack, is dynamically adapted based on previous monitored network parameters.

The authors also apply this method for general TCP SYN and RTP message flooding detection. This work does not provide mitigation features.

Framework

For attack detection, vFDS should be placed at the ingress network point. The authors have implemented vFDS as an add on to the Linux netfilter kernel, which is used for testing. The implementation works well to detect flooding rates of 500 INVITE messages/s or 2500 RTP messages/s. Generally, vFDS can detect attacks within several seconds; during tests with a sampling interval of 10 s, the average flooding detection time was 18 s. The addition of vFDS in the Linux router adds only a marginal delay to call setup.

8.2.9 Sengar06-2

Algorithm

Sengar et al. [54] propose an IDS dubbed "vIDS" based on interacting protocol state machines to detect message flow tampering and DoS flooding attacks.

It is based on the same cross-protocol detection approach as Wu04 and therefore targets the same attacks. While Wu04 monitors pre-defined events for correlation, vIDS is using a full state machine specification similar to Chen06. Compared to pre-defined events a state-machine specification allows more flexibility in attack detection, however the actual attack detection methods presented by the authors are the same: INVITE message flooding is detected by checking if the number of INVITES exceeds a defined threshold. The BYE message flow tampering attack is detected by synchronising the SIP and RTP state machines. The SIP state machine informs the RTP state machine when its *call tear down* state is reached. The RTP state machine continues then to the *RTP close* state. Hence, later arriving RTP packets are an indication of the attack.

To detect these attacks, attack signatures need to be defined that describe the state flow in the state machines. Theoretical, anomaly-based detection is possible if deviation in the state-flow is indicated by the state machines. Also like Wu04, they target only attack detection on UAs and not on the SIP proxy. The authors have not considered attack mitigation.

To avoid the problem that UA end-to-end RTP streams might not be visible, they propose vIDS use only for enterprise networks i.e. it is assumed

that all SIP UAs to be protected are in the same network as the main SIP proxy.

Framework

VIDS is placed at the ingress point of the network to protect where it monitors all passing traffic and evaluates it within its state machines. The authors claim an implementation of vIDS which is not further specified. VIDS has been simulated in a VoIP network using the OPNET network simulator [175]. Simulation with 20 different communicating UAs with an unspecified amount of messages shows only marginal traffic latency overheads of about 100 ms. Also, CPU processing overheads for running vIDS are marginal for the simulation scenario. Thus, the authors extrapolate that vIDS might be able to monitor "thousands of calls at the same time". For performance optimisation, vIDS conserves memory by deleting state information as soon as a call is finished.

8.2.10 Nassar06

Algorithm

Nassar et al. [176] present an IDS concept based on a Bayes inference model [177] for detecting multiple types of attacks.

The IDS considers SIP signalling attack classes, including multiple-source DoS flooding, SPIT, password cracking and vulnerability scans. Bayesian inference, as used in this work, is a statistical inference in which posterior observations are used to update the probability that a prior hypothesis may be true. Here the authors have developed a Bayes network tree where monitored network events relate to posterior observations. The prior hypothesis states that the traffic belongs to one of the introduced attack classes.

The authors define multiple monitoring parameters, like the number of ACK messages in waiting state, request and response distribution in one sampling interval, etc. Each defined parameter is given a probabilistic value for each attack class, e.g. for the DoS attack class they set P(number of ACK messages in Waiting state > 10) $= 0.9$.

Using the Bayes network tree, the actual monitored traffic is evaluated according to these parameters and the attack class is estimated. The defined probabilities are given as reasonable defaults, however the authors propose to define them from previous SIP traffic observations.

Framework

This is a theoretical concept only.

8.2.11 Rebahi07

Algorithm

Rebahi et al. [58, 178] present a method to detect DDoS flooding attacks on VoIP and IMS Systems based on Change-Point Detection with the CUSUM algorithm.

Similar to Reynolds03 the authors use the CUSUM algorithm for DoS flooding detection. While Reynolds correlates the number of INVITE and 200 OK messages, here only the number of INVITE messages are used and analysed whenever a sudden rise of INVITE messages are encountered at the proxy (i.e. the determination of the change-point, where the INVITE rate suddenly increases). Contrary to threshold-based counters, this method takes the current network condition into account. The authors correlate the parametric with the non-parametric application of the algorithm.

Framework

The authors have verified their method with an offline-analysis of SIP traffic captured from a SIP VoIP provider. They show that constant-rate flooding attacks are clearly marked as attacks and that, depending on the configuration, attacks with an increasing flooding rate are mostly discernible from regular traffic.

8.2.12 Ding07

Algorithm

Ding et al. [179] present a timed Hierarchical Coloured Petri Net IDS that is built from the work of Wu04 and Sengar06-2.

The authors incorporate both works into their IDS without any modifications. Besides the referenced work the authors only present two "methods" to handle CANCEL message flow tampering attacks. The first proposal is that the callee should simply callback the caller, while in the second one INVITEs should be re-sent after a timeout.[2]

[2]This "method" however, is based on limited assumptions. The authors assume that after an injected CANCEL by a third party the original sender would not not be notified. However, as the injected CANCEL has to share the same VIA header as the original request, the 200 OK acknowledgement will be sent back to the original sender and not to

Framework

The authors propose a NIDS-based setup at the ingress point. It is unclear if in case of the mentioned CANCEL attack the NIDS is supposed to inject packets on behalf of the attacked UA or the UA should re-send packets itself. The authors claim to have conducted simulations, however the setup is not described.

8.2.13 Nassar07

Algorithm

Nassar et al. [138] present a holistic multilayer IDS system to detect multiple different attacks based on a honeypot setup and network event correlation.

A honeypot SIP setup is deployed to lure malicious users with the intention of conducting SPIT or phishing attempts to use this setup. Once in the honeypot, senders are classified and cannot access the real SIP network later on.

An event correlator is used for DoS message flooding, and message flow tampering detection. The event correlator is the same pattern-based setup as proposed by Wu04. Likewise, attacks are detected with similar signatures. Additionally, the authors propose anomaly-based detection by generating individual SIP user profiles and detecting deviation from this profile. Flooding attacks are detected by monitoring for short inter-arrival times of requests, or by detecting open sessions by monitoring for missing ACK messages.

This method does not provide prevention features, however the authors recommend blocking identified users in the honeypot or flooding requests detected by the event correlator.

Framework

This approach proposes a distributed protection approach by deploying a fully operational but fake honeypot SIP in conjunction with the real SIP network protected by the event correlator. The authors propose a distributed IDS i.e. security events should be monitored at multiple places of interest like proxies, gateways or UAs. Instead of monitoring network traffic, events are generated directly by each call agent (e.g. by parsing log messages). Events are correlated at one central controlling instance.

---

the injecting party. Besides, the original sender will also receive a 487 Request Terminated response code in regard to its initial INVITE request. So, the "method" would only be applicable if nodes that are not fully RFC3261-compliant are involved. These possibilities have not been analysed by the authors.

The authors have implemented a prototype using the Simple Event Correlator SEC [180] as a correlator control instance. Attack patterns are fed as SEC signatures written in Perl. One event monitor has been developed for the OpenSer SIP proxy, a fork of SER. The prototype implementation only operates as a local IDS.

The performance has not been tested, however due to SEC's compact signature format the authors are expecting good scalability.

8.2.14 Barry07

Algorithm

Barry et al [181] present a combination of the work of Geneiatakis05 and Sengar06-2.

They use a layered approach with two layers. The first countermeasure layer consists of a message checker as proposed by Geneiatakis05. The second layer consists of a cross-protocol state machine specification as proposed by Sengar06-2. The authors use this system to target the same threats with the same methods.

Framework

Contrary to Sengar06-2 the authors propose a host-based intrusion detection system like Wu2004 to successfully detect BYE attacks. The authors have tested their framework with a Java implementation. This implementation was able to detect all message tampering, message flow and flooding attacks. Performance measurements are not presented.

8.2.15 Bouzida08

Algorithm

Bouzida et al. [55] present a datamining NIDS to detect multiple SIP intrusion attacks.

This work can be seen as an extension of Wu04. The authors monitor network traffic statefully and gather several attributes from it. Here attributes are finer-grained events than those introduced by Wu04. Gathered attributes can be message header fields and their values (To, From, Nonce ...), message reply codes or gathered statistical data such as the number of INVITE requests per sampling interval, for example. These attributes are correlated into profile classes (normal, known attack, new condition). In the learning phase profiles are generated using decision trees which can then be applied

to actual monitored traffic. If no profile matches (new condition), this can be an indication of a potential new attack. Hence, this work contains both signature-based and anomaly-based detection.

The authors have mostly concentrated on information-gathering attacks (user enumeration) and the usage of this information for fraud attacks (password cracking). DoS flooding attacks against individual users are detected at the proxy by threshold-based counters of different flows to each UA.

Framework

The algorithm should be implemented in a standard NIDS. The only requirement for the placement is that is sees all relevant network traffic. Testing has been done offline with an unspecified testing tool with a 2 hour traffic trace file from a real VoIP operator. The detection rate of the reviewed attacks was 99%.

8.2.16 Rieck08

Algorithm

Rieck et al. [182] present an anomaly-based self-learning system to protect against message payload tampering attacks and other potential network intrusion attacks.

