Georg-August-Universität
Göttingen
Zentrum für Informatik

ISSN 1612-6793
Nummer ZFI-BM-2007-39

Masterarbeit

im Studiengang "Angewandte Informatik"

Diameter WebAuth: An AAA-based Identity Management Framework for Web Applications

Niklas Neumann

am Institut für

Informatik

Bachelor- und Masterarbeiten
des Zentrums für Informatik
an der Georg-August-Universität Göttingen

12. November 2007

Georg-August-Universität Göttingen
Zentrum für Informatik

Lotzestraße 16-18
37083 Göttingen
Germany

Tel. +49 (5 51) 39-1 44 14
Fax +49 (5 51) 39-1 44 15
Email office@informatik.uni-goettingen.de
WWW www.informatik.uni-goettingen.de

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Göttingen, den 12. November 2007

Master's thesis

Diameter WebAuth: An AAA-based Identity Management Framework for Web Applications

Niklas Neumann

November 12, 2007

Supervised by Prof. Dr. Fu
Computer Networks Group
Institute for Computer Science
Georg-August-Universität Göttingen

Abstract

Every day countless users are accessing various personal and personalized information on the Internet, especially the World Wide Web. In order to provide each user proper access, web applications need to be able to establish the user's identity. Identity management is a concept to unify and facilitate such user identification.

The objective of this thesis is to introduce and explore identity management in web applications. First, existing identity management approaches are analyzed and evaluated. Based on the results of this evaluation, a new AAA-based identity management framework, the so-called Diameter WebAuth, is proposed. The proposal is based on the Diameter protocol and intended for an easy deployment in web applications. By using Diameter as basis, the proposal takes advantage of existing Diameter functions and specifications and can be seamlessly integrated into existing Diameter setups. Diameter WebAuth provides features comparable to web-based identity management solutions such as OpenID, the Liberty Alliance project and Microsoft CardSpace.

Keywords

Identity management; digital identity; web application; AAA; network access; authentication, authorization, Diameter; WebAuth; OpenID; Liberty Alliance; Mircrosoft CardSpace; Kerberos.

Contents

1. Introduction

Information and services offered on the World Wide Web can be divided into two categories: those that are independent of the user who is accessing them, and those that are not. A web site that offers personal or personalized data to its users needs to establish their identity in order to match the proper data to the proper user. This applies to incoming data like a post on a message board, for example, in which case the site wants to store the name and email address of the poster along with the message. It also applies to outgoing data like the web frontend for an email account where the web site needs to ensure that only the person who has rightfull access to the account is allowed to use the web frontend. In any case the web site needs to establish the identity of its user.

To determine a user's identity the web site asks a user for his credentials. Usually this will be a username of some sort in combination with a secret password which is used to identify the user. After a successful identification the user can be associated with previously supplied data like favorite topics, an address or a billing account. This data is then used to provide the desired service. However, since those credentials and the associated identity are biletarally negotiated between the user and the web site, they are only valid within the scope of the particular web site. With a growing number of enlisted services, the number of such established identities grows as well.

As a result, increased administrative efforts emerge for the users as well as for the individual service providers. The users have to keep track of numerous identities and need to reenter personal data each time they subscribe for a new service. To minimize these efforts, passwords are used again with different identities which may introduce security risks. For example, having equal passwords at different sites exposes them to the risk of being compromised all at once. It also allows a service provider which has access to the cleartext credentials to abuse the user's identity at other service providers. On the side of the service providers' authentication services and databases have to be setup and registration services developed. Furthermore, those databases have to be maintained and regularly checked for old entries to keep their size within a limit. Last but not least, an extensive sign-up process to subscribe to a service is also an entrance barrier for a new user.

The concept of Identity Management describes the handling of such digital identities [24, 36]. There are several approaches to enable identity management in web applica-

tions like OpenID [41], the Liberty Alliance project [45] or Microsoft CardSpace [33]. They emerged rather recently, are specifically designed for web applications and take the problems and characteristics of a web environment into account.

On the other hand, the problem of access control has been extensively explored in the networking field. Network access and authentication, authorization and accounting (AAA) protocols [15] have been developed and deployed to verify and control access privileges of network users. Those protocols are well established and mature. They are, however, not designed for the web environment and cannot be used together with web applications without special adaptations. Examples for network access control protocols are Radius [50], Diameter [8] or Kerberos [37].

1.1. Motivation and scope

The thesis is motivated by the challenge to explore identity management approaches for web applications. Noticing, that there are existing solutions from the field of network authentication, it will work out a proposal to extend a network AAA protocol for identity management purposes in web applications. The objective is to allow existing AAA infrastructures to support identity management operations in the application layer. This will unify AAA infrastructures and identity management infrastructures on basis of well established and mature protocols and services.

The goal of the thesis is to explore the possibility of adopting network-based access protocol concepts for the purpose of providing identity management functions to web applications. To achieve this goal, it has to effectively fulfill the requirements of identity management in a web-based environment, using the facilities of a network authentication protocol. The thesis will cover an analysis and comparison of existing web-based and network-based approaches to identity management on the basis of common criteria. The result of this analysis will then be used as basis for developing a new proposal to identity management for web-based applications. This new proposal will be developed on top of existing AAA structures, allowing to integrate it in already operating AAA systems without further effort. By using existing protocols, the proposal will also take advantage of previous work done to specify and develop those protocols. After the design, the proposal will be validated by means of a prototype implementation.

1.2. Basic concepts

In order to understand the work detailed in the thesis, a number of basic concepts need to be known established. The concepts of identity management and web-based

authentication are, therefore, shortly introduced here.

1.2.1. Identity management

In order to offer personalized services to consumers, service providers need to be able to distinguish their customers. Therefore the customer's identity is established and linked to the services provided. In the electronic world a suspects identity is usually described with a set of attributes and an identifier. The main purpose of such an electronic identity is to identify and recognize an individual and link it to a service specific account. The account, in turn, holds the information that are necessary for the service provider to render the personalized services that are expected by the client. An electronic identity often comes down to a single identifier like a screen name or an email address which serves as a lookup key to the rest of the personal information. A service provider can, for example, store consumer preferences, purchase histories or shipment addresses linked to the identity. Such additional describing information are called identity attributes [24, 44]. In web-based environments, service providers offering their services using web applications to customers, are also called application providers.

Identity management (IdM) describes actions that handle such identities. Examples for such actions are to establish an identity (enrollment), supplying services to a user and enabling features for him (provisioning), verify that a client is correctly linked to an identity (authentication) and to destroy an identity (de-provisioning) [36]. An identity management provider (IdMP) operates the identity management systems (IdMS) which processes the IdM tasks for its clients. Besides the technical aspects, there are also legal and social aspects to identity management which are not further explored in this thesis [24, 43].

1.2.2. Web-based authentication

Web-based environments are communication infrastructures, usually networks, that use technologies which were developed for the World Wide Web. For example, the Hypertext Transfer Protocol (HTTP), the (Extensible) Hypertext Markup Language ((X)HTML) or Uniform Resource Identifiers (URIs). The biggest web-environment is obviously the World Wide Web itself; however, web technologies are also used in home, private or company networks. There are two basic alternatives for authentication in such environments: authentication using facilities of the communication protocol (HTTP in this case) or authentication using application specific methods and parameters that are exchanged using forms. Usually an authentication is performed to identify the end user to the server. There are also a number of (implicit) methods to authenticate a server to a user. For ex-

ample, the URI which a server responds to can be used to authenticate it or information provided in a SSL certificate that the server offers to secure communication. The following sections, however, focus on user authentication; the concept of server authentication will not be pursued further in this thesis.

HTTP authentication

RFC 2617 [18] specifies an authentication extension to HTTP. This extension allows for the exchange of authentication information on the protocol level. This allows, for example, web servers that only understand HTTP to enforce a user authentication. In contrast to form-based authentication (see below) the user client has to support the authentication method since it is required to process it. The user client, for example, has to recognize a HTTP authentication request, prompt the user for his credentials and send them to the authenticator. Also the HTTP authentication methods only arrange for a username and a password as authentication credentials. Other parameters like an authentication provider or a controll challenge, therefore, need to be guessed from the context or encoded into the username/password tokens.

There are two authentication methods specified in RFC 2617: basic and digest access authentication. The basic authentication transfers username and password in cleartext. This allows for a direct comparison on the receiving end of the authentication with the values that are present there. If the transmitted values match the ones stored before, the authentication is successfull. The transmission, especially of the password, in cleartext, however, bears a number of security risks (like which make basic authentication basically insecure[1] [18]. A brief security consideration of HTTP basic authentication is made in Section 3.6.1.

The second authentication method described by RFC 2617, HTTP digest access authentication, avoids to transfer the user password in cleartext. To achieve this, the digest authentication method applies MD5 cryptographic hashing combined with nonce values to prevent cryptoanalysis. Additionally to the authentication request, the authenticator sends the client a nonce value. Both, client and server, then perform the digest authentication computations shown in Figure 1.1 to calculate the digest response value. The value HA1 contains a MD5 hash of the username, the realm this credentials are valid for and the user password. The authenticator can choose to store the HA1 value in the hashed version opposite to storing the three input values in cleartext. This has the advantage that it saves the HA1 computation every time an authentication needs to be performed. Also, in case the authentication database is compromised, the attacker cannot get access

[1] Form-based authentication, however, also transmits the authentication credentials in cleartext (see below).

$$\text{HA1} = \text{MD5}\Big(\text{A1}\Big) = \text{MD5}\Big(\text{username} : \text{realm} : \text{password}\Big)$$
$$\text{HA2} = \text{MD5}\Big(\text{A2}\Big) = \text{MD5}\Big(\text{method} : \text{digestUri}\Big)$$
$$\text{request} - \text{digest} = \text{MD5}\Big(\text{HA1} : \text{nonce} : \text{nonceCount} : \text{clientNonce} : \text{qop} : \text{HA2}\Big)$$

Figure 1.1.: HTTP digest access authentication formulas

to the cleartext user password. On the other hand, the realm value cannot be changed after the hash value is calculated. The second part of the digest computation is the HA2 value which hashes the HTTP request method and URI. In the last step the HA1 and HA2 value are MD5 hashed using the nonce the authenticator provided and a number of values the client can choose to provide itself. Those additional attributes provide further security enhancements. They include a "Quality of Protection" (qop) parameter that specifies which of the security enhancements are required to be used, a nonce counter that is incremented by the client, and a client generated random nonce. These enhancements are designed to protect against cryptanalysis (e.g. chosen-plaintext attack) of the digest values [18]. A brief security consideration of HTTP digest authentication is made in Section 3.6.2.

Form-based authentication

Most commonly applications are communicating with the clients via (X)HTML documents. Those documents can contain forms which are used to exchange authentication credentials with the client. The credentials are transfered in cleartext using application specific (HTTP) variables and then processed by the application. The advantages of this approach is that the authentication procedure is completely customizable by the web application [52]. Especially the look of the forms and the number of authentication parameters are defined by the application. On the other hand there are no predefined methods or templates which means that the whole authentication procedure has to be implemented from scratch by the web application. Another disadvantage of form-based authentication is that the authentication credentials are not encrypted per se and are transmitted in cleartext. This means that extra steps have to be taken to secure the credentials, like using a secure protocol to transmit them. Also possible is to employ additional facilities like browser addons or client-side scripts to encrypt the credentials before they are transmitted [5].

1.2.3. Authorization

After a user has been successfully authenticated, the authenticator can apply security restrictions on that user. For example, one user may have access to administrative resources, while another one is only allowed access to the common pages. The process of granting or denying access to particular resources, based on the actual access privileges of an user (or an entity in general) is called authorization [31].

Access privileges are often grouped to make them more manageable. Such grouped privileges are called user groups or roles. An individual user is then assigned to one or more of such groups or roles. This authorization based on roles is called role-based access control [42] or role-based security[2]. In contrast to authentication, authorization does not imply any interaction with the user. The authorization decisions are made by the system in the background, usually unaware by the user. In general, only the authentication results, especially if access to a resource is denied, is communicated to the user. Although authorization is mostly applied to authenticated users which implies an earlier authentication, authorization can also be applied to unauthenticated (guest) users. Also, authorization is not a mandatory facility for systems. A system can just leave it at authenticating its users (which, however, could be understood as a kind of authorization, since the system obviously distinguishes between authenticated and unauthenticated users). Authorization and authentication, therefore, closely relate to each other but they are not mutually dependent.

1.3. Terminology

To ensure a common basis of terminology, a number of important terms used in the thesis are defined below.

Application provider An application provider, also called service provider, operates the web applications which are used by the end user. In terms of Diameter Web-Auth, the web applications implement the client side of the Diameter protocol (cp. Section 1.2.1).

Authentication Establish or confirm the identity of an end user (cp. Section 1.2.2).

Authorization Enforcing access restrictions on system resources, therefore, ensuring that only entitled user are granted access to that resource (cp. Section 1.2.3).

[2] Especially Microsoft uses this term in association with their .Net framework [32].

Basic authentication Confirming the identity of an end user by transmitting a user name and a user password from the user client to the web application server (cp. Section 1.2.2).

Digest authentication Confirming the identity of an end user by comparing one way hash values that are independently generated by the user client and the web application server based on a common nonce, the user name and the password (cp. Section 1.2.2).

Digital identity An electronic identifier used to represent an identity. Used by application providers to recognize an end user (cp. Section 1.2.1).

End user The person accessing web services offered by an application provider is called end user. He accesses the web services using his user client.

Identity attribute Additional characteristics to describe features of a digital identity (cp. Section 1.2.1).

Identity provider The identity provider offers identity services such as user authentication and authorization to its customers which usually are application providers. Common data related to an identity is usually stored on systems operated by the identity provider. In terms of Diameter WebAuth, the identity provider operates the Diameter servers responsible for Diameter WebAuth operations (cp. Section 1.2.1).

User client In web-based environments, the user client is usually a web browser. It is used by the end user to access web services.

Web application An application accessable via the World Wide Web. Also called web service and operated by an application provider.

1.4. Thesis organization

The thesis will be organized as follows. First, existing approaches in the field of identity management related to or applicable to web applications will be examined. An evaluation and comparison of the existing proposals in regards to their features and abilities will be conducted. Second, based on the results from Chapter 2, a novel proposal of network access protocol-based identity management will be developed. The proposal is designed to overcome shortcomings of existing solutions, close gaps that are not covered by them and be easily deployable in existing setups. In Chapter 4 the drawn up proposal will be

tested and verified in a prototype implementation, which includes a AAA client, a web application and the identity management server. The thesis concludes with a discussion of the worked out approach and an outlook and possible future steps for advancing the work.

2. Related work

This chapter introduces the work related to the thesis. Since there are a lot of projects and activities related to digital identity management, the presented work is only a selection. The introduced work represents the more present activities regarding identity management in web-based environments. Following the introduction is a comparison will be made of the presented approaches. The comparison will serve as a basis for the design of the identity management framework developed in the following chapters.

2.1. OpenID

OpenID is described as "an open, decentralized, free framework for user-centric digital identity" [41] which is built on top of existing Internet technologies such as HTTP, SSL and URIs. The basic idea of OpenID is that a person is identified by an URI[1] as personal identifier which he can prove to have controll over[2]. Such an OpenID account can be used to log into any site that supports OpenID logins. Development of OpenID was started by Brad Fitzpatrick of LiveJournal but is now being maintained by a community as open source software. The community gets financial and legal support by the OpenID Foundation [41]. Because a number of large organizations like AOL, Microsoft, Sun and Novell are providing OpenID support for its members, the OpenID community claims that there are over 160 million OpenID enabled URIs and nearly ten-thousand sites supporting OpenID logins [41]. The 2.0 version of the specification will among other things support the Yadis protocol [35] increasing its coverage even further.