Contrary to the work of Geneiatakis05 and Niccolini06-1, the goal of this method is to protect against novel attacks and so-called zero-day exploits. As an anomaly-based system, the authors train their system with normal traffic and detect deviations from this model. The feature set used for anomaly detection is made up from text strings extracted from each monitored SIP message. All text fragments are concatenated to form a new string over which a sliding window of length n (a so-called n-gram) is moved and at each position in the string the n-gram formed there is saved. The occurrence of each n-gram in a message defines the feature vector. The authors calculate the Euclidean distance of the feature vector from a "normal" feature vector. With a higher distance the probability of a message payload tampering attack or another potential intrusion increases.

The authors have taken heterogeneous SIP network setups into consideration. In such a setup a comparison to one normal vector might not be sufficient, and thus they propose different normal vectors for comparison. To protect their system from (re-)training set poisoning, they propose the combination of their system with other attack detection tools. This is especially important in the case of DoS flooding attacks as accidental re-training during

a DoS attack will yield an inaccurate normal vector.

Framework

The authors have not introduced a framework for the deployment of their algorithm. Instead, they have provided performance results with an off-line analysis of the traces captured in their test beds and from providers. Attacks are generated with a SIP traffic generator. Their unspecified off-line implementation showed good detection of the generated attacks. The false-positive rate was up to 1 %. The off-line tool was able to process 70 Mbit/s of SIP messages on "AMD Opteron Servers".

8.2.17 Nagpal08

Nagpal et al. [66] present a framework dubbed "Secure SIP" to protect against multiple DoS attacks on the proxy.

Algorithm 1

Their first line of defence is a return-routability check to detect proxy flooding from sources with spoofed addresses. The authors use a feature of the SIP specification i.e. each request can be challenged before being served. An initial request will only be served if the challenge is correctly handled by the sender. Thus simple flooding bots that do not implement the SIP specification correctly or use spoofed IP addresses cannot pass this test.

Algorithm 2

Their second line of defence is a state-machine specification to trace individual requests. It is aimed at detecting BYE flow tampering attacks caused by spurious BYE requests launched with invalid contact header fields. By following the state-machine specification it can also detect and suppress redundant messages flows, e.g. from misconfigured devices. Finally, in a similar manner to Chen06, it is also able to detect messages that do not follow the SIP state specification.

Simple DoS message flooding is detected through a standard threshold based counter as described by Iancu03. If the sending rate of one sender exceeds an upper limit, the rate is limited for following requests from that source.

Framework

The framework consists of a SIP proxy that has been enhanced to control a firewall at the ingress point through a firewall control protocol. The authors have implemented their framework with the sipd SIP proxy [183] and the hardware firewall Cloudshield CS-2000 [184]. The hardware firewall in particular was chosen for scalability reasons, as the handling of dynamic firewall rules can limit the usability of any protection mechanism. The test bed has been extensively tested with an array of 17 SUN servers serving as simultaneous attack generators. While the used SIP proxy could only handle up to 700 requests/s, the hardware firewall could handle more than 17.000 flooding requests/s.

8.3 Rating of SIP Flooding Countermeasures

The goal of this PhD work is to develop countermeasures against flooding-based DoS attacks on SIP infrastructures. The related work we present in this chapter target different DoS attacks. For a rating in comparison to our work we limit the related work to those that also address SIP flooding attacks. As a consequence, for the rating we leave out the works Geneiatakis05, Niccolini06-1 and Rieck08.

We give marks in four categories. For the *novelty* of the idea we consider a work that describes a completely new approach for DoS protection (++), a known DoS approach but new for SIP DoS protection (+), an enhancement of an already known approach (o), a variation of a known approach (-) or a plagiarism (- -).

For the *effectiveness* we distinguish between single-source and multiple-source flooding approaches. Single-source flooding is a rather straightforward approach and all approaches can handle those. The difficulty lies in multiple-source flooding detection where most of the incoming requests have different origins. To gain a (+) rating, a method must provide detection and some sort of mitigation. Accurate detection but no mitigation yields a (o) rating, approaches with a (-) can only address multiple-source floodings with a very limited number of sources.

Finally, we rate how well the method was tested, and thus the claims of the authors are *validated*, ranging from no validation at all (- -), some theoretical projections (-), simulations (o), implementation with test bed (+) and real-life tests (++).

All ratings are visible in Table 8.1.

From the table we can see that only two concepts are based on completely

Table 8.1: Rating of flooding countermeasures

	Novelty	Effectiveness (Single-Source)	Effectiveness (Multi-Source)	Validation
Iancu03	o	+	-	++
Reynolds03	+	++	o	o
Wu04	+	+	- -	o
Markl05	-	+	-	+
Chen06	+	+	-	- -
Niccolini06-2	-	+	-	+
Sengar06-1	+	++	o	+
Sengar06-2	o	+	-	+
Nassar06	o	+	-	- -
Rebahi07	o	++	o	-
Ding07	- -	+	-	-
Nassar07	+	+	-	o
Barry07	- -	+	-	o
Bouzida08	o	+	-	o
Nagpal08-1	++	+	+	+
Nagpal08-2	- -	+	-	+
Ehlert-State	o	++	-	+
Ehlert-PH	+	+	+	+
Ehlert-DNS	++	o	+	+

new ideas. The return-routability check and our non-blockable DNS cache have never been proposed for DoS protection before, as far as we can judge. All other works are variations from previous works in the DoS-countering field. This comes to no surprise, as general TCP/IP attacks have been known already for a much longer time than SIP-based DoS attacks.

For the effectiveness only multiple-source attacks pose a real challenge for researchers, and here only the work of Nagpal et al. goes in a similar direction as our work. This is analysed further in the next section.

All three of our mechanisms have been validated in test beds, which we see as the bare minimum to show the validity of an approach. Only half of the related works are validated with implementation results.

8.4 Discussion of SIP DoS Protection

Together with our work, over 20 different DoS related countermeasure mechanisms have been presented by researchers, which we have all aligned in Tables 8.2 and 8.3 for comparison[3]. We will discuss progress in the research area by considering the different DoS threats from Chapter 3 individually.

8.4.1 Payload Attacks

SIP is a text-based protocol, thus messages are human readable. A sophisticated parser is necessary to translate the human-readable message payload into a machine-readable representation. As experience has shown, flaws in such an implementation like buffer overflows or missing integrity checking can result into serious security breaches. It is therefore highly important to protect against payload attacks.

SIP is now a mature standard and the techniques used to prevent payload attacks are well known. Additionally, there are now different tools to check SIP implementations for correct operations [122, 124], thus any well-established SIP agent should generally be hardened against payload attacks. However, many different parser implementations exist and with SIP's popularity new implementations are constantly becoming available. As it is difficult to check each implementation for correct operation, a viable option for the network operator would be to add another payload attack prevention system in the form of a well-specified and tested message integrity checker as proposed by Geneiatakis05 and Niccolini06-1. This setup is also necessary if network operators are aware of implementation flaws in their devices, but there is no software or firmware update available to fix these flaws. The overhead of an additional message check is generally low as no state information needs to be maintained.

The previously mentioned proposals are signature-based and will detect attacks on known security flaws, like buffer overflows or SQL injection attacks. However, this cannot protect against new security holes. Rieck08 targets this with an anomaly-based payload checker. It is a promising approach and it would be very interesting to evaluate its efficiency in a real provider network.

8.4.2 Flow Tampering Attacks

Several message types can be used to disrupt individual SIP sessions and these attacks are known as Re-INVITE, CANCEL or BYE attacks, depend-

[3]Acronyms used in the tables: SS – Single-Source; MS – Multiple-Source

Table 8.2: Comparison of evaluated approaches, part 1

	Iancu03	Reynolds03	Wu04	Mark105	Geneiatakis05	Chen06	Niccol.06-1	Niccol.06-2	Sengar06-1	Sengar06-2	Nassar06
Principle	Counter / Rate-limiting	Open Session Detection (CUSUM)	Cross-protocol Stateful Correlation	Counter / Rate-limiting, Payload Signatures	Integrity Checker	Transaction State Model	Integrity Checker	Dialog State Model	Open Session Detection (Hellinger Distance)	Cross-protocol State Model	Bayes Inference Model
Attack	Flooding (SS)	Flooding (SS)	Flow tampering, flooding (SS)	Flooding (SS), Payload tampering	Payload tampering	Flooding (MS)	Payload tampering	Flow Tampering, Flooding (SS)	Flooding (MS)	Flooding (SS), Flow Tampering	Flooding (MS), other
Victim	Proxy	UA	UA, possibly Proxy	Proxy	Proxy, UA	Proxy	Proxy	Proxy	Proxy	UA	Proxy
Protocol	SIP	SIP	SIP, RTP	SIP	SIP	SIP	SIP	SIP	SIP, RTP	SIP, RTP	SIP
Reaction	Detection, Prevention	Detection, Prevention	Detection	Detection, Prevention	Detection, Prevention	Detection	Detection, Prevention	Detection, Prevention	Detection	Detection	Detection
Strategy	Patter-based	Anomaly-based	Pattern-based	Pattern-based	Pattern-based	Anomaly-based	Pattern-based	Pattern-based	Anomaly-based	Pattern-based	Anomaly-based
Setup	Extension Module	NIDS	NIDS, possibly distributed	NIDS	Extension Module	NIDS	HIDS	see left	NIDS	NIDS	NIDS
Implem.	C SER Extension	Simulation only	Unspecified Prototype	SIP Signatures for Snort	C Ser Extension	none	C Snort Extension		C Linux Netfilter Extension	Unspecified Implementation, OPNET simulation	none
Placem.	Proxy	Ingress Point	UA	Ingress Point	Proxy	Ingress Point	Ingress Point		Ingress Point	Ingress Point	
Measure	Proxy control (ser module)	Proxy control (NIDS)		Firewall control (SnortSam)	Proxy control (Ser module)		Snort Network Bridge Firewall				
Perform.	Limited at high message floods		Theoretical projection	Same as Snort	Same as Ser		Same as Snort		No information, low memory overhead		
Scalability	Same as Ser			Same as Snort	Same as Ser		Same as Snort				