OpenID only specifies how a web site finds and communicates with the identity provider responsible for a user. How and even if the identity provider actually identifies the user is outside the scope of the specification [47]. This makes the framework vulnerable for phishing attacks [7, 30, 53]. On the other hand however it allows OpenID providers to implement exceptional authentication mechanisms that are more secure than password-

[1] To be exact, OpenID in its current version only supports "http" and "https" URLs as identifiers [47]. For the next version (2.0) additional the support of "Extensible Resource Identifiers" (XRIs) is intended [54].

[2] More specifically, the person has controll over the data that is found at this URI.

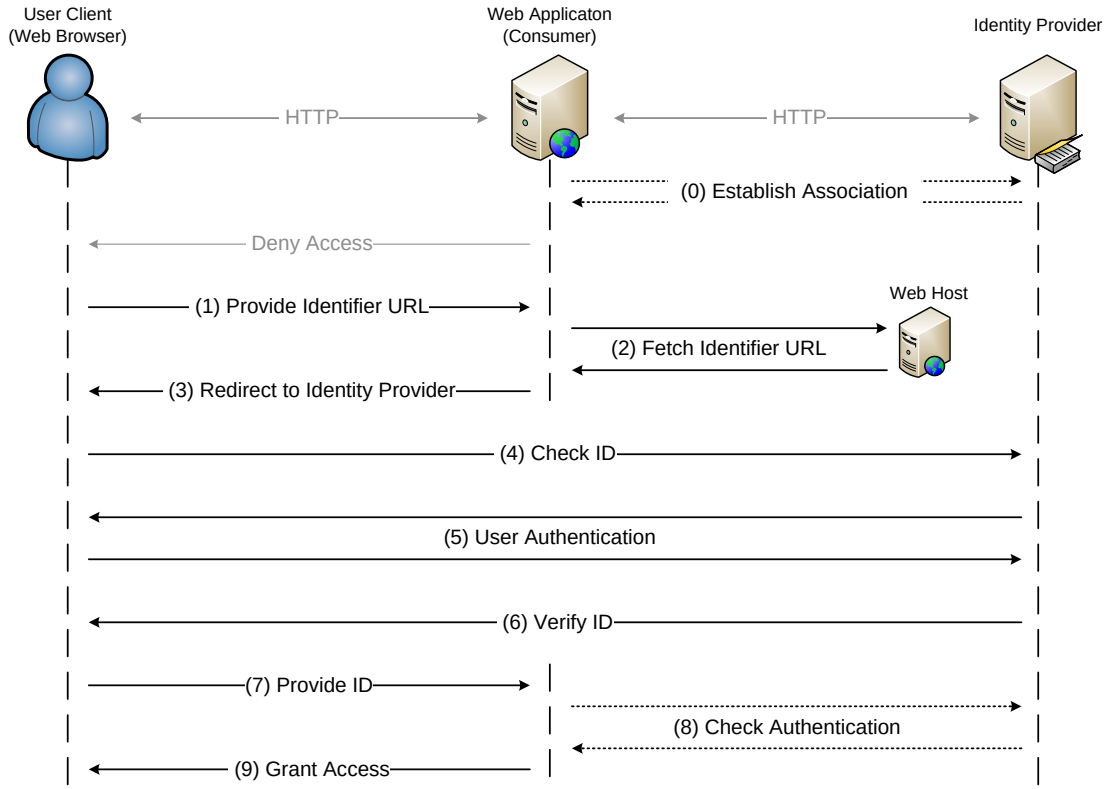


Figure 2.1.: OpenID login protocol flow

based logins like client certificates [21] or even more exotic approaches like image-based authentication [64].

The protocol flow for an OpenID login is shown in Figure 2.1. If a user tries to access a web page that requires him to login, the web application displays a HTML form, requesting the user to supply his OpenID identifier. The login process is started when the user submits his OpenID identifier, which is a simple URL (1). The web application, which is called consumer by the OpenID specification [47], retrieves the document at the indicated URL. The document is then parsed for information about the OpenID identity provider which is responsible for this URL (2). Now the web application knows which identity provider is responsible for the particular user and it responds with a HTTP redirect message to the original user request (3). The redirect message sends the user agent to the web server of the identity provider (4). The message also includes

parameters[3] that specify the web site the user should be redirected back to once the authentication is complete, the claimed identifier the user has submitted and the URL of the web application which requested the authentication. At this point the identity provider will authenticate the user using an arbitrary authentication method bidirectional with the user (5). If the authentication is successfull, the identity provider sends another HTTP redirect message to the user, delegating him back to the original web site (6). This results in a new request for the page the user wanted to access to begin with. Only this time it includes authentication information[4] from the OpenID identity provider that the web application can use to verify the identity of the inquiring user (7). After the application has verified the authentication information, it allows the user access to the requested resource (9).

The verification of user supplied authentication information can happen in two different ways. First, the web application uses a dedicated HTTP request to the identity provider to ask if the authentication message in question is valid (8). This option is called "dumb mode" by the OpenID specification. It has to occur in each authentication run but allows the consumer web application to be stateless in regard to the OpenID authentication process. The second option is for the web application to establish a so called "association" with the OpenID provider. This is done independently from authentication runs (0) and is used to negotiate a shared secret between the identity provider and the web application. Until the shared secret expires, it is used in subsequent authentication runs to sign and verify authentication messages. This option called "smart mode" allows the consumer web application to directly declare authentication identifiers supplied by the user in step (7) as valid. Its drawback is that the web application is required to maintain those shared secrets *individually for every identity provider* it wants to use this mode with.

2.2. The Liberty Alliance

The Liberty Alliance was formed in September 2001 as a business alliance with the goal of establishing an open standard for federated identity management. Currently over 160 profit, non-profit and government organizations are members of the alliance [45]. Their main goals are to

- "provide open standard and business guidelines for federated identity management spanning all network devices,

[3] The parameters are transmitted using HTTP GET variables within the target URL of the redirect message.

[4] The authentication information are included using HTTP GET variables again.

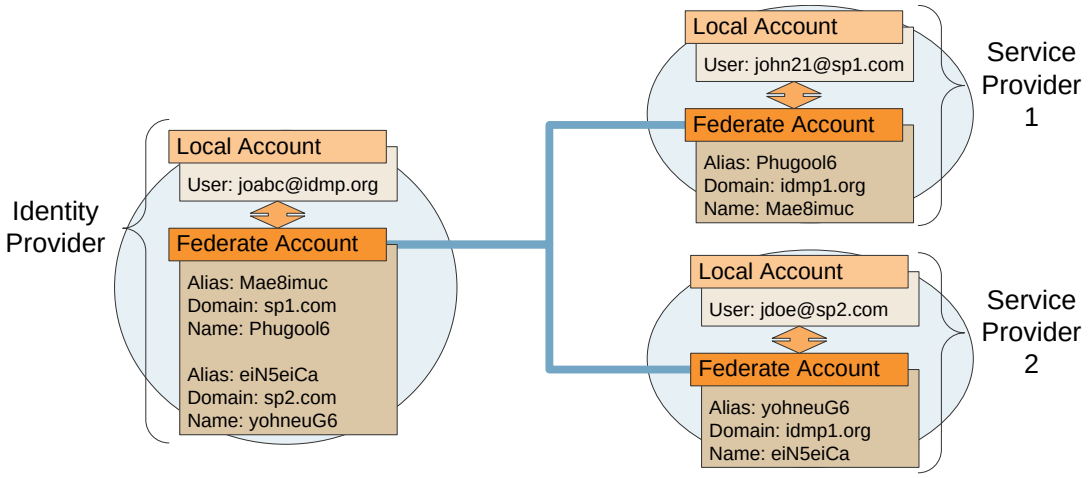


Figure 2.2.: Liberty Alliance - Federated Identity Management

- provide open and secure standard for SSO with decentralized authentication and open authorization, and
- allow consumers/businesses to maintain personal information more securely, and on their terms" [44].

"Federated identity management" is one of the key concepts of the Alliance and describes the cooperation of multiple partners, which each maintain different identities or different parts of identities. More descriptive this means that, for example, a user links multiple accounts at different service providers under a common account at his identity provider. Figure 2.2 illustrates this "federating" of accounts. The federated identity architecture provides the benefit of single sign-on[5] without requiring the user's data to be stored centrally [44].

Another key concept is the *"circle of trust"* which is "a federation of service providers and identity providers that have business relationships based on Liberty architecture and operational agreements and with whom users can transact business in a secure and apparently seamless environment" [44]. These trust relationships are an important attribute of the Liberty Alliance concept which distinguishes it from other decentralized authentication approaches. In OpenID, for example, no explicit trust exists between the service provider and the identity provider.

[5] In terms of the Liberty Alliance single sign-on is actually called simplified sign-on.

The Alliance has published a number of specifications about various aspects of digital identities. They are merge in three identity frameworks:

The Liberty Identity Federation Framework (ID-FF) describes the core identity management features like identity/account linkage, single sign-on, and simple session management.

The Liberty Identity Web Services Framework (ID-WSF) provides the specifications for building interoperable identity services, identity attribute sharing facilities, service description and discovery and associated security profiles.

The Liberty Identity Services Interface Specifications (ID-SIS) enable such interoperable identity services, for instance, personal identity profile service, alert service, calendar service and so on [44].

The Identity Federation Framework (ID-FF) which offers an approach for a single sign-on and single logout solution with digital identities [11] is based on the Security Assertion Markup Language (SAML). The current version 1.2 of the framework has been submitted back into the OASIS Security Services Technical Committee as input for the next version of SAML.

2.2.1. Security Assertion Markup Language (SAML)

The Security Assertion Markup Language (SAML) is a "XML-based framework for communicating user authentication, entitlement, and attribute information". Its purpose is to "allow business entities to make assertions regarding the identity, attributes, and entitlements of a subject to other entities" [38]. SAML is maintained and developed by the OASIS Security Services Technical Committee since January 2001. The current version SAML V2.0 was approved as an OASIS Standard in March 2005. In the beginning the major goals of SAML were to enable single sign-on for web users and to specify the "exchange of authentication and authorization information in a variety of kinds of distributed transaction" [38].

Figure 2.3 shows an example of a SAML assertion which, for example, could have been passed from an identity provider to a service provider. It is taken from the SAML V2.0 technical overview [46] and is an assertion that could have been passed from an identity provider to a web application. It describes a SAML 2.0 assertion that was issued by the site "http://www.example.com" (lines 2 to 6). The site declares that the subject "j.doe@example.com" identified by his email address (lines 7 to 12) was authenticated using a password-protected transport mechanism[6] (lines 7 to 24). If the service provider

[6]For example, the user provided a user name and a password during a SSL-protected browser session.

```
 1: <saml:Assertion xmlns:saml=urn:oasis:names:tc:SAML:2.0:assertion
 2:   Version="2.0"
 3:   IssueInstant="2005-01-31T12:00:00Z">
 4:   <saml:Issuer Format=urn:oasis:names:SAML:2.0:nameid-format:entity>
 5:     http://www.example.com
 6:   </saml:Issuer>
 7:   <saml:Subject>
 8:     <saml:NameID
 9:       Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
10:         j.doe@example.com
11:     </saml:NameID>
12:   </saml:Subject>
13:   <saml:Conditions
14:     NotBefore="2005-01-31T12:00:00Z"
15:     NotOnOrAfter="2005-01-31T12:10:00Z">
16:   </saml:Conditions>
17:   <saml:AuthnStatement
18:     AuthnInstant="2005-01-31T12:00:00Z" SessionIndex="67775277772">
19:     <saml:AuthnContext>
20:       <saml:AuthnContextClassRef>
21:         urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
22:       </saml:AuthnContextClassRef>
23:     </saml:AuthnContext>
24:   </saml:AuthnStatement>
25: </saml:Assertion>
```

Figure 2.3.: SAML assertion with subject, conditions, and authentication statement

is satisfied with this assertion, it can now grant the user access and identify him as "j.doe@example.com" without further inquiry.

2.3. Windows CardSpace

The Windows CardSpace technology was developed by Microsoft to implement an identity selector for the Microsoft Windows operating systems. An identity selector is a mechanism to choose an identity (here an "information card") which can be used to identify the user to a web site or a web service [33]. The identity data associated with these identity cards are self-generated or issued by an identity provider such as a bank, an employer or the government. In order to identify the user, the identity card contains

security tokens which are digitally signed and encrypted by the identity provider that created the card. However, ultimately the user itself makes the decision, which data to release to an inquiring web site [33].

Windows Vista is the first Microsoft operating system with build-in support for the Windows CardSpace technology. However Windows CardSpace is part of the .NET Framework 3.0 and therefore is also available for Windows XP and Windows Server 2003. Since the specifications are made available by Microsoft, it is possible for third parties to implement CardSpace-compatible identity selectors on other platforms and devices. Identity providers and relying parties for CardSpace are not limited to Microsoft operating systems to begin with [34]. There are a number of projects which are implementing different aspects of the Microsoft Windows CardSpace technology like xmldap.org [65], Bandit [4] or the Higgins project [57].

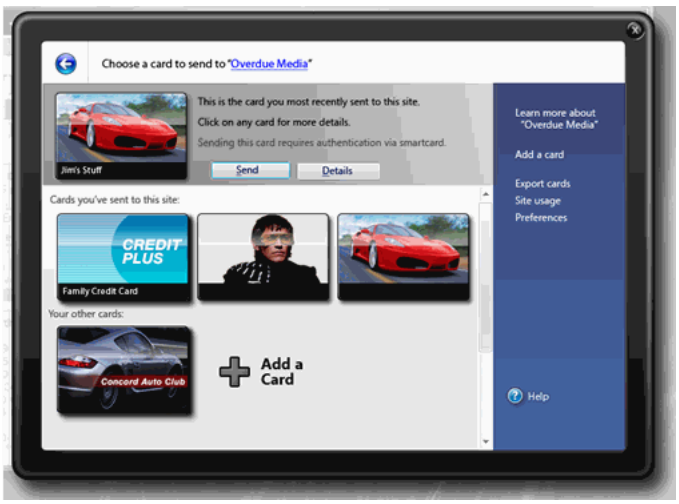
CardSpace is a so called user-controlled identity management system [23]. The user controls which of his personal data he wants to expose to the requesting party. It also offers pseudonyms and aliases in form of multiple cards. An example what the identity selection screen could look like for a user is shown in Figure 2.4. It is noteable that the authentication technology that is present behind a card is not apparent to the user. In fact, it typically isn't even apparent to the CardSpace system itself. The system is entirely agnostic about the format of the security token and, therefore, can support any digital identity system, using any type of security token. This includes simple usernames, X.509 certificates, Kerberos tickets, SAML tokens, or anything else [34].

The CardSpace authentication process is shown in Figure 2.5. First, the CardSpace system requests the relying party's[7] security token requirements (1). Those requirements include a policy which describes the requirements for security tokens the site is willing to accept. The system then asks the user to choose from the cards that are considered based on the policy. Once the user chose a card (2), the CardSpace system request a token from the appropriate identity provider (3). The received token then is used to authenticate the user to the application.

2.4. Kerberos

Kerberos is a network security protocol originally developed by the Massachusetts Institute of Technology (MIT). Meanwhile it is maintained and advanced by the Kerberos Working Group of the Internet Engineering Task Force (IETF). The latest version of Kerberos (V5) is specified in RFC 4120 [37]. Kerberos implements a ticket approach

[7] CardSpace defines the relying party as "an application that in some way relies on a digital identity." For the purpose of this thesis, the relying party is assumed to be a web application.