Table 8.3: Comparison of evaluated approaches, part II

	Rebahi07	Ding07	Nassar07	Barry07	Bouzida08	Riek08	Nagpal08-1	Nagpal08-2	Ehlert-State	Ehlert-PH	Ehlert-DNS
Principle	Change-Point Detection (CUSUM)	Same as Sengar06-2	Distributed Event Correlator, Honeypot	Same as Geneiatakis05 and Sengar06-2	Cross-protocol State Model	N-Gram distance calculation	Return routability check	State Machine Model	Stateful Detection; Statistical Analysis	FW Pinholing	DNS Caching
Attack	Flooding (MS)		Flow tampering, Flooding (MS)		Flooding (SS), other	Payload tampering, other	Flooding (MS)	Flooding (SS), Flow tampering	Flooding (MS)	Flooding (MS)	Flooding (special DNS URIs)
Victim	Proxy		Proxy, UA		Proxy, UA	Proxy, UA	Proxy	Proxy, UA	Proxy	Proxy	Proxy / DNS Resolver
Protocol	SIP		SIP. RTP, other		SIP, possibly RTP	SIP	SIP	SIP	SIP	SIP	SIP
Reaction	Detection		Detection, possibly Prevention		Detection	Detection	Detection, Prevention	Detection, Mitigation	Detection, Mitigation	Detection, Mitigation	Detection, Mitigation
Strategy	Anomaly-based		Anomaly-based, Pattern-based		Anomaly-based, Pattern-based	Anomaly-based	Pattern-based	Pattern-based	Anomaly-based, Pattern-based	Pattern-based	Anomaly-based
Setup	NIDS	NIDS	Distributed HIDS	NIDS	NIDS	not evaluated	NIDS	see left	NIDS	see left	Extension Module, HIDS
Implem.	Off-line analysis calculation, provider tracefile	unspecified simulation	C OpenSer Extension, Perl SEC Signatures	Java prototype	unspecified Off-line tool	Unspecified off-line tool	HW-firewall CS-2000, Enhanced sipd Proxy		C Network Bridge (Linux Kernel), C++ Security Framework		C Ser Extension; C Enhanced Dnsmasq DNS Cache
Placem.		Ingress Point	All SIP entities	All SIP entities	Proxy, UA		Ingress Point (Proxy inside NIDS)		Ingress Point		Proxy; arbitrary (DNS Cache)
Measure							CS-2000 FW control, transaction level		Iptables FW control, transaction level		DNS Cache
Perform.			Same as SEC, OpenSer				very high throughput, high fw processing		High message throughput, performance penalty with FW usage		Same as Ser, Guaranteed Operation on the cost of reachability
Scalability			Same as SEC			Multiple CPU Scaling	Scalable HW design		Scalable Multi-host architecture		

ing on the utilised message type. As shown in this survey, multiple researchers have addressed these attacks, but in the end they are all based on the same cross-protocol stateful correlation work that was first presented by Wu04 and later used by Sengar06-2 and Nassar07. We see several problems with this method.

Monitoring Requirements This method relies on cross-protocol SIP / RTP event correlation, i.e. an IDS needs to monitor both SIP and RTP traffic. In a general SIP network, RTP traffic is routed end-to-end, hence it would not be visibly to a NIDS at the ingress point. This is either addressed by placing multiple monitoring points at all relevant network entities (Wu04, Nassar07) or by limiting protection to devices in the same domain (Sengar06-2). A third option would be to use RTP proxies, especially for NGN infrastructures. However, these options increase either administrative or network overheads or limit the protection to one domain.

Reactive measures The cross-protocol correlation method can only detect an attack, but until now there have been no proposals for preventing these types of attacks. The benefit of a complex detection-only system that has to be placed at every end devices is not well motivated, considering that its sole purpose would be to state information (an attack was detected) that is immediately visible in the end device itself (the session is terminated).

Alternatives Flow tampering attacks are only possible if an attacker can sniff necessary network parameters. If the signalling flow is encrypted it is nearly impossible to launch this type of attack. SIP already defines mature and established encryption methods, like Transport Layer Security (TLS) [81] or IPSec [82], and support for these methods is increasing in end devices. Instead of countering the effects of an attack, encryption would actually prevent the attack itself. Note that while encryption is an advisable option against flow tampering attacks, it does not help against payload attacks or flooding attacks.

Nagpal08-2 proposes a similar flow integrity checking method to detect such attacks. While it is less accurate than the cross-protocol detection schemes, it only relies on SIP monitoring and thus does not have the RTP monitoring problem. As long as SIP traffic continues to be sent unencrypted, then this seems to be a more viable option. It addresses the UA devices flawed transaction matching algorithms as described by [133].

While a cross-protocol correlator thus has only limited benefits for DoS flow tampering attacks, is is nonetheless a viable option for other flow tampering attacks, especially enumeration and fraud attacks. In fact, Bouzida08 considers these attacks in particular and not DoS flow tampering attacks in his correlation solution.

8.4.3 Flooding Attacks

Flooding attacks are the predominate form of DoS attacks. This is reflected by the research papers listed, where the majority of researchers present methods for handling flooding attacks. In this discussion we will consider only attacks directed at the main proxy and not at user agents, as the protection of the latter can easily be controlled by the proxy if the proxy itself is not attacked.

The currently established method is the threshold-based rate-limiting method, introduced by Iancu03 and used in variations in Wu04, Markl05, Niccolini06-2, Nassar06, Bouzida08 and Nagpal08-2. In its simplest form is can be used for flooding detection by counting all incoming messages, regardless of its source. For reactive measures, this mechanism needs to separate counters by considering each source individually. This causes a larger processing overhead and is still only effective against single source flooding attacks.

This method depends on the correct setting of the flooding threshold, as there are variations in the traffic load, especially in real time communication scenarios. Firstly, traffic patterns change at different times of the day and on different days of the week as communication during the night is less likely, for example. Secondly, sudden increases in traffic can occur ("flash crowds") that are not necessarily caused by a DoS attack. For instance, breaking news can cause a sudden increase in communication. These conditions should be taken into consideration when the threshold is set. Currently, most works consider only a static threshold. Some authors hint at the necessity of dynamic updates of these thresholds. But care has to be taken with traffic poisoning attacks: an attacker can slowly increase its traffic generation load to update a dynamic threshold without raising an alarm. These remain unaddressed questions.

The methods that use change-point detection (Reynolds03, Sengar06-1, Rebahi07) already take dynamic network conditions into account. Both Hellinger-distance calculation and CUSUM computation seem to be viable and resource-friendly ways to detect malicious flooding conditions. This principle has been also used in general IP-based flooding detection, bus has some limitation in that case because of the diversity of the different protocols used

[30]. This is, however, not the case in homogeneous SIP environments. The biggest drawback of this method is that it can handle only attack detection.

Another alternative to the threshold-based counters is the evaluation of state machine operations (Chen06, and our Statemachine solution). Through the analysis of a state machine it should be possible to detect attacks more accurately. However, the resource overheads increase considerably, as a lot of state information has to be maintained. Attack mitigation features are limited in the same way as the initial threshold method.

So while attack detection is working sufficiently, attack mitigation work is still limited if multiple source flooding attacks are considered.

There are currently only three approaches that address multiple source flooding mitigation. Our SIP DNS cache is able to mitigate flooding attacks with a non-blocking DNS cache solution. This method is successfully, because is takes only one special type of multiple source flooding attack into consideration. It cannot be applied to a general multiple-source flooding scenario.

Our pinholing approach and the work from Nagpal08-2 are a first step towards general multiple-source flooding attack mitigation by eliminating floods from spoofed sources. We present a firewall pinholing algorithm while Nagpal et al. introduce a return-routability checker i.e. both actively use a dedicated SIP feature for attack mitigation. It has been reported [185] that general IP DDoS attacks with spoofed IP addresses are declining in favour of distributed attacks using zombie botnets. It remains to be seen if this will also be the case in SIP environments.

So the challenge remains be to devise even better mitigation schemes against SIP DDoS flooding attacks.

8.4.4 Frameworks

A protection mechanism has only limited applicability if it does not scale with the amount of traffic encountered in real life attacks. DoS attacks, especially distributed flooding attacks, can generate a high load of traffic at the server which a protection framework should be able to process. Currently only with our VoIP Defender and with Secure SIP by Nagpal08 a dedicated protection infrastructure for protection is considered. Most of the remaining ideas have been tested using prototype implementations that have only limited scalability support. Some are also considering protection mechanism deployed directly at the to be secured host. However, this can easily lead to a self-inflicted DoS, as shown by [56]. A NIDS based-setup has better scalability.

VoIP Defender is a multi-layered architecture. Each security algorithm

which uses this framework is split into a scalable part and a non-scalable part. The framework can be deployed on multiple different hosts. It is a software-based solution with its protection based on the Linux iptables firewall component. However, this component has performance limitations with dynamic firewall rule updates. Contrarily, Secure SIP is a hardware-based solution. Thus, the firewall is easily able to dynamically update multiple thousand firewall rules per seconds. However, the algorithm intelligence is provided by one SIP proxy instance installed on the firewall. There is no way to scale the algorithm controller.

An interesting idea would thus be to combine the scalable VoIP Defender framework with the high-performance Secure SIP firewall.

Chapter 9

Conclusions

9.1 Summary and Impact

SIP is a complex protocol and also a SIP network setup can become complex as multiple different entities, probably from different providers, have to interact with each other. Here, security aspects are becoming important. What are possible attacks on SIP networks? What are the deficiencies in the SIP specification that can be exploited by malicious users? How can systems be hardened against attacks?

In this PhD work we have illuminated SIP Denial-of-Service threats with its three dominating forms *payload attacks*, *message flow tampering attacks* and *plain flooding attacks*.

SIP payload attacks can be easily handled by a correct and fail-resistant parser implementation. Together with signatures for known intrusions (like in virus protection) this proves to be an effective protection mechanism against known attacks. The best practice to counter flow tampering attacks seems to be plain message encryption. This is likely to be more effective than dedicated protection methods that have several shortcomings (see Chapter 8). If encryption is not an option, a simple SIP flow sanity checker helps preventing attacks which target implementation flaws in end devices. Unfortunately, many end devices still suffer from poor SIP implementations.

The hardest challenge are SIP DoS flooding attacks. These attacks are easy to launch, especially when the target machine is running unoptimised software that does not pay attention to possible resource depletion attacks. Thus, a first way to protect against DoS attack is to design software with depletion attacks in mind, e.g. prevent blocking states by implementing parallel operation or do not allocate unnecessarily memory until it is really needed. It is important for a network operator to make himself familiar with

the threats described in this thesis and assess the used software with the guidelines published herein.

However, hardening the software is just one step in an effective DoS countermeasure setup. Several attacks cannot be handled by the attack target itself, either while the target lacks the knowledge to detect such attacks or because the involved prevention mechanisms further increase the load at the target under attack and thus create a self-inflicting DoS attack.

It it therefore necessary to deploy external monitoring and mitigation schemes to target different DoS attack types. We have prototyped a **scalable security framework** called VoIP Defender that can handle a high message load common in DoS attacks. With the right detection methods, many attacks can be handled by this framework without the attack target being involved.

One problem of high flood Denial-of-Service is that the attacks may consist of seemingly valid messages. In this case it is difficult to differentiate malicious DoS flooding attacks from valid random traffic flash crowds, which at first sight share the same characteristics. With our proposed **state-machine detection** scheme, we have shown that a difference does indeed exist between traffic flash crowds and DoS attacks, and this difference can be made visible. As such, in contrast to common DoS detection solutions based on counting incoming messages over time, this solution has a very low false error rate. Given the right firewall setup like our VoIP Defender framework, malicious traffic flows can be selectively blocked, and hence the SIP proxy protected. Furthermore, the state machine is especially capable of detecting out-of-band redundant messages, which are common when broken user agents are in use. When multiple broken or misconfigured user agents are deployed in a network, redundant message flooding caused by such user agents can generate serious overhead load at the target proxy. A solution based on the specification-based scheme has been submitted to to the European Patent Office for patent application.

Distributed DoS attacks can only be reliably detected but not automatically prevented with the specification-based scheme. Here, another of our solutions does a much better job. Especially, when considering DDoS attack mitigation, there have no real mitigation schemes proposed in the research community until now. With our **pinholing algorithm**, the first step of an effective DDoS prevention mechanism has been established. It is a simple but effective solution to defend against special kinds of distributed Denial-of-Service attacks on SIP networks. The solution adds a marginally longer delay for regular users, but keeps the proxy safe from overhead traffic, caused by DDoS attacks. Because this solution is a lightweight scheme, it can be easily combined with other protection modules. Also for this solution a patent

application has been submitted to the European Patent Office.

Additionally, we have shown that there are multiple further attack possibilities to launch DoS attacks on SIP networks different from pure message flooding. We have especially evaluated attacks on SIP servers that utilise unresolvable DNS names in SIP messages. Such attacks might be interesting for malicious users, as such SIP messages can easily be crafted and already a low amount of such message can reduce server operation by far. Our experiments show that over-provisioning the servers with massive parallel operation and asynchronous DNS lookup capabilities as well as reducing the usage of DNS names in the SIP messages can not successfully counter such attacks. That is while such measures can improve the performance under attack a severe attack will also manage to block the server at some point. By combining **DNS caching** with a blocking threshold after which no new DNS requests are issued, we have shown that a SIP server can continue working even under heavy attack. Additionally, in case the used cache can not accommodate all possible DNS names, our experiments suggest that using a least frequently used replacement strategy for the cache has resulted in the highest hit rate and thus the best possible performance under attack.

We have validated the applicability of the presented methods by testing them within the VoIP Defender framework. The deployment in the VoIP Defender framework however is just one application. Additionally, they could be easily adopted to be implemented in commercial SIP security solutions like SBCs. Especially the pinholing algorithm would be a valuable addition to any current SBC, as it would provide a first line of defence against DDoS flooding attacks with only a limited implementation overhead.

The algorithms developed in this thesis have been the fundament of an international research project [64]. In this project, several different protection schemes against DoS attacks have been evaluated in the SIP network of a SIP VoIP service provider. Besides the mentioned patent applications for the algorithms, this work is also now cited by other researchers in the field, including H. Schulzrinne, the inventor of SIP [66], T. Peng, one of the main DoS researchers [30], IBM Research [67], University of Tübingen [68], University of Texas [69] and others.

The VoIP Defender framework has shown its usability as a SIP DoS protection framework. Additionally, the framework is completely open for extension and can easily be utilised for non-security related topics. It is currently actively being used in the FOKUS IMS Management framework where it passively checks the health status of the monitored network. This framework is currently deployed as a test bed at a third party SIP service provider. Furthermore, this framework will be the basis of an upcoming international research project which will deal with SIP message fraud [126].

9.2 Outlook

Together with the introduced related works, the methods presented in this thesis are a first but necessary step for a DoS resistant SIP infrastructure. Especially when it comes to DDoS mitigation solutions, the research work is still in its infancy. More research is needed in this direction, but this is probably a very difficult topic to handle. Researchers have struggled for years to prevent general IP based DDoS attacks with only limited success.

In fact, if there would be a chance to turn back time, it could have been advisable to limit the focus only on the DDoS mitigation topic. Simple DoS flooding detection and prevention has not turned out to be such a big obstacle to handle than anticipated. Thus, following work should focus entirely on advancing DDoS mitigation. Our pinholing method could be used as a starting point, together with the Null-authentication scheme from Nagpal et al. as introduced in Chapter 8.

The challenge will be then to devise even better mitigation schemes against SIP DDoS flooding attacks. This is a daunting task, and much research has already been conducted to handle general IP-based DDoS flooding attacks. However, chances are that mitigation might be more easily handled for SIP networks, as there is much more information available if the SIP payload is also considered by security solutions. This increases the chance to correctly classify flooding SIP traffic.

For example, in IP protection [186] it is argued that legitimate traffic tends to have different properties, while malicious flooding attacks seem to be highly correlated because traffic generators can generate the same packets to the same destination. They propose a Kolomogorov complexity-based algorithm to detect correlated traffic, i.e. DoS attack traffic. However one cannot depend on correlated DoS attacks any more if one takes the introduction of bot nets into account as they control different types of hosts with different types of captured attack generators.

Contrarily, in SIP networks, even if bot nets are involved, SIP clients to be captured and controlled are not (yet) common in the infected host. Thus the attack still has to be generated by common attack generators and thus the attack traffic is likely to be highly similar. As SIP is a text-based protocol with multiple header fields it allows easy classification of different message classes (so called finger-printing of SIP message generators). Rieck et al. [182] demonstrates such a payload attack detection method by extracting all text information from a SIP message for correlation.

Another method was proposed by Yan et al. [187] for SPIT prevention. They combine all SIP-message header fields to form a unique fingerprint of the sender. Such methods could also be easily adopted for SIP flooding

protection, by only allowing hosts with a known fingerprint class to access the service. Such methods would be a feasible option, especially for providers that also provide a standard SIP client with a known fingerprint class with their service.

Ultimately, protection should only be enforced if an attack condition is actually detected. For example, under low traffic conditions Nagpal's return routability check would unnecessarily increase latency for all users, and a fingerprinting sensor could falsely deny access to regular users even if no flooding attack is under way. Thus, it is advisable to install a lightweight detection algorithm, like a change point detection algorithm (as shown in Chapter 8), and only activate the mitigation feature if server load increases considerably due to detected flooding traffic.

Also more research is advisable to examine the interaction of different protection mechanisms. As mentioned, SIP DoS protection requires different protection methods to handle variations of the DoS threat. In this work we have already demonstrated three different possibilities to mitigate against different instances of DoS attacks. There are multiple other mitigation schemes developed by other researchers. The effects when these methods are all deployed in one IDS setup need to be further evaluated, e.g. what happens if one protection scheme indicates a DoS attack while another one does not? Also, it is still undefined how the controlling IDS should react when it receives contradicting orders how it should update its access security policy.

Another direction that should be further examined are tests in real life scenarios with very high flooding storms. If SIP's popularity continues to increase in future years, and attacks become more common, the demand for high scale tests should also raise. The VoIP Defender framework was designed with scalability in mind and should be well prepared for further tests – maybe with a new firewall engine, as the used iptables solution has some performance problems.