(Source: Windows Vista Technical Articles - Introducing Windows CardSpace [34])

Figure 2.4.: CardSpace identity selection screen

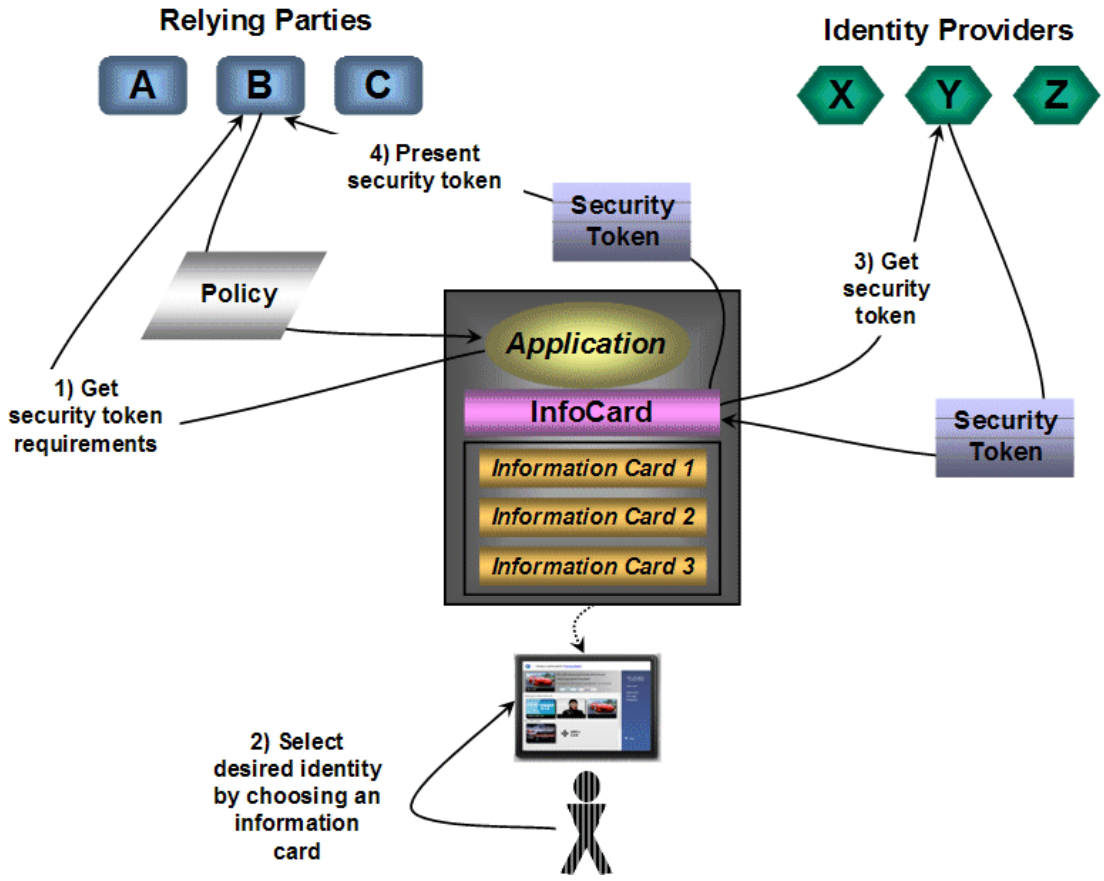
to network authentication. After a successfull authentication against the Authentication Service, the Kerberos client receives a Ticket-Granting-Ticket. Using this ticket it can request service specific tickets from the Ticket-Granting Server. The service specific tickets then can be used to access the particular service. Using this ticket approach, Kerberos supports single sign-on functionalities for all supporting services [37].

Since Kerberos needs to be supported by client and server, it is not present in common web environments. There is some support for Kerberos as a web server authentication back-end though [28] and a couple of projects to support Kerberos authentication in browsers [20, 29]. In september 2007 the MIT announced the launch of the Kerberos Consortium. With prominent founding sponsors such as Google, Stanford University, Sun Microsystems and the University of Michigan its goal is to advance the propagation of Kerberos. Their plans also include web authentication [59].

2.5. Diameter

The Diameter protocol is an authentication, authorization and accounting (AAA) network security protocol corresponding to the "Criteria for Evaluating AAA Protocols for Network Access" [1]. It is the intended successor for the Radius protocol [50] which is

.....



(Source: Windows Vista Technical Articles - Introducing Windows CardSpace [34])

Figure 2.5.: CardSpace authentication process

also an AAA protocol. RFC 3588 [8] defines the Diameter Base Protocol which needs to be extended by specific Diameter applications in order to be applicable in the particular usage scenario it is deployed. For example, there exist Diameter applications for Network Access Servers [10], Mobile IPv4 [9], Credit-Control [22] and the Session Initiation Protocol [19]. A Diameter application, therefore, defines an own protocol-based on the Diameter base protocol rather than being a software application in the common sense. A Diameter client and server both need to implement the same Diameter protocol in order to be able to perform AAA tasks against each other.

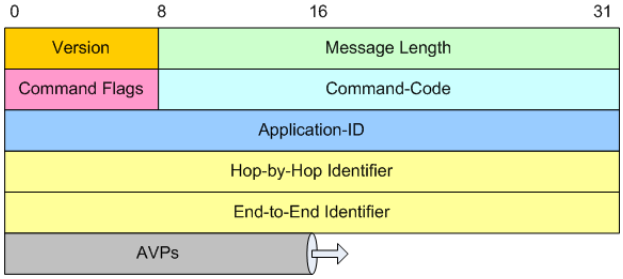
2.5.1. The Diameter base protocol

The core of a Diameter implementation is the Diameter base protocol. It provides facilities for:

- Delivery of AVPs (attribute value pairs),
- Capabilities negotiation,
- Error notification,
- Extensibility, through addition of new commands and AVPs,
- Basic services necessary for applications, such as handling of user sessions or accounting

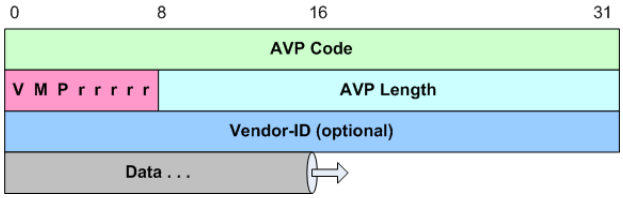
and is the basis for extending the Diameter protocol through Diameter applications [8]. The base protocol on its own only implements accounting mechanisms, any other functionality (especially authentication and authorization) needs to implemented by such a special application.

The base protocol defines information to be transmitted by the exchange of attribute value pairs (AVPs). Every Diameter message consists of some header information and a number of such AVPs. Nearly every infomation in a Diameter message exchange is encoded in an AVP. The only exceptions are the Diameter header information like the application ID or the command code of the current Diameter message. 2.6 shows the format of a Diameter packet and 2.7 that of an AVP. The intended function of a message exchange is indicated by the command code which is a numerical value in the message header. The base protocol for example defines the command code 274 to indicate an Abort-Session-Request or an Abort-Session-Answer depending on whether the current message is a request or an answer [8]. Diameter commands define which AVPs are mandatory to be included in the message exchange an how the receiving side is supposed to process the message.



(Source: http://en.wikipedia.org/wiki/DIAMETER)

Figure 2.6.: Diameter packet format



(Source: http://en.wikipedia.org/wiki/DIAMETER)

Figure 2.7.: Diameter AVP format

2.5.2. Extending Diameter

Extensibility is a design characteristic of the Diameter protocol. The base protocol lists a number of mechanisms to extend the Diameter protocol [8]:

- Defining new AVP values.
- Creating new AVPs.
- Creating new authentication/authorization applications.
- Creating new accounting applications.
- Application authentication procedures.

It is common to combine all or multiple of the mechanisms above in order to create a new Diameter application. Although the reuse of existing AVP values, AVPs and even Diameter applications is strongly recommended by the base protocol specification, new Diameter applications usually specify multiple new AVP values, AVPs and Diameter commands. It is the function of the Diameter Maintenance and Extensions (DiME) working group of the Internet Engineering Task Force (IETF) [58] to keep maintain and extend the Diameter protocol. They also maintain the Diameter Applications Design Guidelines [16] which contains guidelines and suggestions on how to define a new Diameter application.

2.6. Comparison

After the related work has been introduced, the existing approaches will be evaluated and compared. A number of criteria will be used for the evaluation which are oriented at the web-based environment. The criteria selection is mainly inspired by the features promoted by the works which were introduced above.

End user authentication Identity management solutions do not necessarily specify the authentication of the end user. It may seem strange at first, that an approach which centers around digital identities doesn't address the question of authenticating its users. After all a vital aspect of an identity management system is to provide its clients with assurances about the identity of a suspect. However, a number of identity solutions consider the exact protocol used to authenticate a user outside the scope of their specifications. They focus on providing the participant with assertions about existing trust relationships. The actual authentication task is delegated to the systems that subsequently are making those assertions.

For example in OpenID the client requesting an OpenID authentication will only get an assertion from the identity provider of the particular OpenID identity that the user belongs to him and is authenticated. What kind of authentication was used in order to get to that assertion is outside the scope of the specification.

On the one hand this gives the authenticating parties (usually the identity management providers) the freedom to implement secure, reliable and even exotic authentication methods. On the other hand those systems integrity usually is bound to the strength or weakness of the system used to authenticate the end user. Especially in identity management systems which provide single sign-on this can have severe consequences because an attacker just needs to perfrom a single successfull authentication to get access to the whole system.

When looking at the end user authentication mechanisms a systems specifies, the extensibility of those mechanisms should also be evaluated. The more flexible a system is, the more resilient it is against security risks that may arise at a later point.

Authorization Having different roles for users allows a more granulated access control. For example, a web site may want to distinguish between normal users and administrative users. The difference to the authentication process is that both of the user groups have legit identities and they both pass authentication. The site just wants to apply different access privileges depending on a certain status of the particular user. Identity management systems that support authorization allow for a finer control over access privileges on different levels rather than in an either all or none at all fashion.

Single sign-on/sign-out Providing single sign-on (SSO) for the end user can be considered a perk or one of the main reasons that advance identity management from the user's perspective. First and foremost, SSO is a convenience. The user still needs to login and he does that with the same mechanism with or without SSO. He just doesn't need to do it that often anymore.

Different levels of single-sign on can be distinguished based on its automation. Fully automated SSO system detect the login status of a user without his help. The services accessed constantly adjust the user's status to the system. Only partially automated systems require some input from the user in order to be able to determine his status like an identifier or a prompt to recheck the user's status. This can be due to limitations to the system or because of intentionally forced restrictions for performance, security or privacy reasons. In the first case for example the system may need the user to supply a hint about his identity (like an OpenID identifier) in order to be able to verify his login status. In the later case the system may be designed to wait for the user to push

a button before the systems tries to assess his status. This way the system doesn't have to constantly query for a status update and also the user can still control whether the system identifies him or not.

Systems that offer a single sign-on for its users usually also offer single sign-out[8]. One can argue, that single sign-out is a system immanent feature of SSO systems since it is something like the "not (single) signed-on" state. Therefore, a system that allows a client to have a system-wide signed-on state, inevitable performs a single sign-out action when it revokes that state. However, just like the with the SSO functions, the degree of automation has to be considered. A user's authentication status either can be pushed from the system to the clients[9] or needs to be pulled by the clients from the system. If the push or pull operation is only performed when a user signs-on, a single sign-out cannot work. For example, if a client in the pull modell doesn't update the authentication status for its users after they signed-on, it will not notice the status change when the user signs-out. Therefore a single sign-out does need to be specially considered in the design of an identity management system if it should be supported.

Accounting At first sight, identity management and accounting does not go necessarily hand in hand. Accounting was introduced in AAA systems in order to keep track of the resource or service consumption an authorization decision entails [15]. Among other things, the accounting records can be used to charge the end user for the services he has enlisted or to evaluate the quality of service provided by the system. Within the scope of web-based identity management, accounting mostly makes sense under monetary aspects. For example, if the identity management provider also handles incurring charges for its user.

User client support For the deployment of an identity management system it makes a big difference whether an implementation on the end user client is required. In case of web-based identity management systems the end user client is usually the web browser. If a system requires special support from the user client which needs to be implemented, this has a negative impact on its deployability. I case a user client implementation is required, the extend and complexity of the modifications can be used to further distinguish between the approaches.

Technologies The effort it takes to implement a system specification depends to a great deal on the amount of different technologies and on the question how established

[8] Also called single sign-off [6] or single logout [11, 46].
[9] In this case the web application.

those technologies are. The more different technologies are used, the more knowledge and consideration the implementation requires. In the same manner, more established technologies are often easier to implement than comparatively new ones. This arises from the assumption that more knowledge, references and experts exist for the more established technologies.

On the other hand, exactly the opposite can be the case. A new technology offers solutions for problems that were solved very cumbersomely with other technologies in the past. Or the new technology is particularly easy to implement. In the end, the technologies a system employs need to be properly evaluated on their own to come to a conclusion. However, they do reflect back on the system and permit some reasoning about the approach.

Identity attributes A digital identity can be no more than a simple alphanumeric identifier [2]. If the identity is connected to a person, however, there are a multitude of other attributes that describe that person. For example, a first and a last name, a birthdate or an email address. Identity management systems can be differentiate whether they also provide means to transport such additional identity attributes.

Maturity level Especially for security related protocols and systems a high maturity level is an important attribute. The more work has been invested in a subject, the better it should be understood. Problems and security risks are more likely to be found and to be resolved in highly mature software or specifications. A high maturity grade also reflects how well an approach is established. Maturity criteria can be, for example, the number of working implementations, the number of revisions a specification had or the level of standardization. The more mature a a specification is, the less likely it is to change. Usually, mature approaches are more stable and more reliable.

Primary focus The primarily intended area of application for a system is usually reflected by its design. On the hand, this makes the system better suited for its purpose, on the other it also restricts its range of application. This isn't necessarily a bad thing. A very specific approach can take advantages of circumstances, that a more general approach cannot without loosing exactly this generality. However, a more general approach maybe able to generate synergy effects because it can carry out the same tasks that would otherwise be handled by a number of different systems. Therefore the evaluation will also consider if the evaluated work focuses on web technologies. In the end, the trade-off between specialization and generalization has to be justified by the advantages and disadvantages it introduces.

2.6.1. Results

Table 2.1 provides a short overview of the results of the evaluation. First of all, it is noteable that every approach has a slightly different focus. Although they all deal with digital identities their intended use, purpose and goals do vary. Liberty and OpenID seem closer together in their goals than the other approaches. A closer examination, however, shows different motivations stand behind both approaches. OpenID aims at providing a convenient method for a user to create a digital identity and use it with as many web sites as possible. The OpenID approach is user-centric, distributed and does not include trust as an important concept. Liberty on the other hand focusses precisely on such trust relationships. The project is centered around assertions between web sites about their users. The Liberty specifications therefore also cover single sign-out and differentiated user authorization. However, compared to the OpenID specifications, they are also much more comprehensive and complex.

The Microsoft CardSpace technology is also considered a user-centric approach and evolves around different identities a user owns and how he uses them to identify himself. OpenID and Liberty don't cover the actual authentication process between a user and a web site, while CardSpace is all about this process. CardSpace describes a framework to create or obtain digital identities including related identity attributes. It leaves the user in control to use the identity he sees fit to identify himself to web sites. Because of its different focus, CardSpace is more a complementary technology to Liberty and OpenID than a competing one.

Kerberos and Diameter are both rather mature protocols (compared to the other technologies). They were designed from a network point of view without any focus on web technologies. They are, however, intended for very similar purposes than the other identity management approaches. They are designed to authenticate and authorize users[10] and allow them access to (network) services and resources. Kerberos requires the user client to support the protocol, while Diameter allows a service specific protocol between the user client and the access gateway.

Besides those different focusses and origins of the discussed approaches, the results also show that they all provide common features for identity management. They provide means for applications and users to handle digital identities for identification and authorization purposes. The Liberty Alliance project, OpenID and Microsoft CardSpace are web-based approaches that emerged rather recently while Kerberos and Diameter are network-based approaches which have been established for quite some time. Although they are all approaches to digital identity management, their objectives do not necessar-

[10] Diameter is actually not limited to users but can also be used to authenticate and authorize devices in general.

Table 2.1.: Feature comparison for identity management systems

Feature	OpenID	Liberty	CardSpace	Kerberos	Diameter[1]
Authentication	no	no	yes	yes	yes[2]
Authorization	no	yes	yes	yes	yes
Single sign-on/ sign-out	yes/ no	yes/ yes	no[3]/ no	yes/ no	no/ no
Accounting	no	no	no	no	yes
Requires user client support	no	no	yes	yes	no[4]
Technologies	HTTP	SAML	XML	IP	IP
Identity attributes	no[5]	yes[6]	yes	no	no
Maturity	medium	medium	low	high	high
Primary focus	Decentralized identities (Web)	Trust relations (Web)	Authentication (Web)	Service authorization	Network access

[1] Diameter Base Protocol [8].
[2] Authentication needs to be implemented by a Diameter application.
[3] CardSpace can be part of a SSO system though.
[4] Diameter usually uses service specific protocols between user client and Diameter client.
[5] A proposed draft is available for identity attributes in OpenID [25].
[6] Not part of the Identity Federation Framework (ID-FF) but the Identity Service Interface Specifications (ID-SIS) [27].

ily coincide. While the Liberty Alliance provides a highly comprehensive approach to federated identity management, OpenID aims at a simple approach to decentralized user identities. Windows CardSpace provides an alternative to common authentication methods centered around user managed digital identities. Kerberos and Diameter, finally, are classical network access protocols, however, it can be concluded, that they are well suited for identity management purposes as well.

2.7. Summary

This chapter has introduced the related work to this thesis and evaluated existing approaches to identity management.

- Three web-based identity management approaches have been introduced: OpenID, the Liberty Alliance project and Microsoft CardSpace.

- Two network-based authorization and authentication protocols have been introduced: the Kerberos protocol and the Diameter protocol.
- A comparison of those different approaches shows that all approaches, despite different focusses, provide basic identity management functions.
- While the web-based approaches are rather recent, the network-based approaches are well established and proven.
- It can be assumed, that if Kerberos or Diameter can be adapted to a web-based environment, they would be valid options as basis for an identity management system for web applications.
- Diameter seems to be more suitable for such an adaptation since it doesn't require support in the end use client.

3. Design

Based on the evaluation in the previous chapter, a new proposal for an identity management framework for web applications will be developed. This chapter describes the design of the framework. As shown by the evaluation in Section 2.6 the Diameter protocol already provides a number of features that are commonly required by identity management solutions. It is also designed to be very well extendable by the means of Diameter applications. Basis for the framework, therefore, will be the Diameter protocol which will be extended by the design of a "Diameter Application for AAA and Identity Support in Web Applications", called Diameter WebAuth.

The Diameter Base Protocol itself is specified in RFC 3855 [8] and Diameter applications are usually specified in internet drafts which in some cases manage to become RFCs themselves. The Diameter Maintenance and Extensions working group [58] attends those application drafts. The group also published the Diameter Applications Design Guidelines [16]. The design process in this chapter will follow the design guidelines as well as the recommendations by the RFC Editor concerning factual and technical issues[1] [49, 60, 61].

3.1. Introduction

This chapter describes the Diameter Application for AAA and Identity Support in Web Applications (Diameter WebAuth). The intended area of application for Diameter WebAuth are web applications that want to utilize a Diameter server for authentication, authorization and accounting of their users. It implements means to supply a web server with data to authenticate its user via common HTTP authentication methods and to authorize and account the access to resources or services provided by the web server. It also supports basic commands for the transmission of further identity information about authenticated users in different identity information schemes.

Diameter WebAuth is meant to be applicable in scenarios like Identity Management Frameworks where there are different trust relationships between the user, the Diameter

[1] Specifications regarding the format and formal standards have been widely ignored for the sake of a consistent and appropriate layout of this thesis.

client and the Diameter server. For example this means that no re-usable authentication credentials are shared with the Diameter client and that the Diameter server can hold back authentication or authorization information until they are actually needed by the Diameter client. See Section 3.5 for privacy considerations.

The Diameter WebAuth does not rely on any other Diameter applications and can be employed to provide as a lightweight, stand-alone AAA client or server. In appropriate environments it can be used as a replacement for more complex authentication applications like the Diameter Network Access Server Application [10], especially if there are complexity restraints like in embedded systems. Nevertheless it is interoperable with other Diameter applications like the Diameter credit-control application [22] to extend its capabilities.

3.1.1. Motivation and goals

There are Diameter applications available for a wide range of services, like network access ([10]), Mobile IP [9] or the Session Initiation Protocol [19]. The existing applications however are not particularly suited for the deployment inside of web applications, although a multitude of current web applications require authentication and authorization. They are either very extensive and complex to implement or do not offer methods suitable for authentication, authorization and accounting within a web-based environment. Therefore web applications (or web servers itself for that matter) implement local or proprietary authentication back-ends and databases or use services that are not primarily designed for external AAA operations like LDAP [66] servers, database servers or even IMAP [13] servers. Even though there might be a AAA service like Diameter available within their administrative domain. The objective of this chapter is to specify a Diameter application that does allows web servers and web applications to employ existing AAA structures for authentication, authorization and accounting.

Because web applications usually offer services to a human end-user, they regularly need further information about their user's identity or preferences to personalize the offered services. While a number of those attributes are highly service specific, there are others which are closer associated with the user than with the service. Examples are the user's first and last name, his gender, age, contact or billing information. Instead of having those information stored in a number of application specific databases for every different service the user accesses within the AAA domain, it makes sense to deposit them together with the user's authentication and authorization credentials. In addition to methods for authenticating, authorizing and account its users, this specification therefore defines means to enable web applications to query Diameter servers for additional identity information. This allows the web application to efficiently handle all its information

demands within the Diameter protocol.

This specification is written with scenarios in mind, where Diameter server and Diameter client are not part of the same administrative domain. For example this is the case when the end-user signed up with a dedicated identity management provider which operates a Diameter server infrastructure to provide authentication services to web application providers. In these three-party scenarios, the end- user has a profound trust relationship with the identity management provider but not with the service he us accessing. Therefore special attention has to be paid to secure the privacy of the end-user against the application service provider while enabling the service provider to render its service.

Recapitulatory, the goals for this Diameter application are, to be:

Lightweight and easily implementable as client in web servers, web applications and other devices which utilize web-based user interfaces as well as a Diameter server implementation.

Secure and private to be feasible for scenarios where the Diameter server and the Diameter client are not part of the same administrative Domain, like third party identity management provider services.

General in regards to identity information, to be able to transport and manage a wide range of identity information data.

3.1.2. Use cases

This section describes a number of typical use cases that this specification is intended to cover.

A web site wants to authenticate a user

A user accessing a web site needs to be authenticated in order to link him to some identity. This can be necessary for example if a returning visitor ought to be matched to his profile or if access to the site is limited to registered users only. Authentication is usually also necessary to establish the identity of a user in order to perform advanced tasks covered by this specification since they all center around a known user.

A web site wants to authorize a user for a specific service

A resource that is requested by a user requires special access privileges. The web site needs to authorize the user for this resource before allowing him access. It is possible

for the web site to maintain different areas with different access requirements so that authorization needs to be repeated for different services and can yield different results.

A web site wants to charge a user for a specific service

If a user accesses a service that requires some sort of payment, the web site can charge the user using this Diameter application. However, no credit reservations or assurances for future credit operations are made. It is up to the web site to make sure to perform the credit charge operation and wait for a positive response before granting access to the service.

A web site wants to credit a user for a specific service

Besides charging a user, a web site can also credit or refund a user. This mechanism can be used to counteract a charge that was made to the client earlier or to reflect a deposit the user has made. The possibility to charge a user for a discount and later refund credits that were not used can be used as a crude kind of credit reservation mechanism. It is important to note that no service-specific accounts are maintained. Every web site using the Diameter server for credit control is accessing the same account. Therefore web site or service specific credit lines cannot be maintained.

A web site wants to check if a user has a certain credit

Sometimes before actually charging a user, a web site is only interested if a user has a certain amount of credit available. For example to decide if a user is granted access to a certain premium area for prosperous customers or to filter offers based on the available credit. Or the web site wants to warn a customer in the beginning of a tedious operation if he may not have sufficient credit in the end. However, checking if a user has a certain credit cannot be used to ensure a charging operation. Only the result of an actually credit charge operation determines the amount of credit that has been deducted from the user's account.

A web site wants to retrieve user specific identity data

Authenticating a user only yields only a username which often is just some kind of handle or alias. If a web site wants to retrieve additional information about the user it can query the Diameter server for those identity information. This can be attributes like the first and last name, a title, address or banking account. If the Diameter server can answer the query, it will retrieve the information and return it to the web site. It is possible (and

desireable) for a server side implementation of this Diameter application to allow the end user some control over which of his personal data is accessable by the web site. This may lead to a negative response from the Diameter server even if the data is available. Such mechanisms are outside of the scope of this specification and depend on the respective implementation. However, the web site will get an answer to its request, indicating that access to the data is denied, This allows the web site for example to display a message asking the user to check for potential access restrictions on his data.

A web site wants to store user specific identity data

Besides retrieving identity data, a web site can also request a Diameter server to store user specific identity data. However, the Diameter server implementation might impose certain restrictions on this operation, similar to the use case above where data is retrieved. Although it maybe possible for implementation to arrange for site specific data storage or some sort of private namespace for web sites, this is usually against the design of a centralized data storage. Therefore web sites must account for the possibility that other sites have access to and can modify the data stored on the Diameter server. The intended data to be stored on a Diameter server is data that is only linked to the user but not to the web site. Web site specific data should still be stored locally.

3.2. Overview

Goal of this specification is to support identity management functions in web applications using the Diameter protocol. This includes identifying users, verify their access privileges and transport personal attributes that describe them. Furthermore this specification will provide basic means to charge users for accessed services. In terms of the Diameter protocol, Diameter WebAuth provides a web server with the means to utilize an AAA infrastructure to authenticate, authorize and account its users for the access to its resources and services. Furthermore it provides methods to query the Diameter server for additional information about the user's identity. The following sections detail these functions.

3.2.1. Authentication and authorization

Identifying a user and verifying his access privileges is probably the most vial part of identity management. Diameter WebAuth extends the facilities provided by the Diameter Base Protocol for a Diameter client to authenticate and authorize a user for this

purpose. Two requirements are to be kept regarding the authentication. First, Diameter WebAuth must use standard authentication methods that are supported by the user client. The reason for that is, that Diameter WebAuth only specifies the protocol between the Diameter client and the Diameter server. It cannot alter or adjust the service specific protocol between the user client and the Diameter client, HTTP in this case. The second requirement is that the authentication method needs to provide protection against unauthorized access to secret credentials. In case of username/password authentication this would be the password. Particularly this means that in scenarios where the Diameter client is outside the trust domain of the Diameter server, the secret credentials needs to be protected against the Diameter client itself.

The most common authentication method supported by web browsers is username/-password authentication. RFC 2617 [18] specifies two HTTP authentication methods which are widely supported by web browsers: Basic Authentication and Digest Access Authentication. While the basic authentication exchanges the credentials including the password in cleartext, the digest access authentication uses a one-way hash function to prevent sending the password in cleartext. Although the digest authentication is not intended to be an absolutely secure authentication scheme[2] it serves the purpose of protecting the user password against snooping by any entity between the user client and the authenticator[3]. Besides HTTP digest access authentication, the Diameter WebAuth specification will, nevertheless, support basic authentication as well. It can be used as a fall back in environments where digest authentication is not available or not necessary and to more generally support different authentication mechanisms. For example, HTML-form-based authentication.

Authorization should support different roles and access levels. To implement this, first a standard has to be defined that describes the different access levels. Because the Diameter WebAuth design already uses parts of the Diameter credit-control application (cp. Section 3.2.2), it will also use its way of specifying services. The service identifiers can be used by web applications to request user differentiated authorization. The Diameter credit-control application introduces service description based on a service context identifier combined with a service identifier. While the service context identifier is used to describe the service specific document that applies to the request, the service identifier designates the exact service within that document (cp. [22, Sections 4.1.2., 8.28., 8.42.]). More descriptive this means, that, for example, the service "access to administrative functions" is identified by the service identifier "3" within the service context identifier

[2] HTTP digest authentication offers no confidentiality protection beyond protecting the actual password and is vulnerable to certain attacks (cp. [18, Section 4] and Section 3.6).

[3] In this case the Diameter server.

"service-manifest-01@sp1.example.com."

Operation details

The authentication and authorization procedure starts when a user tries to access a resource on the web server that is subject to authorization. Diameter WebAuth supports the HTTP Basic and Digest Access Authentication scheme [18]. This also includes derivative methods such as user name/password authentication via HTML forms. The Diameter client can include User-Name and User-Password attributes in an AA-Request command to request basic authentication. If an authentication request does not include a User-Password attribute, digest authentication is performed and the Diameter server MUST respond with an AA-Answer that includes a HTTP-Digest-Challenge.

The Diameter client can choose to initiate a HTTP Digest Authentication with its user client prior to issuing a AA-Request by generating the values for the HTTP Authorization Request Header by itself. It then uses its user clients response to construct a HTTP- Digest-Response attribute and includes it in the initial AA-Request. This is called "HTTP digest quick mode." It effectively saves one Diameter message exchange and relieves the Diameter sever form the necessity to maintain a state for the client during a MULTI_ROUND authentication. If the HTTP-Digest-Response attribute cannot be verified by the Diameter server or it does not accept requests with Diameter client generated contributions the Diameter server MUST discard the HTTP- Digest-Response attribute and continue the authentication process as if the client had not sent the HTTP-Digest-Response. If the Diameter server could verify the digest response but does not accept Diameter client generated digest challenges in its authorization process, it MAY include a Digest-Stale attribute set to "true" (without the surrounding quotes) in the HTTP-Digest-Challenge to indicate to the Diameter client that the authentication was otherwise successful but needs to repeated with the Diameter server generated challenge.

Another possibility to reduce protocol round trips and to mitigate load on the Diameter server is to delegate the final authentication check to the Diameter client as described in RFC 4740, Section 6.3. [19]. Because the user name is required in order to lookup the password, this is only viable if the Diameter client already has an idea about the identity of its client and can include a User-Name AVP into the AA-Request. In web environments, for instance, a web application can try to discover a recurring user's identity by using cookies to save a corresponding hint. However, if a Diameter client was mistaken about a user's identity and included a false User-Name in an initial AA-Request, it MUST terminate the Diameter session used for the initial request and start a new session for the authentication using the correct user name.

Figure 3.1 shows an example for Diameter WebAuth using digest authentication. If a

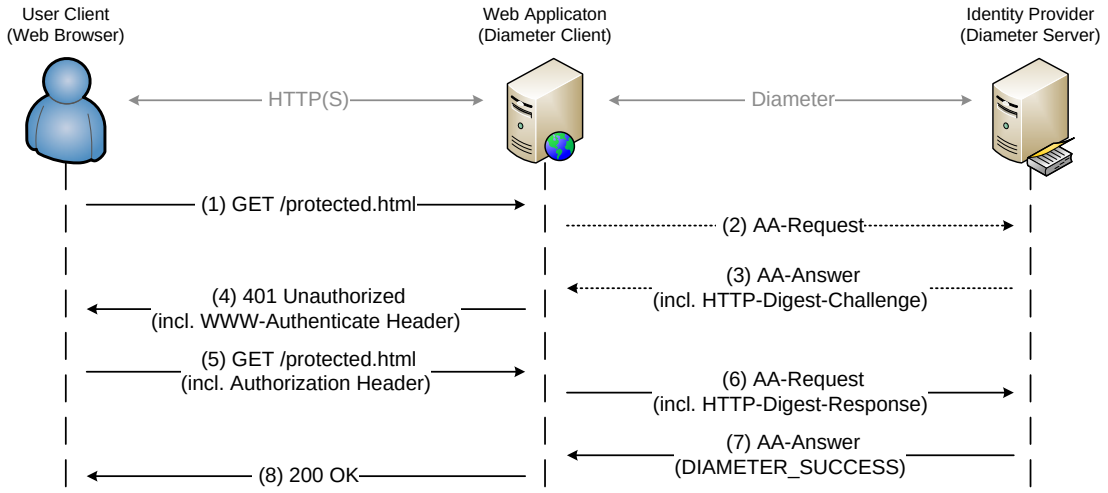


Figure 3.1.: Diameter WebAuth using HTTP digest authentication

user client initially sends a request for a resource without including any credentials (1), the Diameter client starts the authentication and authorization request. The Diameter client sends a, AA-Request to the Diameter server (2) which replies with an AA-Answer (3). The AA-Answer includes data usable by the web server to compose a response to the user clients request (4), challenging an authentication. If the Diameter client uses the HTTP digest quick mode, the message exchange in steps (2) and (3) is omitted and the Diameter client generates the data required for the authentication challenge in (4) itself. Next, the client assembles his authentication credentials and sends another request to the web server (5) including the user clients credentials. The Diameter client assembles an AA-Request to the Diameter server with the corresponding information from the clients request (6). If the credentials match the records in the Diameter server, it returns an AA-Answer with the Result-Code AVP set to DIAMETER_SUCCESS (7). After receiving a positive authentication response, the web server can respond to the user clients request (8).