During the years of this research we did indeed witness a large increase in SIP usage. However, its scope has also changed during these years. Now it is becoming ever more unlikely that SIP will become the ubiquitous VoIP standard in an open Internet in a similar fashion as email. While SIP's adaptation rate is high, providers are building SIP "isles" that cannot connect to other "isles" without the help of gateways. Unhindered communication is as such not easily possible if one wants to interconnect to other SIP networks. The advent of SIP IMS has further introduced the concept of separate gated networks for communication. Maybe in such independent SIP networks Denial of Service attacks will not be the main threat in the future. Instead, billing and fraud attacks might become a much bigger threat further on. Such attacks share some characteristics with those that our specification-based approach is

able to detect, as fraud attacks often consist of flooding (albeit generally with lower intensity) attempts. Our specification-based scheme would therefore be a likely basis to counter fraud attacks.

Finally, the gained insight about SIP DoS protection will be beneficial for further researchers addressing DoS attacks on other protocols. Most network protocols utilise state-machines, so our specification-based scheme might be applicable to other protocols. Similarly, protocols that define its own message re-transmission method might benefit from the pinholing scheme, and protocols that have a high dependency on DNS systems would benefit from a dedicated DNS caching scheme.

Bibliography

[1] **Note: author's publications and submissions start here**.

[2] S. Gritzalis, S. Ehlert, G.F. Marias, Y. Rebahi, and Y. Soupionis. SPIDER: A Platform for Managing SIP-based Spam Over Internet Telephony (SPIT). *submitted for Journal of Computer Security.*

[3] S. Ehlert, A.-A. Onofrei, T. Magedanz, and D. Sisalem. A Lightweight Scheme to Protect from Distributed Denial-of-Service Attacks on SIP Infrastructures. *submitted for Security and Communication Networks.* 10, 12, 99, 133

[4] S. Ehlert, D. Geneiatakis, and T. Magedanz. Survey of Network Security Systems to Counter SIP-based Denial-of-Service Attacks. *Computers and Security Journal (to appear).* 12, 91, 169

[5] G. Kambourakis, D. Geneiatakis, S. Ehlert, J. Fiedler, and S. Gritzalis. High Availability for SIP Based Infrastructures: Solutions and Real-Time Measurements Performance Evaluation. *Journal of Universal Computer Science (to appear).*

[6] G. Zhang, S. Fischer-Hübner, and S. Ehlert. CPU-blocking Attack Possibilities on SIP VoIP Proxies. *Journal of Telecommunication Systems – Special Issue on Secure Multimedia Services (to appear).* 12, 116

[7] S. Ehlert, Y. Rebahi, and T. Magedanz. Intrusion Detection System for Denial-of-Service Flooding Attacks in SIP Communication Networks. *International Journal of Security in Networks*, 4(3), July 2009. 10, 12, 99, 133

[8] C. Lambrinoudakis, D. Geneiatakis, G. Kambourakis, A. Kafkalas, and S. Ehlert. A First Order Logic Security Verification Model for SIP. In *IEEE International Conference on Communications (ICC 2009)*, Dresden, Germany, June 2009.

[9] G. Zhang, S. Fischer-Hübner, L. Martucci, and S. Ehlert. Revealing the Calling History on SIP VoIP Systems by Timing Attacks. In *Fourth International Conference on Availability, Reliability and Security (ARES 2009)*, Fukuoka, Japan, March 2009.

[10] G.F. Marias, M. Theoharidou, Y. Soupionis, S. Ehlert, D. Gritzalis, and L. Mitrou. *IP Vulnerabilities for SPIT, SPIT Identification Criteria, Anti-SPIT Mechanisms Evaluation Framework and Legal Issues.* SIP Handbook: Services, Technologies, and Security of Session Initiation Protocol. CRC Press, December 2008.

[11] S. Ehlert, G. Zhang, and T. Magedanz. Increasing SIP Firewall Performance by Ruleset Size Limitation. In *IEEE PIMRC 2008 - VoIP Technologies Workshop*, Cannes, France, September 2008. 12, 133

[12] Y. Rebahi, S. Ehlert, and A. Bergmann. A SPIT Detection Mechanism Based on Audio Analysis. In *4th International Mobile Multimedia Communications Conference (MobiMedia 2008)*, Oulu, Finland, July 2008. 65

[13] S. Ehlert, C. Wang, T. Magedanz, and D. Sisalem. Specification-based Denial-of-Service Detection for SIP Voice-over-IP Networks. In *Third International Conference on Internet Monitoring and Protection (ICIMP2008)*, Bucharest, Romania, July 2008. 10, 12, 99

[14] S. Ehlert, G. Zhang, D. Geneiatakis, G. Kambourakis, T. Dagiuklas, J. Markl, and D. Sisalem. Two Layer Denial of Service Prevention on SIP VoIP Infrastructures. *Computer Communications*, 31(10):2443–2456, June 2008. 10, 12, 176

[15] Y. Rebahi, J.J. Pallares, T.M. Nguyen, S. Ehlert, G. Kovacs, and D. Sisalem. Performance Analysis of Identity Management in the Session Initiation Protocol. In *The 6th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-08)*, Doha, Qatar, March 2008. 65

[16] G. Zhang, S. Ehlert, T. Magedanz, and D. Sisalem. Denial of Service Attack and Prevention on SIP VoIP Infrastructures Using DNS Flooding. In *Principles, Systems and Applications of IP Telecommunications (IPTComm 2007)*, New York, USA, July 2007. 10, 12, 99, 133

[17] J. Fiedler, T. Kupka, S. Ehlert, T. Magedanz, and D. Sisalem. VoIP Defender: Highly Scalable SIP-based Security Architecture. In *Principles, Systems and Applications of IP Telecommunications (IPTComm 2007)*, New York, USA, July 2007. 12, 91, 99, 133

[18] S. Ehlert, S. Petgang, T. Magedanz, and D. Sisalem. Analysis and Signature of Skype VoIP Session Traffic. In *Fourth International Conference on Communications, Internet, and Information Technology (CIIT 2006)*, St. Thomas, US Virgin Islands, November 2006. 32

[19] D. Sisalem, J. Kuthan, and S. Ehlert. Denial of Service Attacks Targeting a SIP VoIP Infrastructure - Attack Scenarios and Prevention Mechanisms.

IEEE Network – Special Issue on Securing VoIP, 20(5):26–31, September 2006. 9, 10, 12, 63, 91

[20] D. Geneiatakis, T. Dagiuklas, G. Kambourakis, C. Lambrinoudakis, S. Gritzalis, S. Ehlert, and D. Sisalem. Survey of Security Vulnerabilities in Session Initiation Protocol. *IEEE Communications Surveys and Tutorials*, 8(3):68–81, July 2006. 9, 10, 12, 63

[21] **Note: author's publications and submissions stop here**.

[22] IP Multimedia Subsystem (IMS); Stage 2. Technical Report TS 23.238 (Release 8), 3GPP, 2007. 3, 34, 75, 115

[23] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Spark, M. Handley, and E. Schooler. *Session Initiation Protocol*, 2002. RFC 3261. 4, 8, 15, 119

[24] Packet-based Multimedia Communications Systems. Technical Report H.323, ITU-T, June 2006. 4, 32

[25] J. Rosenberg. *A Hitchhiker's Guide to the Session Initiation Protocol (SIP)*. IETF, February 2008. Interet Draft draft-ietf-sip-hitchhikers-guide-05, Work in Progress. 4, 5, 15

[26] D. Burdett. *Internet Open Trading Protocol - IOTP Version 1.0*, 2000. RFC 2801. 5

[27] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. *Network File System (NFS) version 4 Protocol*, April 2003. RFC 3530. 5

[28] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher. *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall, 2005. 6, 35, 37, 48, 146

[29] D. Geneiatakis, G. Kambourakis, C. Lambrinoudakis, T. Dagiuklas, and S. Gritzalis. A Framework for Protecting a SIP-based Infrastructure Against Malformed Message Attacks. *Computer Networks*, 51(10):2580–2593, July 2007. 7, 92, 175

[30] T. Peng, C. Leckie, and K. Ramamohnarao. Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems. *ACM Computing Surveys*, 29(1), April 2007. 8, 13, 192, 197

[31] R. Kuhn, T.J. Walsh, and S. Fries. Security Considerations for Voice Over IP Systems – Recommendations of the National Institute of Standards and Technology. Technical Report SP 800-58, National Institute of Standards and Technology, USA, January 2005. 8

[32] VoIP Security. Technical report, DATAMONITOR, November 2005. 8

[33] Emerging Cyber Threats Report for 2009. Technical report, Georgia Tech Information Security Center, October 2008. 8

[34] B. Smith. Quiet Revolution in Networking Security. In *RSA Conference 2007*, San Francisco, USA, January 2007. Conference Keynote. 8

[35] M. Collier. Protecting Your Investment: Top VoIP Security Threats. In *VoiceCon Fall 2008*, San Francisco, USA, November 2008. Panel Discussion. 8

[36] Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); NGN SECurity (SEC); Requirements. Technical Report ETSI TS 187 001 V1.1.1, ETSI TISPAN, March 2006. 8

[37] Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); TISPAN NGN Security (SEC); Threat, Vulnerability and Risk Analysis. Technical Report ETSI TS 187 002 V1.2.1, ETSI TISPAN, March 2008. 8

[38] V. Tzvetkov and H. Zuleger. Service Provider Implementation of SIP Regarding Security. In *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW 2007)*, Niagara Falls, Canada, May 2007. 8