Besides general authentication, service specific authorization is supported. Services for which the Diameter client wants to authorize its user are identified by the combination of the Service-Context-Id and Service-Identifier attributes.

3.2.2. Accounting

In addition to common identity management functions, Diameter WebAuth aims at providing web applications with some basic accounting support. This allows web applications to relay fees it wants to charge the user for accessing a particular service to the Diameter server. The Diameter server then checks the charge against the user's account and approves or denies the charge. This mechanism offers a number of advantages compared to a solution where the web application has to implement accounting functions on its own. First of all, there is not implementation effort on part of the web application. By utilizing Diameter WebAuth the web application is automatically capable of processing credit charges. Second of all, the web application provider does not need to concern itself with invoicing, credit verification, transaction processing or security concerns. Thirdly, the end user only needs to trust its identity provider with payment related data. This increases the trust the user has into the payment system and also allows for a, for the web application, transparent handling of payment methods. The user can maintain several different payment methods and switch between them as needed without effecting the web application provider in any way. Last but not least, centralizing the payment processing at the identity provider generates synergy effects for the affiliated web application providers because the effectively share a payment infrastructure. Especially for small web sites, with only a very limited number of services or products they want to charge for, it is beneficial to be able to use the identity provider also as a payment provider for their customers.

The Diameter credit-control application RFC 4006 [22] specifies very extensive support for credit handling in Diameter environments. The Diameter WebAuth specification will use a subset of this specification to provide some basic credit control commands. This coincides with the recommendation of the Diameter applications design guidelines [16] to reuse existing Diameter applications and commands if possible. Furthermore it allows a Diameter WebAuth client to be part of a full-blown Diameter credit-control infrastructure. This specification uses the Credit-Control-Request and Credit-Control-Answer commands from RFC 4006 [22, Sections 3.1. and 3.2.] to support the one time credit events "Balance Check", "Direct Debiting" and "Refund" [22, Sections 6.2. to 6.4.]. The events will be detailed in the following sections.

Balance check

In order to check if a user has sufficient balance for a specific service, the Diameter client issues a Credit-Control-Request command. The request MUST include a CC-Request-Type AVP set to EVENT_REQUEST, a Requested-Action AVP set to

CHECK_BALANCE and a Requested-Service-Unit AVP which contains the corresponding balance value to be checked for. The Diameter server replies to a balance check Credit-Control-Request with a Credit- Control-Answer where the Check-Balance-Result AVP is present and set to either ENOUGH_CREDIT or NO_CREDIT. For more information on the credit-control balance check see RFC 4006, Section 6.2. [22].

Direct debiting

To charge a user for a service, the Diameter client issues a Credit- Control-Request command. The request MUST include a CC-Request-Type AVP set to EVENT_REQUEST and a Requested-Action AVP set to DIRECT_DEBITING. The actual charged value is included in a Requested-Service-Unit AVP. The Diameter server responds with a Credit-Control-Answer which includes a Granted-Service-Unit AVP. For more information on credit- control direct debiting see RFC 4006, Section 6.3. [22].

Refund

If a user needs to be refunded, the Diameter client issues a Credit- Control-Request command with the CC-Request-Type AVP set to EVENT_REQUEST and the Requested-Action AVP set to REFUND_ACCOUNT. The refund value is included in a Requested-Service-Unit AVP. The Diameter server responds with a Credit-Control-Answer which includes a Granted-Service-Unit AVP. For more information on credit-control refunding see RFC 4006, Section 6.4. [22].

3.2.3. Identity attributes

Reliable authentication is probably the most important feature an identity management framework has to provide. However, once the identity of a user is verified by the web application, further personalization of the offered services is possible. For example, the user can be welcome with a personal greeting that includes his name or the web site can offer the user to use his personal address for shipping goods he orders. Those characteristics describing an identity beyond the login name are called identity attributes. This specification aims at including identity attributes into its framework. This allows to store information that are closely linked to an identity in the same central manner the identity is stored.

Diameter WebAuth provides some basic means to transport identity information over the Diameter protocol. This can be attributes like first name, last name, address or other contact information. The scope and value of the potential information is dependent on the schema used to exchange these identity information. The schema is interchangeable

and has to be predefined between the web application provider (Diameter client) and the identity management provider (Diameter server). Therefor the actual format for the values of the Identity-Attribute-Request AVP and the Identity-Attribute-Value AVP is dependent on the employed identity information schema and is beyond the scope of this thesis. Suitable schemes SHOULD be defined in another Diameter application, in standards written by other standardization bodies, or in service- specific documentation.

Identity attributes are transported using the Identity-Information- Query/Response AVPs. Multiple such AVPs can be included in a Diameter request or response to query multiple identity information. Every Identity-Information-Query MUST be processed separately by the Diameter. A special order of the AVPs does not need to be considered or retained. However the answer command to the request containing the Identity-Information-Query AVPs MUST include a corresponding Identity-Information-Response AVP for every Identity-Information- Query AVP.

Encapsulated identity querying within AA commands

Identity information querying is possible directly within the AA-Request/Answer commands. This allows the client to immediately obtain identity information after an authentication or authorization request without the need for another protocol run. If the client includes an Identity-Information-Query AVP in an AA-Request command, the Diameter server MUST first process the authentication and/or authorization request. If the authentication and/or authorization is successful, the server then MUST process the identity information request and include the corresponding Identity- Information-Response in its final message to the client. If the authentication and/or authorization fails, the server MUST discard the Identity-Information-Query. Figure 3.2 shows an example of encapsulated querying.

Using encapsulated identity querying is recommended when protocol efficiency is an issue and the needed identity information are already known at the time of the authentication and/or authorization request. Also the need for having a session opened on the Diameter server can be avoided when the web application queries all the necessary identity, authentication and authorization information during only one protocol run.

Dedicated identity querying

In case the web application needs identity information about its user outside an authentication and/or authorization request, it can query identity information using the Identity-Information-Request (IIR) command. Depending on the nature and sensitivity of the identity data the Diameter server may require the Diameter client to authenti-

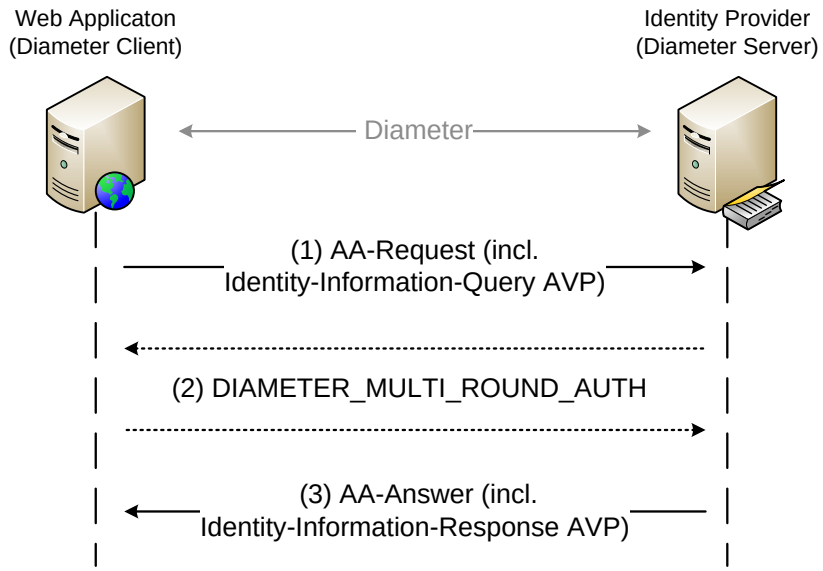


Figure 3.2.: Encapsulated querying of identity information inside AA commands

cate and/or authorize its user prior to process an Identity- Information-Query. In this case, the Diameter server MUST set the Identity-Action-Result AVP to AUTHENTICATION_REQUIRED or AUTHORIZATION_REQUIRED and not include an Identity-Attribute-Value AVP in the corresponding Identity-Information-Result. Further privacy considerations are discussed in Section 3.5.

3.3. Diameter WebAuth commands

A Diameter message exchange is dedicated to a certain command which is indicated by the Diameter command code value inside the header of the packets associated with the message exchange. Diameter commands put a Diameter message into a context for the communicating partners. They define the AVPs that are valid in this context and how the Diameter server, and the Diameter client respectively, are to process the messages (cp. Section 2.5.1).

This section describes the Diameter commands and the associated Command-Code values that MUST be supported by all Diameter implementations conforming to this specification. The command codes are listed in table 3.1 and detailed consecutively.

Command Name	Abbrev.	Code	Reference
AA-Request	AAR	265	Section 3.3.1
AA-Answer	AAA	265	Section 3.3.2
Credit-Control-Request	CCR	272	Section 3.3.3
Credit-Control-Answer	CCA	272	Section 3.3.4
Identity-Information-Request	IIR	TBD*	Section 3.3.5
Identity-Information-Answer	IIA	TBD*	Section 3.3.6

* The values for new command codes have to be assigned by the IANA.

Table 3.1.: Diameter WebAuth command codes

3.3.1. AA-Request (AAR) command

The AA-Request (AAR) command is specified in RFC 4005, Section 3.1. [10] and It is used by the Diameter client to request authentication and/or authorization for its user.

If authentication is requested, depending on the authentication scheme and the sequence of requests different attributes MUST be present: User-Name and User-Password for basic authentication and a HTTP-Digest-Response if it is an AA-Request following an AA-Answer with its Result-Code set to DIAMETER_MULTI_ROUND_AUTH and including a HTTP-Digest-Challenge.

If authorization is requested, the Service-Context-Id and Service- Identifier attributes are used to identify the service for which authorization is requested. If these attributes are missing in the request and the Auth-Request-Type attribute is set to AUTHORIZE_AUTHENTICATE, the Diameter server SHOULD handle the request as if authorization has not been requested.

The AA-Request command has the following ABNF[4] grammar (AVPs not required by this specification are omitted):

```
<AA-Request> ::= < Diameter Header: 265, REQ, PXY >
                 < Session-Id >
                 { Auth-Application-Id }
                 { Origin-Host }
                 { Origin-Realm }
                 { Destination-Realm }
                 { Auth-Request-Type }
                 [ User-Name ]
```

[4] ABNF stands for Augmented Backus-Naur Form and is a formal syntax specification [14].

```
                [ User-Password ]
                [ HTTP-Digest-Response ]
                [ Destination-Host ]
                [ Service-Context-Id ]
                [ Service-Identifier ]
              * [ Proxy-Info ]
              * [ Route-Record ]
              * [ AVP ]
```

3.3.2. AA-Answer (AAA) command

The AA-Answer (AAA) command is specified in RFC 4005, Section 3.2. [10] and is sent by the Diameter server in response to an AA- Request.

If the AA-Answer is a response to a AA-Request initiating a digest authentication, the Result-Code AVP MUST be set to DIAMETER_MULTI_ROUND_AUTH and a HTTP-Digest-Challenge AVP MUST be present. If the AA-Answer is a response to an authorization request, the Service-Context-Id and Service-Identifier attributes identifying the service for which authorization is granted or denied MUST be present.

The AA-Answer command has the following ABNF grammar (AVPs not required by this specification are omitted):

```
<AA-Answer> ::= < Diameter Header: 265, PXY >
                < Session-Id >
                { Auth-Application-Id }
                { Auth-Request-Type }
                { Result-Code }
                { Origin-Host }
                { Origin-Realm }
                [ User-Name ]
                [ HTTP-Digest-Challenge ]
                [ HTTP-Authentication-Info ]
                [ Service-Context-Id ]
                [ Service-Identifier ]
              * [ Proxy-Info ]
              * [ AVP ]
```

3.3.3. Credit-Control-Request (CCR) command

The Credit-Control-Request (CCR) command is specified in RFC 4006, Section 3.1. [22] and is sent by a web application to initiate a credit-control accounting action. This specification only requires the support of AVPs required for the the one time events "Balance

Check", "Direct Debiting" and "Refund" (cp. Section 3.2.2. This means that the CCR command mainly has to allow for the CC-Request-Type AVP and the Requested-Action AVP which indicate the type of credit operation requested and the Requested-Service-Unit AVP which specifies the particular amount associated with the request. The CCR command has the following ABNF grammar (AVPs not required by this specification are omitted):

```
<Credit-Control-Request> ::= < Diameter Header: 272, REQ, PXY >
                             < Session-Id >
                             { Origin-Host }
                             { Origin-Realm }
                             { Destination-Realm }
                             { Auth-Application-Id }
                             { Service-Context-Id }
                             { CC-Request-Type }
                             { CC-Request-Number }
                             [ Destination-Host ]
                             [ User-Name ]
                             [ Event-Timestamp ]
                           * [ Subscription-Id ]
                             [ Service-Identifier ]
                             [ Requested-Service-Unit ]
                             [ Requested-Action ]
                           * [ Proxy-Info ]
                           * [ Route-Record ]
                           * [ AVP ]
```

3.3.4. Credit-Control-Answer (CCA) command

The Credit-Control-Answer (CCA) command is specified in RFC 4006, Section 3.2. [22] and is sent by a Diameter server to acknowledge a Credit-Control-Request command. The CCA command in this specification, therefore, has to allow for AVPs that are used to reply to the one time events supported here. These are the Granted-Service-Unit AVP to answer "Direct Debiting" and "Refund" requests and the Check-Balance-Result AVP which is used to answer a "Balance Check" request. The CCA command has the following ABNF grammar (AVPs not required by this specification are omitted):

```
<Credit-Control-Answer> ::= < Diameter Header: 272, PXY >
                            < Session-Id >
                            { Result-Code }
                            { Origin-Host }
```

```
                { Origin-Realm }
                { Auth-Application-Id }
                { CC-Request-Type }
                { CC-Request-Number }
                [ User-Name ]
                [ Event-Timestamp ]
                [ Granted-Service-Unit ]
                [ Check-Balance-Result ]
              * [ Proxy-Info ]
              * [ Failed-AVP ]
              * [ AVP ]
```

3.3.5. Identity-Information-Request (IIR) command

The Identity-Information-Request (IIR) is indicated by setting the Command-Code field to TBD[5]. The "R" bit in the command flags also has to be set to designate this command as a request. The IIR command is used by the Diameter client to request additional information about the identity of its user. The requested identity information are encoded in one or more Identity-Information-Query AVPs that are included in the request. The Identity-Information-Query AVP will be specified in Section 3.4.2. Depending on the schema used for the identity request, the Diameter server might expect a User-Name AVP also present to identify the user for which the information is requested. The IIR command has the following ABNF grammar:

```
<Identity-Information-Request> ::= < Diameter Header: TBD, REQ, PXY >
                                   < Session-Id >
                                   { Auth-Application-Id }
                                   { Origin-Host }
                                   { Origin-Realm }
                                   { Destination-Realm }
                                 * { Identity-Information-Query }
                                   [ Destination-Host ]
                                   [ User-Name ]
                                 * [ Proxy-Info ]
                                 * [ Route-Record ]
                                 * [ AVP ]
```

[5]The values for new command codes have to be assigned by the IANA.

3.3.6. Identity-Information-Answer (IIA) command

The Identity-Information-Answer (IIA) is indicated by setting the Command-Code field to TBD[6]. The "R" bit in the command flags has to be cleared to designate this command an answer. The IIA command is used by the Diameter server to respond to an Identity-Information-Request. It includes an Identity-Information-Result AVP for every Identity-Information-Query AVP that was present in the corresponding request. The Identity-Information-Result AVP contains the outcome of the identity information operation in the server that was initiated by the corresponding identity information query. The IIA command has the following ABNF grammar:

```
<Identity-Information-Answer> ::= < Diameter Header: TBD, PXY >
                                  < Session-Id >
                                  { Auth-Application-Id }
                                  { Result-Code }
                                  { Origin-Host }
                                  { Origin-Realm }
                                * { Identity-Information-Result }
                                  [ User-Name ]
                                * [ Proxy-Info ]
                                * [ Route-Record ]
                                * [ AVP ]
```

3.4. Diameter WebAuth AVPs

Attribute-Value-Pairs (AVPs) are used by the Diameter protocol to encapsulate the individual attributes of a Diameter message (cp. Section 2.5.1). The Diameter applications design guidelines [16] strongly suggest to reuse AVPs that are defined in existing Diameter applications where appropriate. The Diameter WebAuth specification will, therefore, import AVPs from the Diameter Base Protocol [8], the Diameter Network Access Server Application [10] and the Diameter credit-control application [22] to conform to the design guidelines and to avoid having to define new AVPs unnecessarily. To implement the identity information commands, however, a number of new AVPs are defined by this specification. The following section provides a listing of the AVPs used in Diameter WebAuth commands and their values. Where appropriate, further explanations will be given.

[6] The values for new command codes have to be assigned by the IANA.

3.4.1. Imported AVPs

Diameter base protocol

A number of common attributes are used by Diameter WebAuth that are specified in the Diameter base protocol (RFC 3588 [8]). Most noteably this is the User-Name AVP. Since every Diameter implementation MUST implement the Diameter base application, support for these AVPs can be taken as granted. For the sake of completeness, Table A.1 in Appendix A lists all the AVPs that are reused from the Diameter base protocol specification.

Diameter Network Access Server application

The HTTP basic authentication described in the Diameter WebAuth design, uses the User-Password AVP defined in the Diameter Network Access Server application (RFC 4005 [10]) to transport the user password. Table A.2 in Appendix A lists all the AVPs that are reused from the Diameter Network Access Server application specification.

HTTP-Digest authentication

The Diameter Session Initiation Protocol (SIP) application (RFC 4740 [19]) defines AVPs to implement the HTTP-Digest authentication for its purposes. The Session Initiation Protocol also uses the digest authentication described in RFC 2617 for its own underlying digest authentication scheme [51, Section 22.4.]. The Diameter WebAuth specification, therefore, reuses the AVPs specified in the Diameter SIP application in order to implement the HTTP digest authentication. For descriptive reasons, however, some of them are renamed. The following list describes the container AVPs that are imported.

The HTTP-Digest-Challenge AVP is identical to the SIP-Authenticate AVP [19, Section 9.5.3.] and contains data related to the HTTP Authorization Request Header [18, Section 3.2.2.]. Detailed ABNF grammer for this AVP is available in Appendix A, Section A.3.

The HTTP-Digest-Response AVP is identical to the SIP-Authorization AVP [19, Section 9.5.4.] and contains data related to the HTTP WWW-Authenticate Response Header [18, Section 3.2.1.]. Detailed ABNF grammer for this AVP is available in Appendix A, Section A.3.

The HTTP-Digest-Info AVP is identical to the SIP-Authentication-Info AVP [19, Section 9.5.5.] and contains data related to the HTTP Authentication-Info Header

Attribute Name	AVP code	Value Type	Section defined
Identity-Information-Query	TBD*	Grouped	3.4.2
Identity-Information-Result	TBD*	Grouped	3.4.2
Identity-Action-Requested	TBD*	Enumerated	3.4.2
Identity-Action-Result	TBD*	Enumerated	3.4.2
Identity-Information-Schema	TBD*	UTF8String	3.4.2
Identity-Attribute-Request	TBD*	UTF8String	3.4.2
Identity-Attribute-Value	TBD*	UTF8String	3.4.2

* The values for new command codes have to be assigned by the IANA.

Table 3.2.: Identity information AVPs

[18, Section 3.2.3.]. Detailed ABNF grammer for this AVP is available in Appendix A, Section A.3.

A complete list of all AVPs from the Diameter Session Initiation Protocol (SIP) application that are reused by this specification is provided in Appendix A.3.

Diameter credit-control application

This specification uses a subset of the Diameter credit-control application (RFC 4006 [22]). The credit control commands (cp. Sections 3.3.3 and 3.3.4) as well as the associated AVPs used by Diameter WebAuth are defined in RFC 4006. A complete list of AVPs that are reused from the Diameter credit-control application specification as well as some further explanations are provided in Appendix A.4.

3.4.2. Identity information AVPs

The following section details the AVPs used by this specification to exchange identity information. They are newly defined here and MUST be implemented by Diameter applications conforming to the Diameter WebAuth specification. In the following the AVPs will be described and further specified by the allocation of an AVP data format according to the base protocol specifications (cp. [8, Sections 4.2. to 4.4.]).

Table 3.2 lists the AVPs, their AVP code values and types.

Table 3.3 lists possible AVP header flag values as described in the base protocol (cp. [8, Section 4.1.]) and whether the AVP MAY be encrypted. The “MUST” column indicates

which bit has to be set in the AVP header. All the identity information AVPs specified in this section MUST have the "M" bit set which indicates that the support of this AVP is mandatory. If an AVP with this bit set is received and not recognized, the complete message MUST be rejected unless the receiving party is a Diameter Relay. The reason for this is that all the AVPs are essential to the functionality of the identity information operations. If an implementation does not support any of them, the identity information facilities cannot work.

The "MAY" column indicates which header bits MAY be set for the corresponding AVP. All the AVPs specified in this section MAY have the "P" bit set which indicates the need for encryption for end-to-end security for the particular AVP. The "P" bit is allowed for all the AVPs because encryption does not break the functionality of the identity information operations since no other party than the Diameter client and server need to be able to access them.

The "SHLD NOT" column indicates header flags that should not be set for the corresponding AVP. For the AVPs specified in this section, there are no flags that should not be set. The "MUST NOT" column indicates header flags that cannot be set for the corresponding AVP. The "V" which indicates that the corresponding AVP code belongs to the specific vendor code address must not be set for any AVP specified in this section. An adoption as internet draft of this specification assumed, the AVPs will get official AVP codes assigned by the IANA, and therefore, not belong to any vendor specific address space.

The "Encr" column specifies whether encryption for this AVP is mandatory if the Diameter message containing that particular AVP is sent via a Diameter agent (proxy, redirect or relay). RFC 3588 specifies that "the message MUST NOT be sent unless there is end-to-end security between the originator and the recipient and integrity / confidentiality protection is offered for this AVP OR the originator has locally trusted configuration that indicates that end-to-end security is not needed" [8, Section 4.5.]. This form of protection is required for all of the AVPs specified in this section to ensure that an intermediary party cannot access or alter the identity information included in the message.

Identity-Information-Query AVP

The Identity-Information-Query (AVP code TBD) is of type Grouped[7] and contains data to specify the identity information requested by the client. It includes all the AVPs necessary to compose a single identity information request. Those are an Identity-Action-Requested AVP which specifies what kind of operation the client expects from

[7] Grouped AVPs are containers that include a number of other AVPs (cp. [8, Section 4.4.]).

Attribute Name	MUST	MAY	SHLD NOT	MUST NOT	Encr
Identity-Information-Query	M	P		V	Y
Identity-Information-Result	M	P		V	Y
Identity-Action-Requested	M	P		V	Y
Identity-Action-Result	M	P		V	Y
Identity-Information-Schema	M	P		V	Y
Identity-Attribute-Request	M	P		V	Y
Identity-Attribute-Value	M	P		V	Y

Table 3.3.: Identity information AVP flags

the server, an Identity-Information-Schema AVP which specifies the schema to put the request in context, and an Identity-Attribute-Request AVP which contains the actual identity request. It can also include an Identity-Attribute-Value AVP in case the request is supposed to store a value in the Diameter server. This means, the Identity-Action-Requested AVP is set to STORE_DATA (cp. Section 3.4.2). In this case, the Identity-Attribute-Value AVP contains the value to be stored.

The Identity-Information-Query AVP has the following ABNF grammar:

```
Identity-Information-Query ::= < AVP Header: TBD >
                               { Identity-Action-Requested }
                               { Identity-Information-Schema }
                               { Identity-Attribute-Request }
                               [ Identity-Attribute-Value ]
                             * [ AVP ]
```

Identity-Information-Result AVP

The Identity-Information-Result (AVP code TBD) is of type Grouped and contains the result for an Identity-Information-Query. It mirrors the Identity-Information-Query AVP from the corresponding request and additionally contains AVPs with the results. This is for one, an Identity-Action-Result AVP which indicates whether the request was processed successfully or the reason why it was not. In case the requested action was not to store a value, the Identity-Information-Result AVP also includes an Identity-Attribute-Value AVP which contains the result value for the request. If the requested action was to store a value, the Diameter server MAY choose to mirror the Identity-Attribute-Value AVP from the corresponding Identity-Information-Query AVP.

The Identity-Information-Result AVP has the following ABNF grammar:

```
Identity-Information-Result ::= < AVP Header: TBD >
                                { Identity-Action-Requested }
                                { Identity-Action-Result }
                                { Identity-Information-Schema }
                                { Identity-Attribute-Request }
                                [ Identity-Attribute-Value ]
                               * [ AVP ]
```

Identity-Action-Requested AVP

The Identity-Action-Requested AVP (AVP code TBD) is of type Enumerated and is used to determine the requested type of action for the corresponding identity information. At this moment, the only actions supported are to retrieve data or to store data. The value

RETRIEVE_DATA (0) requests the retrieval of the data specified by the Identity-Attribute-Request AVP, and the value

STORE_DATA (1) requests the storage of the data submitted with an Identity-Attribute-Value AVP corresponding to the schema specific location in the Identity-Attribute-Request AVP.

Identity-Action-Result AVP

The Identity-Action-Result AVP (AVP code TBD) is of type Enumerated and indicates the result of an identity information request. The result can either be successfull in which case the AVP contains a simple confirmation or the requested operation may have failed in which case the AVP contains a hint to the reason for that. The values are:

RESULT_OK (0) if the request has been processed without errors.

ACCESS_DENIED (1) if the Diameter client has no access privileges for the requested action to the specific attribute. The access can be dependent on the Identity-Action-Requested. For example a RETRIEVE_DATA request maybe granted but a STORE_DATA request is denied.

AUTHENTICATION_REQUIRED (2) if the Diameter client is required to authenticate its user before the corresponding identity information query will be processed.

AUTHORIZATION_REQUIRED (3) if the Diameter client is required to authorize its user before the corresponding identity information query will be processed.

UNKNOWN_SCHEMA (4) if the Diameter server does not recognize the identity information schema the client submitted. Therefore the request could not be processed.

INVALID_REQUEST (5) if the Diameter sever could not process the request because it could not be validated according to the corresponding identity information schema.

Identity-Information-Schema AVP

The Identity-Information-Schema AVP (AVP code TBD) is of type UTF8String and contains the identifier for the identity information schema that applies to the corresponding Identity-Information- Attribute. The UTF8String format is chosen for this and the following AVPs since it seems to be the most appropriate. An UTF8String is a human readable string in Unicode format. The identifier is needed in order to understand how to interpret the identity information correctly and is allocated by the service provider, the web application provider, the device manufacturer or by a standardization body. It MUST uniquely identify a schema to describe identity information. The format of the Identity-Information-Schema is:

```
"identity-schema" "@" "domain"

identity-schema = Token

domain = String that represents the entity that allocated the
         identity-schema (e.g. ietf.org or provider.example.com)
```

If the requested identity schema is unknown to the responding Diameter server, it MUST include the Identity-Action-Result AVP set to UNKNOWN_SCHEMA in its reply.

Unless the identity information schema is for private use only, it SHOULD be defined in another Diameter application, in standards written by other standardization bodies, or in service- specific documentation. Otherwise it is RECOMMENDED to publish the specification as informal RFC.

Examples of identity schemes are:

```
vCardv3.0@imc.org
ax1.0@openid.net
...
```

Identity-Attribute-Request AVP

The Identity-Attribute-Request AVP (AVP code TBD) is of type UTF8String and contains the attribute request according to the identity information schema.

The possible values of the Identity-Attribute-Request AVP are schema specific and therefore not specified in this thesis. Whoever predefined the identity information schema is also responsible for describing the syntax and semantics for the Identity-Attribute-Request values.

Identity-Attribute-Value AVP

The Identity-Attribute-Value AVP (AVP code TBD) is of type UTF8String and contains the attribute value matching the Identity-Attribute- Request AVP.

The possible values of the Identity-Attribute-Value AVP are schema specific and therefore not specified in this thesis. Whoever predefined the identity information schema is also responsible for describing the syntax and semantics for the Identity-Attribute-Value values.

3.5. Privacy considerations

The Diameter application aims at covering setups where Diameter clients and Diameter servers belong to more than one administrative domain. In those setups the end user often has a trust relationship with the provider of the Diameter server but not with the provider of the web applications that are the Diameter clients. In order to allow a smooth operation of the services the user requested, the Diameter server has to make certain personal information about the user available to the application provider. And although the user should be aware of that, it can be generally expected that access to such personal information is kept on a minimum need-to-know basis across different administrative domains. For example the application provider may need to know if the user has a certain membership which allows him to access the service he requested. The number and details about further memberships the user may or may not have however, is not relevant for the application provider at that moment. This section therefore addresses a number of privacy consideration that may arise in general or when dealing with a setup over multiple administrative domains. Since usually there are no private information that the client has but not the server, the privacy considerations will focus on the issue to protect information that are available in the Diameter server from access by the Diameter client.

There are three different sets of data that are directly (by value) or indirectly (by true/false responses) accessible by the Diameter client. There are authentication and authorization data, credit related data and identity information data.

3.5.1. Authentication

Generally, in setups where user privacy is an aspect, Diameter servers SHOULD always require a user authentication before any kind of personal information is made accessible to the Diameter client. By requiring an authentication, user data probing by a rogue or compromised Diameter client is made more difficult since only data from users that are currently logged onto the client or whose login credentials are known can be pried. If the authentication status for a session is not maintained on the server, every action specified in this chapter can be queried using an AA-Request command which then MUST also include proper authentication credentials. However since an authentication procedure possibly triggers some kind of user interaction in the web client, it is RECOMMENDED to keep such AA- Requests to a minimum. This can be achieved for example by querying the Diameter server for all the data that is likely to be needed for a session inside the first request. Although this may sound counterintuitive to the objective of keeping private information exposure on a minimum need-to-know basis, it doesn't make a difference if data which a client is entitled to is transferred all at once in the beginning of a session or gradually throughout the session. Implementations of this specification which want to allow privacy protection SHOULD offer a configuration option to enforce user authentication before any other operation is allowed.