[39] J. Larson, T. Dawson, M. Evans, and J.C. Straley. Defending VoIP Networks from Distributed DoS (DDoS) Attacks. In *VoIP Security – Challenges and Solutions Workshop*, Dallas, Texas, USA, December 2004. 8

[40] D. Sisalem. VoIP and Security - DoS, Fraud and more. In *VoIP Security – Challenges and Solutions Workshop*, Cannes, France, September 2008. 8

[41] Sipera VIPER Lab Reveals VoIP Security Threat Predictions, January 2008. SIPERA Viper Labs Press Release. 8

[42] Global VoIP Market 2008: 9th Annual Update. Technical report, iLocus, May 2008. 8

[43] N.J. Croft and M.S. Olivier. A Model for Spam Prevention in Voice over IP Networks using Anonymous Verifying Authorities. In *Fifth Annual Information Security South Africa Conference (ISSA2005)*, Sandton, South Africa, June 2005. 9, 65

[44] B. Mathieu, Q. Loudier, and Y. Gourhant. SPIT Mitigation by a Network-Level Anti-Spit Entity. In *Proc. of the 3rd Annual VoIP Security Workshop*, Berlin, Germany, June 2006. 9, 65

[45] Y. Rebahi, D. Sisalem, and T. Magedanz. SIP Spam Detection. In *International Conference on Digital Telecommunications (ICDT'06)*, Cap Esterel, France, August 2006. 9

[46] M. Sher, S. Wu, and T. Magedanz. Security Threats and Solutions for Application Server of IP Multimedia Subsystem (IMS-AS). In *IST/ITG Workshop on Monitoring and Intrusion Detection (MonAM 2006)*, Tübingen, Germany, September 2006. 9

[47] M. Zhang and Y. Fang. Security Analysis and Enhancements of 3GPP Authentication and Key Agreement Protocol. *IEEE Transactions on Wireless Communication*, 4(2):734–742, March 2005. 9

[48] S. Salsano, L. Veltri, and D. Papalilo. SIP Security Issues: The SIP Authentication Procedure and Its Processing Load. *IEEE Network*, 16(6):38–44, November 2002. 9

[49] B. Materna. Proactive Security for VoIP Networks. *Information Systems Security*, 15(2), May 2006. 9, 65

[50] R. Zhang, X. Wang, X. Yang, and X. Jiang. Billing Attacks on SIP-Based VoIP Systems. In *1st USENIX Workshop on Offensive Technology (WOOT '07)*, Boston, USA, August 2007. 9, 65

[51] S. Vuong and Y. Bai. A Survey of VoIP Intrusions and Intrusion Detection Systems. In *6th International Conference on Advanced Communication Technology (ICACT 2004)*, Phoenix Park, South Korea, February 2004. 9, 10

[52] VoIP Security and Privacy Threat Taxonomy. Technical Report Public Release 1.0, VOIPSA, October 2005. http://www.voipsa.org. 9

[53] Y.-S. Wu, S. Bagchi, S. Garg, N. Singh, and T. Tsai. SCIDIVE: A Stateful and Cross Protocol Intrusion Detection Architecture for Voice-over-IP Environments. In *International Conference on Dependable Systems and Networks (DSN-2004)*, Firenze, Italy, July 2004. 9, 10, 92, 174

[54] H. Sengar, D. Wijesekera, H. Wang, and S. Jajodia. VoIP Intrusion Detection Through Interacting Protocol State Machines. In *International Conference on Dependable Systems and Networks (DSN-2006)*, Philadelphia, USA, June 2006. 9, 92, 178

[55] Y. Bouzida and C. Mangin. A Framework for Detecting Anomalies in VoIP Networks. In *Third International Conference on Availability, Reliability and Security (ARES 08)*, Barcelona, Spain, March 2008. 9, 182

[56] M. Luo, T. Peng, and C. Leckie. CPU-based DoS Attacks Against SIP Servers. In *IEEE/IFIP Network Operations and Management Symposium (NOMS 2008)*, Salvador, Bahia, Brazil, April 2008. 9, 97, 104, 157, 192

[57] B. Reynolds and D. Ghosal. Secure IP Telephony using Multi-layered Protection. In *10th Annual Network and Distributed System Security Symposium*, San Diego, USA, February 2003. 9, 10, 102, 173

[58] Y. Rebahi. Change-Point Detection for Voice over IP Denial of Service Attacks. In *15. ITG/GI - Fachtagung Kommunikation in Verteilten Systemen (KiVS 2007)*, Bern, Switzerland, February 2007. 9, 180

[59] H. Sengar, H. Wang, D. Wijesekera, and S. Jajodia. Detecting VoIP Floods using the Hellinger Distance. *IEEE Transactions on Parallel and Distributed Systems*, 19(6):794–805, June 2008. 9, 10, 177

[60] B. Iancu. SER PIKE Excessive Traffic Monitoring Module, 2003. http://www.iptel.org/ser/doc/modules/pike. 10, 172

[61] Y. Rebahi, D. Sisalem, J. Kuthan, A. Pelinescu-Oncicul, B. Iancu, J. Janak, and D. C. Mierla. The SIP Express Router - An Open Source SIP Platform. In *Evolute Workshop*, Guildford, UK, 2003. http://www.iptel.org/ser. 10, 59

[62] Borderware SIPassure VoIP / SIP Security Solution. http://www.borderware.com/products/sipassure. 10, 112, 172

[63] AcmePacket Net Net SBC. http://www.acmepacket.com. 10, 172

[64] SNOCER EU Security Project for VoIP DoS Protection, 2004-2006. FP6-2002-SME-1-005892, http://www.snocer.org. 10, 63, 197

[65] SPIDER EU Security Project for VoIP SPIT Protection, 2006-2008. FP6-2006-SME-COOP-32720, http://projectspider.org. 12, 65

[66] S. Nagpal, E. Yardeni, H. Schulzrinne, and G. Ormazabal. Secure SIP: A Scalable Prevention Mechanism for DoS Attacks on SIP-based VoIP Systems. In *Principles, Systems and Applications of IP Telecommunications (IPTComm 2008)*, Heidelberg, Germany, July 2008. 12, 184, 197

[67] V. A. Balasubramaniyan, A. Acharya, M. Ahamad, M. Srivatsa, I. Dacosta, and C. P. Wright. SERvartuka: Dynamic Distribution of State to Improve SIP Server Scalability. In *28th International Conference on Distributed Computing Systems*, Los Alamitos, CA, USA, 2008. IEEE Computer Society. 13, 197

[68] A. Fessi, H. Niedermayer, H. Kinkelin, and G. Carle. A Cooperative SIP Infrastructure for Highly Reliable Telecommunication Services. In *Principles, Systems and Applications of IP Telecommunications (IPTComm '07)*, New York, NY, USA, 2007. ACM. 13, 197

[69] P. Gupta and V. Shmatikov. Security Analysis of Voice-over-IP Protocols. In *20th IEEE Computer Security Foundations Symposium (CSF '07)*, Venice, Italy, July 2007. 13, 197

[70] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. *SIP: Session Initiation Protocol*, March 1999. RFC 2543 (obsoleted). 15, 84

[71] J.H. Saltzer, D.P. Reed, and D.D. Clark. End-to-end Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277 – 288, November 1984. 15

[72] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, June 1999. RFC 2616. 15

[73] J. Klensin. *Simple Mail Transfer Protocol*, April 2001. RFC 2821. 15, 110

[74] A. Johnston. *SIP: Understanding the Session Initiation Protocol*. Artech House, 2nd edition, November 2003. 15

[75] H. Sinnreich and A. Johnston. *Internet Communications Using SIP: Delivering VoIP and Multimedia Services with Session Initiation Protocol*. Wiley, 2nd edition, July 2006. 15

[76] T. Russel. *Session Initiation Protocol (SIP): Controlling Convergent Networks*. McGraw-Hill Osborne Media, 1st edition, June 2008. 15

[77] F. Yergeau. *UTF-8, a Transformation Format of ISO 10646*, January 1998. RFC 2279. 20

[78] N. Freed and N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, November 1996. RFC 2045. 21

[79] J. Arkko, V. Torvinen, G. Camarillo, A. Niemi, and T. Haukka. *Security Mechanism Agreement for the Session Initiation Protocol*, January 2003. RFC 3329. 28

[80] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. *HTTP Authentication: Basic and Digest Access Authentication*, 1999. RFC 2617. 29, 67

[81] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.1*, April 2006. RFC 4346. 29, 42, 92, 190

[82] S. Kent and K. Seo. *Security Architecture for the Internet Protocol*, December 2005. RFC 4301. 30, 92, 190

[83] B. Ramsdell. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification*, July 2004. RFC 3851. 30

[84] M. Handley, V. Jacobson, and C. Perkins. *SDP: Session Description Protocol*, July 2006. RFC 4566. 30

[85] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*, July 2003. RFC 3550. 31

[86] Skype – The Whole World Can Talk for Free. http://www.skype.com. 32

[87] S. A. Baset and H. Schulzrinne. An analysis of the Skype P2P internet telephony protocol. *IEEE Infocom*, 2006. 32

[88] M. Spencer, B. Capouch, F. Miller, and K. Shumard. *IAX: Inter-Asterisk eXchange Version 2*, March 2008. Interet Draft draft-guy-iax-04, Work in Progress. 32

[89] Cisco Skinny Call Control Protocol (SCCP) Specification Introduction. http://www.cisco.com/en/US/tech/tk652/tk701/tk589/tsd_technology_support_sub-protocol_home.html. 32

[90] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. *Diameter Base Protocol*, September 2003. RFC 3588. 34

[91] Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); NGN Functional Architecture. Technical Report ETSI ES 282 001 V2.0.0, ETSI TISPAN, March 2008. 35

[92] D. Dittrich. Distributed Denial of Service (DDoS) Attacks/tools. http://staff.washington.edu/dittrich/misc/ddos, Reference page about DoS attacks, tool and events. Last updated Aug. 2008. 36

[93] Ping of Death Exploit. http://insecure.org/sploits/ping-o-death.html. 36

[94] J. Postel. *Internet Control Message Protocol*, September 1981. RFC 792. 36

[95] CERT Advisory CA-1997-28 IP Denial-of-Service Attacks. http://www.cert.org/advisories/CA-1997-28.html. 37

[96] CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks. http://www.cert.org/advisories/CA-1996-21.html. 38

[97] D. Moore, G. Voelker, and S. Savage. Performance Comparison of Different Cache-Replacement Policies for Video Distribution in CDN. In *10th USENIX Security Symposium, pages 9–22s*, 2001. 39

[98] B. Schneier. *Applied Cryptography.* John Wiley & Sons, 2nd edition, 1996. 40

[99] U.S. Department of Commerce / National Institute of Standards and Technology. *Data Encryption Standard (DES)*, October 1999. FIPS PUB 46-3. 40

[100] U.S. Department of Commerce / National Institute of Standards and Technology. *Advanced Encrption Standard (AES)*, November 2001. FIPS PUB 197. 40

[101] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978. 40

[102] T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985. 40

[103] R. Rivest. *The MD5 Message-Digest Algorithm*, April 1992. RFC 1321. 41

[104] U.S. Department of Commerce / National Institute of Standards and Technology. *Secure Hash Standard*, August 2002. FIPS PUB 180-2. 41

[105] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*, February 1997. RFC 2104. 41

[106] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976. 42

[107] CERT Incident Note IN-99-07 Distributed Denial of Service Tools. http://www.cert.org/incident_notes/IN-99-07.html. 44

[108] P. Ferguson and D. Senie. *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*, March 1998. RFC 2267. 49

[109] M. Yung, P. Janson, and G. Tsudik. Scalability and Flexibility in Authentication Services: The KryptoKnight Approach. In *IEEE INFOCOM 1997*, Tokyo, Japan, April 1997. 51

[110] T. Aura and P. Nikander. Stateless Connections. In *International Conference on Information and Communications Security (ICICS'97)*, 1997. 51

[111] P. Karn and W. Simpson. *Photuris: Session-Key Management Protocol*, 1998. RFC 2522. 52

[112] C. Dwork and M. Naor. Pricing via Processing or Combatting Junk Mail. In *CRYPTO'92*, 1992. 53

[113] A. Juels and J. Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *Network and Distributed System Security Symposium (NDSS'99)*, 1999. 53

[114] T. Aura, J. Leiwo, and P. Nikander. Towards Network Denial of Service Resistant Protocols. In *15th International Information Security Conference (IFIP/SEC 2000), Bejing, China*, August 2000. 53

[115] M. Roesch. Snort – Lightweight Intrusion Detection for Networks. In *13th USENIX Large Installation System Administration Conference (LISA'99)*, Seattle, USA, November 1999. 54, 56, 175, 177

[116] Prelude IDS Communication System. http://www.prelude-ids.org. 55, 57, 176

[117] Tripwire Host-based Intrusion Detection System. http://www.tripwire.org. 57

[118] The Simple WATCHer (SWATCH). http://swatch.sourceforge.net. 57

[119] Stefan Axelsson. Intrusion Detection Systems: A Survey and Taxonomy. Technical report, Chalmers University of Technology, Goteborg, Sweden, 2000. 58

[120] Dorothy E. Denning. An Intrusion Detection Model. *IEEE Transactions on Software Engineering, SE-13(2):222-232*, Feb 1987. 58

[121] Netfilter / iptables Firewall. http://www.netfilter.org. 59

[122] C. Wieser, M. Laakso, and H. Schulzrinne. Security Testing of SIP implementations. Technical Report CUCS-024-03, Columbia University, Department of Computer Science, New York, USA, 2003. http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip. 61, 91, 187

[123] SIPp – SIP Traffic Generator. http://sipp.sourceforge.net. 61

[124] H. Abdelnur, O. Festor, and R. State. KiF: A stateful SIP Fuzzer. In *Principles, Systems and Applications of IP Telecommunications (IPTComm 2007)*, New York, USA, July 2007. 65, 91, 187

[125] X. Wang, R. Zhang, X. Yang, X. Jiang, and D. Wijesekera. Voice Pharming Attack and the Trust of VoIP. In *4th International Conference on Security and Privacy in Communication Networks (SecureComm 2008)*, Istambul, Turkey, September 2008. 65

[126] SCAMSTOP EU Security Project for VoIP Fraud Protection, 2009-2011. FP7-SME-COOP, http://www.fokus.fraunhofer.de/en/ngni/projects/ current_projects/scamstop. 65, 197

[127] J. Rosenberg and C. Jennings. *The Session Initiation Protocol (SIP) and Spam*, January 2008. RFC 5039. 65

[128] J. Quittek, S. Niccolini, S. Tartarelli, M. Stiemerling, M. Brunner, and T. Ewald. Detecting SPIT Calls by Checking Human Communication Patterns. In *IEEE International Conference on Communications (ICC '07)*, Glasgow, UK, June 2007. 65

[129] J. Peterson and C. Jennings. *Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)*, August 2006. RFC 4474. 65, 116

[130] D. Geneiatakis, G. Kambourakis, C. Lambrinoudakis, T. Dagiuklas, and S. Gritzalis. SIP Message Tampering: The SQL Code Injection Attack. In *13th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2005)*, Split, Croatia, September 2005. 67, 175

[131] C. Anley. Advanced SQL Injection In SQL Server Applications. Technical report, 2002. An NGS Software Insight Security Research (NISR) Publication. 67

[132] MySQL Open Source Database. http://www.mysql.com. 67

[133] J. Seedorf, K. Beckers, and F. Huici. Testing Dialog-Verification of SIP Phones with Single-Message Denial-of-Service Attacks. In *4th International Conference on Global E-Security*, London, UK, June 2008. 69, 190

[134] R. Sparks. *The Session Initiation Protocol (SIP) Refer Method*, 2003. RFC 3515. 73

[135] R. Sparks. *The Session Initiation Protocol (SIP) Referred-By Mechanism*, September 2004. RFC 3892. 73

[136] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. *STUN - Simple Traversal of UDP Through NATs*, March 2003. RFC 3489. 77

[137] S. Niccolini, R.S. Garroppo, S. Giordano, G. Risi, and S. Ventura. SIP Intrusion Detection and Prevention: Recommendations and Prototype Implementation. In *1st IEEE Workshop on VoIP Management and Security*, Vancouver, Canada, April 2006. 92, 176

[138] M. Nassar, S. Niccolini, R. State, and T. Ewald. Holistic VoIP Intrusion Detection and Prevention System. In *Principles, Systems and Applications of IP Telecommunications (IPTComm 2007)*, New York, USA, July 2007. 92, 181

[139] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. *Request Header Integrity in SIP and HTTP Digest using Predictive Nonces*, June 2001. expired Internet Draft, work in progress, draft-rosenberg-sip-http-pnonce-00.txt. 93

[140] E. Harris. The Next Step in the Spam Control War: Greylisting. Technical report, Greylisting Project, 2004. http://projects.puremagic.com/greylisting/whitepaper.html. 110

[141] P.V. Mockapetris. *Domain Names – Concepts and Facilities*, November 1987. RFC 1034. 113

[142] J. Rosenberg and H. Schulzrinne. *Session Initiation Protocol (SIP): Locating SIP Servers*, 2002. RFC 3263. 115, 121

[143] J. Peterson, H. Liu, J. Yu, and B. Campbell. *Using E.164 Numbers with the Session Initiation Protocol (SIP)*, 2004. RFC 3824. 115

[144] A. Gulbrandsen, P. Viexie, and L. Esibov. *A DNS RR for Specifying the Location of Services (DNS SRV)*, February 2000. RFC 2782. 115

[145] V. Pappas, Z. Xu, S. Lu, D. Massey, and L. Zhang. Impact of Configuration Errors on DNS Robustness. 2004. 115

[146] Berkeley Internet Name Domain (BIND). Open source DNS server, http://www.isc.org. 116

[147] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, USA, 2000. 128

[148] C. Aggarwal, J. Wolf, and P. Yu. Caching on the World Wide Web. *IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 1*, 1999. 130, 132

[149] Dnsmasq DNS Caching Proxy. Project web page for documentation: http://www.thekelleys.org.uk/dnsmasq/doc.html. 131

[150] A. Silberschatz and P.B. Galvin. *Operating System Concepts, Fourth ed.* Addison-Wesley, 1994. 132

[151] D. Hoffman, D. Prabhakar, and P. Strooper. Testing iptables. In *2003 conference of the Centre for Advanced Studies on Collaborative research (CASCON'03)*, pages 80–91. IBM Press, 2003. 143

[152] J. Kadlecsik and G. Pásztor. Netfilter Performance Testing. Technical report, NetFilter Project, 2005. http://people.netfilter.org/kadlec/nftest.pdf. 143