A Diameter WebAuth implementation SHOULD protect personal data by keeping authorization data service specific and by limiting available authentication schemes to the ones which do not expose sensitive data. Keeping authorization data service specific means that the Diameter server SHOULD NOT authorize the user for services that the Diameter client doesn't actually offer. This means that an AA-Answer to an authorization request SHOULD NOT include Service-Identifiers for services that are unavailable at the client the request came from. Unfortunately, the Diameter server cannot directly influence the authentication scheme that the Diameter client uses with its web client (cp. also Section 3.6.3). However, limiting the available authentication schemes to more secure ones will hopefully encourage Diameter clients to be deployed using only the available authentication schemes to begin with. This should make eavesdropping on the Diameter client, web client connection more difficult and also will require more changes to a compromised Diameter client in order to gain access to plain text authentication credentials. The only authentication scheme which can be considered reasonable secure and is currently supported by this specification is HTTP-Digest authentication.

3.5.2. Credit-control

In order to protect user sensitive credit data, a Diameter server implementation MAY offer a facility to hide the user's credit balance. If a Diameter server wants to hide the actual user's credit balance it MUST answer a credit-control balance check request with ENOUGH_CREDIT, regardless of the actual credit balance if the credit control server is willing to accept credit charges for the particular user and NO_CREDIT otherwise. Please not, that this behavior is not covered by RFC 4006 [22]. However since a balance check answer is not intended to make any credit-reservations or promises about the credit balance beyond the moment of the request, this behavior will not break compatibility with other Diameter credit-control implementations. To be sure that a user is charged for a service the Diameter client is about to grant him access to, the Diameter client MUST only rely on the resulting Credit-Control-Answer to his DIRECT_DEBITING Credit-Control-Request.

3.5.3. Identity information

Identity information available at the Diameter server can include a wide variety of personal data. Depending on the overall site setup those data can vary from marginally sensitive data with relevance to all Diameter clients to highly sensitive data which is only relevant to a small fraction of the attached Diameter clients. A specific recommendation how to protect identity information on an implementation level cannot be given here. In general, a Diameter WebAuth server implementation SHOULD offer some sort of access control system for the available identity information, which allows to control what identity information is available to which Diameter client. Beyond the actual server implementation a Diameter WebAuth server provider SHOULD offer its clients some sort of control or at least disclosure about what identity information are available and accessible to which Diameter client provider.

3.6. Security considerations

This chapter describes a Diameter application which enables web applications to access AAA services of a Diameter server. The Diameter Base Protocol (RFC 3588) is used for the communication between the Diameter client and the Diameter server. The security considerations for the Base Protocol, therefore, apply for this specification as well [8, Section 13]. They address the application of IP security (IPsec) and Transport Layer Security (TLS) to secure Diameter messages. For this specification it is assumed, that the message exchange between the Diameter client and the Diameter server can be reasonably

secured by respecting the security considerations in RFC 3588.

For the communication between the end user and the Diameter client a service specific protocol is used. When the Diameter client is employed in a web application, usually this will be the Hypertext Transfer Protocol (HTTP, RFC 2616). The security considerations for the service specific protocol SHOULD be considered when a Diameter WebAuth client is implemented. In case of a web application employing HTTP, the correspondent security considerations are made in [17, Section 13]. In either case, since the service protocol is used to exchange authentication information with the end user, measures SHOULD be taken to secure the communication between the Diameter client and the end user client. To secure HTTP message exchanges, for example, HTTPS (HTTP over TLS, RFC 2818 [48]) SHOULD be used.

3.6.1. Basic authentication

The basic authentication scheme uses a cleartext[8] user name/password combination to authenticate a user. This makes the basic authentication absolutely insecure. First of all, the password is exposed to any third party which might be able to listen to the message exchange between the user client and the Diameter client. For example because the message exchange is not encrypted, the encryption was broken of for other reasons. And second of all, the password, inevitably, is exposed to the Diameter client. Especially in setups where the Diameter client and the Diameter server are not part of the same administrative domain this severely compromises the end user's identity. Even in setups where Diameter client and server are within the same administrative domain, user passwords should never be accessable in cleartext. Otherwise in case of a compromised Diameter client, all the user accounts are compromised too. Because the basic authentication is that insecure, it SHOULD NEVER be employed in a productive Diameter setup, unless absolutely no other option is viable. Furthermore basic authentication SHOULD only be used over encrypted and secure transport channels with some sort of server authentication before the credentials are sent. Besides these Diameter WebAuth oriented security considerations, those of the HTTP Authentication specification (RFC 2616, [18, Section 4.] also need to be considered. They state explicitly that "the Basic authentication scheme is not a secure method of user authentication, nor does it in any way protect the entity, which is transmitted in cleartext across the physical network used as the carrier."

[8] Technically the user name and password are encoded using the base 64 encoding scheme [26]. This is, however, only an encoding, not an encryption and thus reversible without difficulty.

3.6.2. Digest authentication

Like the basic authentication, the digest authentication uses a user name in combination with a password to authenticate the user. In contrast to the basic authentication, however, the digest authentication does not exchange the password in cleartext. It uses a one-way hash function to calculate a check value from the combination of the user password and a nonce that was exchanged with the authentication partner. Both authentication partners calculate this value on their own, and the client which is to be authenticated sends its value to the authenticator. If the values match, both used the same password and, therefore, the authentication is successfull.

The digest authentication, if implemented and executed correctly, does provide a better authentication mechanism than basic authentication. Especially an eavesdropping third party cannot recover the cleartext password from an intercepted message exchange. Nor can he use it for replay attacks when the server does not reuse its nonce values. Nevertheless, the HTTP Authentication specification (RFC 2616, [18, Section 4.]) has a number of security considerations that must be considered. Especially is the digest authentication scheme susceptible to man in the middle attacks. It does provide some resilience against the attacker recovering the cleartext password in those cases though. Also the security considerations of the RADIUS Extension for Digest Authentication specifications (RFC 4590) which the Diameter digest authentication is derived from need to be considered [55, Section 8.]. As a result, the digest authentication scheme also SHOULD only be used over encrypted and secure communication channels. This includes the authentication of the Diameter client to the user client, for example HTTPS with public key certificates.

3.6.3. Renegade or compromised WebAuth clients

Special considerations need to be made for the situation where a Diameter WebAuth client is compromised or renegade. In both cases the WebAuth client will try to exploit its natural position as man in the middle between the user client and the Diameter server to compromise user accounts. A natural goal of an attacker in this position is to gain access to cleartext user credentials. Since the Diameter WebAuth server does not allow direct querying of user names or passwords, the WebAuth client has two possibilities. It can probe for valid user name/password combination if the server accepts basic authentication AA-Requests or it can wait for user to authenticate themselves. Probing for valid combinations is not very promising and will not considered any further here. Having users to try and authenticate themselves to a WebAuth client that is trying to compromise their accounts, on the other hand, is a severe problem.

As discussed above, when using digest authentication even a man in the middle attack

has only limited chances of recovering the cleartext password. A man in the middle attacker, however, can simply switch the authentication scheme used towards the user client to basic authentication. This would give him unrestricted access to the cleartext user name and password for every user that logs in through the Diameter client. This kind of attack is described in RFC 2616 as well [18, Section 4.]. Coinciding with the RFC, the only viable options to counteract such attacks lie within the user agent. For example, only if the user agent warns the user when basic authentication is requested, or in general indicates to the user what kind of authentication is about to be used, this kind of attack can be prevented by the user. Another possibility is for the client to offer a configuration option which either disables basic authentication completely or just for different web sites. For the future of this specification it also SHOULD be considered to implement other authentication methods. This will not prevent renegade or compromised WebAuth clients from being able to switch authentication schemes, but from a user's perspective it is much more obvious when, for example, instead of the usual certificate based authentication a web server suddenly ask for a password.

3.7. Summary

In this chapter, a new approach to identity management for web applications was designed.

- A proposal for network-based identity management in web applications was developed.
- The proposal is called Diameter WebAuth and is implemented as a Diameter application.
- Diameter WebAuth includes commands to authenticate and authorize users, charge them, and query additional identity information about them.
- The specification includes HTTP basic and digest authentication mechanisms.
- A Diameter WebAuth client can be seamlessly integrated into a Diameter credit-conroll infrastructure or use a Diameter WebAuth server for basic credit-control operations.
- The identity information commands are suitable to support arbitrary identity information schemes.
- Specific privacy and security considerations have been made.

4. Implementation

In the previous chapter a new approach to identity management for web applications based on the Diameter protocol was developed. To augment and support the theoretical approach with a practical implementation, the different elements of a common web application setup that employs Diameter WebAuth were implemented. This includes the development of a working prototype of the Diameter WebAuth application as designed in the previous Chapter as well as a web application to showcase the potentials of the proposal.

4.1. Overview

Figure 4.1 shows an overview of the different parts of the implementation work that will be done in this chapter. The setup includes a user that access a web application via his browser. The web application will employ the Diameter WebAuth framework to perform identity management actions against a Diameter server. For example, user authentication and identity attribute querying.

To reproduce this setup in the prototype implementation, three different components need to be implemented. First, the server element of Diameter WebAuth. Second, the client element and third the web application which employs the WebAuth framework. The goal of the implementation work is to have a working prototype implementation of a complete Diameter WebAuth setup including a web application to demonstrate its features. The use-cases specified in Section 3.1.2 will be used as a guideline for the implementation and its validation. This means, the implementation has to cover and handle all the use cases as described in the design.

4.2. Diameter WebAuth application

In the following the implementation of the Diameter application will be presented in detail. First the implementation of the general classes that are employed by both, client and server applications, will be illustrated. After that, the server implementation followed

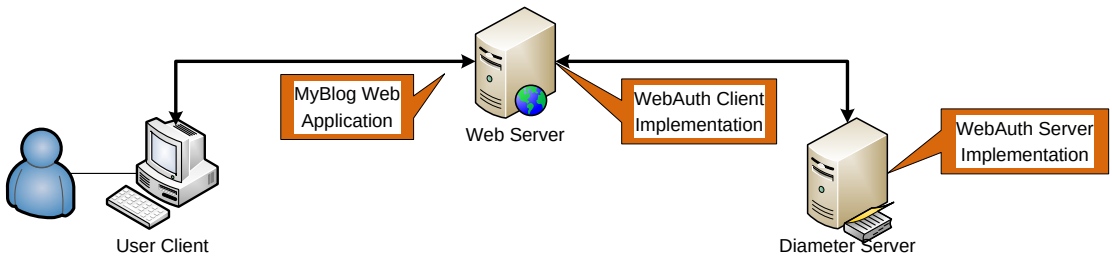


Figure 4.1.: Overview of the prototype implementation

by the client implementation will be detailed. Finally, the implementation of a small test suite is discussed.

4.2.1. Implementation basis

At the beginning of the implementation work, a suitable implementation basis had to be found. Since the focus of this work is a Diameter application rather than a comprehensive implementation of a Diameter stack including the base protocol, it was desireable to find an Diameter implementation that could be extended by the application designed in this thesis. The criteria for such a project were that it must have a working implementation of the base protocol as well as a working client/server implementation, that is was available under a licence which permits to use it for this work and that it was acceptable documented.

At the end of the evaluation phase the decision was made to use the Java implementation JDiameter [63] as basis for the implementation work. At a close second was the OpenDiameter C++ implementation [40] but it doesn't seem to have a solid client/server implementation yet and the documentation was outdated. Furthermore the Java technology is very well suited for web application development, which was assumed to ease the development of the showcase web application in this work later on.

4.2.2. Structure

The Diameter WebAuth application specifies three command sets which need to be implemented in the prototype implementation. Those are AA commands (AA-Request and AA-Answer), Credit-Control commands (Credit-Control-Request and Credit-Control-Answer) and Identity-Information commands (Identity-Information-Request and Identity-Information-Answer). Also there will be a client application and a server application. Since the server application requires user specific data to answer requests it also needs

some database backends for data storage. Therefore the Diameter implementation is split into the following packets:

diameterwebauth is the root package.

diameterwebauth.application holds generic Diameter WebAuth classes.

diameterwebauth.application.auth holds AA command related classes.

diameterwebauth.application.credit holds credit-control command related classes.

diameterwebauth.application.identity holds identity information command related classes.

diameterwebauth.client holds client implementation specific classes.

diameterwebauth.server holds server implementation specific classes.

diameterwebauth.server.db holds database classes for the server implementation

diameterwebauth.tests holds simple test applications to verify the client and server implementation.

This package hierarchy reflects a separation of client, server and the specific command concerns. It follows the modularity approach of object oriented programming and facilitates the maintenance of the software.

4.2.3. Message abstraction

A Diameter application from the programmers point of view can be seen as a very message-driven matter. The application logic on the client side as well as on the server side surrounds one or multiple message exchanges. According to the task which has to be accomplished, the Diameter client has to compose a Diameter message, send it and then react to the answer message. For the server application it is the reversed direction. The server receives a message, reacts to it, then composes an answer message and sends the answer. In order to ease the implementation, the first step therefore is it to implement an easy access to the Diameter messages from the application logic code. Indeed the underlying JDiameter implementation does allow direct access to the messages, more specific to the AVPs in a message. However, the access is rather cumbersome by decoding or encoding the actual data values by specifying the AVP code and data type everytime the value is read or written. To access grouped AVPs it is furthermore necessary to first specify the grouped AVP and then access the AVPs in a recursion.

Therefore a layer of abstraction on top of the JDiameter messages will be implemented. This DiameterMessage class will provide getter and setter methods for all message parameters by name and provide means to be encoded into and decoded from an underlying protocol message. It will also offer constants for AVP values which are of the enumerated type and command codes. By introducing the DiameterMessage class every application object can handle Diameter message by the usage of qualified instance methods and constants. This allows for a cleaner and more apparent access than using the generic methods which require numerical AVP codes and error handling everytime a value is accessed. For example application classes can now use code like

```
if (request.getAuthRequestType() == request.AUTHORIZE_AUTHENTICATE) {
    if (!checkPassword(request.getUsername(), request.getPassword())) {
        resultCode = DIAMETER_AUTHENTICATION_REJECTED;
    } else if (!checkAuthorization(request.getUsername(),
            request.getServiceIdentifier())) {
        resultCode =  DIAMETER_AUTHORIZATION_REJECTED;
    } else {
        resultCode = DIAMETER_SUCCESS;
    }

    ...

    answer.setUsername(request.getUsername());
    answer.setServiceIdentifier(request.getServiceIdentifier());

    ...

}
```

instead of

```
try {
    if (request.getAvps().getAvp(274).getUnsigned32() == 3) {
        try {
            username = request.getAvps().getAvp(1).getUTF8String();
            password = request.getAvps().getAvp(2).getOctetString();
            serviceId = request.getAvps().getAvp(439)
                    .getOctetString();
        catch (Exception e) {
            ....
        }
        if (!checkPassword(username, password)) {
            resultCode = 4001;
```

```
        } else if (!checkAuthorization(username, serviceId)) {
            resultCode = 5003;
        } else {
            resultCode = 2001;
        }

        ...

        if (username != null) {
            answer.getAvps().addAvp(1, username, false);
        }
        if (serviceId >= 0) {
            answer.getAvps().addAvp(439, serviceId);
        }

        ...

    }
} catch (Exception e) {
    ...
}
```

As shown by these two examples, the new message abstraction layer does make the code more readable and alleviates exception handling. It also conveniently checks for null values before encoding any data into a message. But the most important gain is that the actual AVP code values and datatypes are being encapsulated by the abstraction layer. Any application class using those abstracted messages does not need to be concerned with anything else but Java types, named attributes and named constants.

In order to adapt the DiameterMessage class to each application command, the basic class is implemented as an abstract class which need to be extended for each particular command. The parent class provides common constants for example for Diameter status codes. It also implements getter and setter methods for common attributes like the result code AVP. Furthermore it implements methods to encode and decode single AVPs into the actual Diameter message. Those encode and decode methods conveniently handle exceptions, null pointers and the conversion between Diameter data types and Java data types. The inheriting classes will add getter and setter methods for command specific attributes and constants. They will also extend the methods to encode and decode the complete message and to verify it. Figure 4.2 shows the hierarchy for the implemented message classes. Visible are the three classes AuthMessage, CreditMessage and IdentityMessage which each succeed the common DiameterMessage class in order to adapt them to the particular commands.