[153] S. Meier. Entwurf einer Software-Schnittstelle zur betriebssytemunabhängigen Firewall-Konfiguration (English Document). Master's thesis, University of Stuttgart, Communication Network Department, Stuttgart, Germany, October 2007. 143

[154] A. Hussain, J. Heidemann, and C. Papadopoulas. A Framework for Classifying Denial of Service Attacks. In *2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003. 146

[155] J. Markl, D. Sisalem, S. Ehlert, D. Geneiatakis, G. Kambourakis, T. Dagiuklas, M. Rokos, O. Bontron, J. Rodriguez, and J. Liu. General Reliability and Security Framework for VoIP Infrastructures. Technical report, September 2005. Deliverable SNOCER-D2.2, http://www.snocer.org. 146, 175

[156] U. Chejaral, H.-K. Chail, and H. Chol. Performance Comparison of Different Cache-Replacement Policies for Video Distribution in CDN. In *7th IEEE International Conference on High Speed Networks and Multimedia Communications*, 2004. 150

[157] IP Set Firewall. Project Webpage: http://ipset.netfilter.org. 155

[158] M. Bellion. Optimizing HiPAC for Policy Routing, Packet Filtering and Traffic Control. Diplomarbeit, Universität des Saarlandes, 2004. http://www.hipac.org. 155

[159] Y. Qiu, J. Zhou, and F. Bao. Design and Optimize Firewalls for Mobile Networks. In *Proceedings of IEEE 60th Vehicular Technology Conference*, 2004. 157

[160] B. Potter. Open Source Firewall Alternatives. *Network Security*, 2006(1), January 2006. 157

[161] P. Gupta and N. McKeown. Packet Classification on Multiple Fields. In *Conference on Application, Technologies, Architectures and Protocols for Computer Communication*, 1999. 157

[162] A. X. Liu and M. G. Gouda. Removing Redundancy from Packet Classifiers. In *Proceedings of ACM SIGCOMM*, 2004. 158

[163] M. Yoon, S. Chen, and Z. Zhang. Reducing the Size of Rule Set in a Firewall. In *Proceedings of IEEE International Conference on Communications, ICC' 07*, 2007. 158

[164] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman Co., New York, NY, USA, 1979. 166

[165] R. L. Rardin. *Optimization in Operations Research*. Prentice Hall, 1998. 166

[166] E.S. Page. Continuous Inspection Schemes. *Biometrika*, 41(1/2):100–115, June 1954. 173

[167] H. Wang, D. Zhang, and K. Shin. Detecting SYN Flooding Attacks. In *IEEE INFOCOM 2002*, New York, USA, June 2002. 173

[168] D. Geneiatakis, G. Kambourakis, T. Dagiuklas, C. Lambrinoudakis, and S. Gritzalis. A Framework for Detecting Malformed Messages in SIP Networks. In *14th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN)*, Chania, Greece, September 2005. 175

[169] Perl Compatible Regular Expressions. http://www.pcre.org. 175

[170] SnortSam, Firewall Control Plugin for the Snort IDS system. http://www.snortsam.org. 176

[171] E. Y. Chen. Detecting DoS Attacks on SIP Systems. In *1st IEEE Workshop on VoIP Management and Security*, Vancouver, Canada, April 2006. 176

[172] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based Anomaly Detection: A New Approach for Detecting Network Intrusions. In *9th ACM Computer and Communication Security Conference (CCS 2002)*, Washington DC, USA, November 2002. 176

[173] H. Sengar, D. Wijesekera, H. Wang, and S. Jajodia. Fast Detection of Denial of Service Attacks on IP Telephone. In *14th IEEE International Workshop on Quality of Service*, New Haven, CT, USA, June 2006. 177

[174] E. Hellinger. Neue Begründung der Theorie quadratischer Formen von unendlichvielen Veränderlichen. *Journal für die reine und angewandte Mathematik*, 136(3/4), 1909. 177

[175] OPNET - Optimum Network Performance Modeler. http://www.opnet.com. 179

[176] M. Nassar, R. State, and O. Festor. Intrusion Detection Mechanisms for VoIP Applications. In *3rd Annual VoIP Security Workshop, Berlin, Germany*, 2006. 179

[177] S.M. Stigler. Thomas Bayes' Bayesian Inference. *Journal of the Royal Statistical Society, Series A (General)*, 145(2), 1982. 179

[178] Y. Rebahi, M. Sher, and T. Magedanz. Detecting Flooding Attacks Against IP Multimedia Subsystem (IMS) Networks. In *The 6th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-08)*, Doha, Qatar, March 2008. 180

[179] Y. Ding and G. Su. Intrusion Detection System for Signal based SIP Attacks through Timed HCPN. In *2nd International Conference on Availability, Reliability and Security (ARES 2007)*, Vienna, Austria, April 2007. 180

[180] SEC - Simple Event Correlator. http://www.estpak.ee/ risto/sec. 182

[181] B.I.A. Barry and H.A. Chan. A Hybrid, Stateful and Cross-Protocol Intrusion Detection System for Converged Applications. In *International Conference on Grid computing, high-performance and Distributed Applications (GADA'07)*, Vilamoura, Portugal, November 2007. 182

[182] K. Rieck, S. Wahl, P. Laskov, P. Domschitz, and K.-R. Müller. A Self-Learning System for Detection of Anomalous SIP Messages. In *Principles, Systems and Applications of IP Telecommunications (IPTComm 2008)*, Heidelberg, Germany, July 2008. 183, 198

[183] sipd – Columbia InterNet Extensible Multimedia Architecture SIP proxy, December 2002. http://www.cs.columbia.edu/IRT/cinema. 185

[184] Cloudshield CS-2000 Content Processing Platform. http://www.cloudshield.com/Products/cs2000.asp. 185

[185] M. Handley. DoS-resistant Internet - Subgroup Report. Technical report, Internet Architecture Working Group: DoS-resistant Internet Working Subgroup, 2005. www.communicationsresearch.net/object/download/1543/doc/mjh-dos-summary.pdf. 192

[186] A.B. Kulkarni and S.F. Bush. Detecting Distributed Denial-of-Service Attacks using Kolmogorov Complexity Metrics. *Journal of Network and Systems Management*, 14(1):69–80, March 2006. 198

[187] H. Yan, K. Sripanidkulchai, H. Zhang, and Z. Shae. Incorporating Active Fingerprinting into SPIT Prevention Systems. In *Third International VoIP Security Workshop*, Berlin. Germany, June 2006. 198

List of Abbreviations

3GPP	3rd Generation Partnership Project, 35
AAA	Authentication, Authorization, and Accounting, 35
AES	Advanced Encryption Standard, 41
ALG	Application Layer Gateway, 157
AS	Application Server, 35
CSCF	Call Session Control Function, 35
DDoS	Distributed DoS, 35
DES	Data Encryption Standard, 41
DH	Diffie-Hellman, 43
DNS	Domain Names Service, 114
DoS	Denial of Service, 35
DSL	Digital Subscriber Line, 35
ETSI	European Telecommunications Standards Institute, 35
FIFO	First in first out, 132
FMC	Fixed-Mobile Convergence, 35
FQDN	Fully Qualified Domain Name, 115
GF	Gateway Function, 35
GSS	Golden Section Search, 166
HIDS	Host IDS, 57

HMAC	Hash MAC, 41
HSS	Home Subscription Server, 35
HTTP	Hypertext Transport Protocol, 15
I-CSCF	Interrogating CSCF, 35
IAX	Inter Asterisk Exchange, 32
ICMP	Internet Control Message Protocol, 37
IDS	Intrusion Detection System, 55
IETF	Internet Engineering Task Force, 15
IMS	Internet Multimedia Subsystem, 35
IPTV	IP Television, 35
ISP	Internet Service Provider, 38
ITU-T	International Telecommunication Union - Telecommunication Standardization Sector, 32
LFU	Least frequently used, 132
LRU	Least recently used, 132
MAC	Message Authentication Code, 41
MD5	Message Digest 5, 41
MDC	Modification Detection Code, 41
MIME	Multipurpose Internet Mail Extensions, 21
MRF	Media Resource Function, 35
NGN	Next-Generation-Network, 35
NIDS	Network IDS, 56
P-CSCF	Proxy CSCF, 35
P2P	Peer-to-Peer, 32
PBX	Private Branch Exchange, 32

PSTN	Public Switched Telephony Network, 33
QoS	Quality-of-Service, 35
RFC	Request for Comment, 15
RSA	Rivest Shamir Adleman, 41
RTCP	Real-time Transport Control Protocol, 32
RTP	Real-time Transport Protocol, 32
S-CSCF	Serving CSCF, 35
S/MIME	Secure MIME, 28
SBC	Session Border Controller, 8
SCCP	Skinny Call Control Protocol, 32
SDP	Session Description Protocol, 31
SER	SIP Express Router, 59
SHA	Secure Hash Algorithm, 41
SIP	Session Initiation Protocol, 15
SMTP	Simple Mail Transfer Protocol, 15
TC	Time cost, 132
TISPAN	Telecoms & Internet converged Services & Protocols for Advanced Networks, 35
TLS	Transport Layer Security, 28
TU	Transaction User, 20
UA	User agent, 16
UAC	User agent client, 16
UAS	User agent server, 16
UCS	Universal Character Set, 21
URI	Uniform Resource Identifier, 15

UTF UCS Transformation Format, 21

VoIP Voice-over-IP, 15