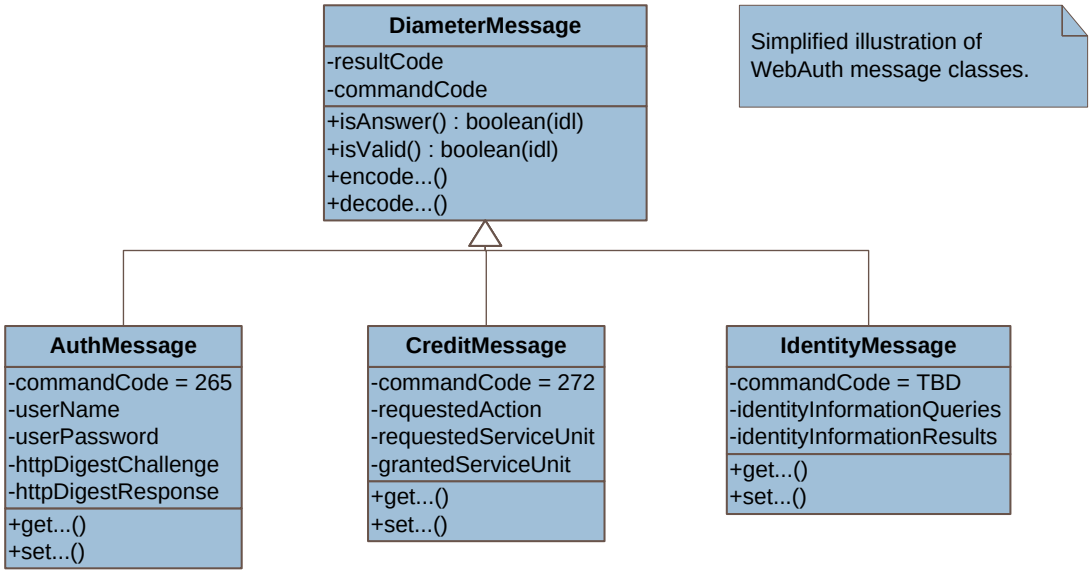


Figure 4.2.: Diameter WebAuth message classes

4.2.4. Server implementation

The Diameter server is responsible for accepting Diameter requests, process them and reply with a corresponding Diameter answer. The underlying JDiameter server implementation handles tasks like network access, bit-encoding and decoding of messages, transport and routing. Therefore the main task is to provide the server-side logical implementation of the Diameter WebAuth application.

Message handling

The actual server program is implemented in the WebAuthServer class. It is intended to be started from the command line and run as a background process. When the server process is started, it reads the configuration file and setups the Diameter stack accordingly. Then it will initialize the databases with the user data and register a handler with the Diameter stack. The handler is responsible for processing incoming Diameter requests with a matching Diameter application id. Different handlers can be registered in order to support different Diameter applications. The Diameter WebAuth handler itself uses three subhandlers for the three different command sets. Figure 4.3 shows the

class diagram of the server implementation.

If the server receives a message with its application id AVP set to Diameter WebAuth that doesn't belong to an existing session, it invokes the `processInitialRequest(...)` method of the registered ServerHandler. The ServerHandler then processes the request by examining the command code AVP value and delegating the request to the appropriate sub-handler. The sub-handler responsible for the request then processes the command and returns a DiameterMessage object. The message object is then encoded into a Diameter answer and handed-off to the Diameter stack for transport.

Authentication and authorization handler

Incoming AA-Requests are processed by the ServerAuthHandler. It examines the present AVPs and deducts the intend of the request. Then the appropriate action is taken in order to fulfil the request. The result is encapsulated into a DiameterMessage which then is returned.

A client can request user authentication or user authorization or both using a AA-Request.[1] Table 4.1 lists the possible actions the ServerAuthHandler class will perform.

First the handler processes the authentication part of the request. Depending on the present AVPs either a basic authentication or a digest authentication is performed. The digest authentication consists of a digest challenge and a digest response (cp. [18], Section 3.2.). If the request does not include a HTTP-Digest-Response AVP, the handler creates a digest challenge and puts it in the AA-Answer message. The answer is then send back to the client as part of a multiple round trip authentication. This means that the client has to provide further data in order for the server to be able to perform the authentication. If a HTTP-Digest-Response AVP is present in the AA-Request, the message is treated as a subsequent message in a multiple round authentication. The values of the HTTP-Digest-Response AVP can then be used to perform a digest authentication. If authentication was requested by the client and did not lead to a success, processing of a potential authorization request is skipped. This is done because a Diameter message is limited to a single result code. Therefore the authentication result gets precedence and its result code is returned to the Diameter client.

Processing the authorization part of an AA-Request is rather simple. The WebAuth specification uses the Service-Id AVP in context with an optional Service-Context-Id AVP to identify the service for which authorization is requested. If the service con-

[1] Actually, the WebAuth application also allows for requesting credit control actions or identity information in a AA-Request. But in those cases the requests are passed on to be processed by the credit control handler and identity information handler respectively. The corresponding handler than treats them as if they were intended for it.

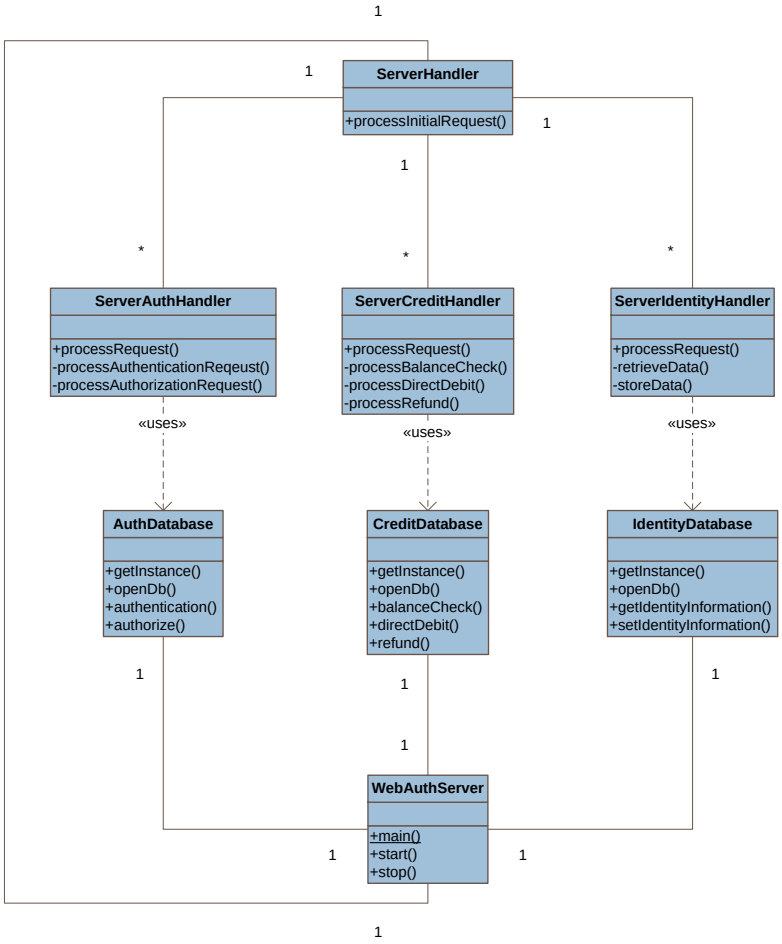


Figure 4.3.: Diameter WebAuth server implementation class diagram

Table 4.1.: ServerAuthHandler actions

Condition	Action performed	Result
Authentication actions		
User-Name and User -Password AVPs present	Basic authentication	SUCCESS or AUTHEN- TICATION REJECTED
HTTP-Digest- Response AVP present	Digest authentication	SUCCESS or AUTHEN- TICATION REJECTED
User-Password AVP and HTTP-Digest- Response AVP missing	Generate HTTP- Digest-Challenge	MULTI ROUND AUTH
Authorization actions		
Service-Id AVP present	Authorization with standard service context	SUCCESS or AUTHO- RIZATION REJECTED
Service-Id and Service- Context-Id AVP present	Authorization with specified service context	SUCCESS or AUTHO- RIZATION REJECTED

text is transmitted in the request, it is used together with the service identifier in the authorization process. Otherwise a standard service context is used. Authorization requests result in an AA-Answer with its result code set to DIAMETER_SUCCESS or DIAMETER_AUTHORIZATION_REJECTED, depending on the outcome of the authorization.

Credit control handler

The ServerCreditHandler class is responsible for processing incoming CC-Requests. The Diameter WebAuth application implements a small subset of the Diameter credit-control application [22]. These are the actions "Balance Check", "Direct Debiting" and "Refund" (cp. Section 3.2.2). They are called one time events, which means that they are handled in only one message exchange.

The implementation of the ServerCreditHandler class is rather straight forward. The Requested-Action AVP of an incoming CC-Request determines whetever the client requests a balance check, debiting a user or refunding a user. Then the appropriate attributes are evaluated and a request is made to the credit database. The database determines to what amount the request can be fullfiled. For example, if a client requests to debit a user for 100 credit units but the particular user only has an amount of 80 credit units available, only those 80 credit units are actually debited. The Granted-Service-Unit AVP in the CC-Answer reflects this amount of provided credits.

Identity information handler

II-Requests are processed by the ServerIdentityHandler class. Every II-Request contains a number of Identity-Information-Query AVPs. Each of these AVPs contains details about the requested identity attribute and is processed separately. The identity information handler iterates over the included Identity-Information-Query AVPs and passes the containing requests to the identity database. Then the II-Answer is populated with the corresponding Identity-Information-Result AVPs.

Backend databases

The user data required by the Diameter server to process incoming requests, is stored in a number of backend databases. The implementation includes three different databases for the three command sets. Those are the AuthDatabase class, the CreditDatabase class and the IdentityDatabase class. The reason to implement a different database for each command set was mainly a sense of modularity. Each of the commands requires different data and there are no intersections between them. The data for each database class is

```
users.xml:
    <webauth-users>
        <user name="bob" password="bobssecret" services="1, 2" />
        <user name="alice" password="alicessecret" services="2, 3" />
        ...
    </webauth-users>

credit.xml:
    <credit-users>
        <user name="bob" credit="2300" />
        <user name="alice" credit="1000" />
        ...
    </credit-users>

identity.xml:
    <identity-users
            identity-schema="myblog.key-value-pairs@id-provider.example.com" >
        <user name="Bob">
            <identity-information key="firstname" value="Bob" />
            <identity-information key="lastname" value="Bobber" />
            <identity-information key="title" value="Mr." />
            <identity-information key="color" value="#3366FF" />
            <identity-information key="email" value="bobb@example.com" />
        </user>

        ...
    </identity-users>
```

Figure 4.4.: XML database files

kept in XML files which are read when the corresponding database is instantiated. Figure 4.4 shows a short excerpt of each of the XML files. XML provides a fast and structured possibility to make data persistent. The XML data backends, however, are designed to be easily switchable for other backends. For example relational databases or directory services.

Besides getter and setter methods for single data atoms, the database classes also provide a number of methods to perform actions related to their data. For example the credit database encapsulates debit and refund actions in corresponding methods. This way other classes can conveniently perform more complex operations without having to handle exceptions, null values or synchronization issues everytime.

4.2.5. Client implementation

The counterpart to the server implementation is the implementation of a Diameter WebAuth client. Opposite to the WebAuthServer class, the WebAuthClient class is not intended to be run from the command line but to be used by other applications. The server applications is a stand along program, that is self-contained and runs in the background. The Diameter client application on the other hand provides access to Diameter functions for other programs. In this case the client is designed to provide web applications an interface to Diameter WebAuth operations. Those are authentication, authorization, credit control and identity information querying. The WebAuthClient class attends to initializing the underlying Diameter client stack, assembling the Diameter request messages and evaluating the answers send by the server. It provides a number of methods which encapsulate basic Diameter operations, the Diameter WebAuth specification offers. In the following some of those methods available for applications employing the WebAuthClient class are listed and explained.

```
public boolean basicAuthentication(String username,
                                   String password)
```

Sends an authentication request (AA-Request) to the Diameter server, requesting basic authentication for the given user with the given password. Returns `true` if the authentication with those credentials is positive, `false` otherwise.

```
public boolean authorize(String username, int service)
```

Sends an authorization request (AA-Request) to the Diameter server, requesting authorization for the given user and the given service in the standard service context. Returns `true` if the user is authorized for the service, `false` otherwise.

```
public Set<IdentityInformation> getIdentityInformation(
        String username,
        Set<IdentityInformation> identityInformation)
```

Sends an identity information request (II-Request) to the Diameter server. The request contains the given username and Identity-Information-Query AVPs for each of the IdentityInformation objects in the given set. Returns a `Set<IdentityInformation>` containing one IdentityInformation object for each of the Identity-Information-Result AVPs present in the answer.

```
public long directDebit(String username,
        long serviceUnits)
```

Sends a credit-control request (CC-Request) to the Diameter server, requesting to debit the given user the given amount of service units. Returns a `long` value containing the actual amount of units that were debited from the user's account.

```
public boolean balanceCheck(String username,
        long serviceUnits)
```

Sends a credit-control request (CC-Request) to the Diameter server, inquiring if the given user has the given amount of service units at his disposal. Returns `true` if the user's account covers the amount of units and `false` otherwise.

```
public long creditRefund(String username,
        long serviceUnits)
```

Sends a credit-control request (CC-Request) to the Diameter server, requesting to refund the given user the given amount of service units. Returns a `long` value containing the actual amount of units the Diameter server could refund to the user's account.

4.2.6. Test suite

To complement the implementation of the Diameter WebAuth client and server applications, a small test suite was also developed. The test suite is used to employ the WebAuth implementation under controlled conditions. It performs a multitude of Diameter WebAuth operations and evaluate the results. There are a number of test specified for each of the WebAuth command sets.

The test suite is implemented as a stand-alone application which uses the WebAuthClient class to act as a Diameter client. It performs its tests by using the methods the WebAuthClient class offers to query a Diameter server. However, the expected results of the requests are known in advance and compared to the actual results the client provides. By comparing the expected results to the actual results the test suite deems a test as successfull or as failed. Although there are complete unit testing frameworks available (e.g. JUnit [39]) for testing software components, developing a small proprietary test suite was preferred. It was found faster and more flexible to implement than employing a unit testing framework. For example, the test suite evaluates the time it took the process the test case by the Diameter system as well. Also it is possible to add performance or stress tests to the test suite when deemed necessary.[2]

[2] A performance evaluation was not performed within the scope of this thesis (cp. Section 5.3). Therefore that part of the test suite was not implemented.

An extract of a simple test case is shown below. It tests the basic authentication mechanism first with valid and then with void credentials. In the first case a successfull authentication is expected and in the second case a failed one. If either the basic authentication with valid credentials fails or the one with void credentials is successfull the corresponding test case is considered failed. It also runs tests to make sure, non existent users are properly rejected and that null values don't produce unwanted null pointer exceptions in the implementation.

```
public void runTests(WebAuthClient client) {
    String goodUser = "bob";
    String goodPass = "bobssecret";
    String badUser = "__xxxxxxxxxx__";
    String badPass = "__xxxxxxxxxx__";
    boolean success;

    startRunTests("AA commands verification tests");

    success = client.basicAuthentication(goodUser, goodPass);
    addResult("Basic authentication with valid user and valid pass",
        success, client.getLastRuntime());

    success = !client.basicAuthentication(goodUser, badPass);
    addResult("Basic authentication with valid user and void pass",
        success, client.getLastRuntime());

    success = !client.basicAuthentication(badUser, badPass);
    addResult("Basic authentication with void user and void pass",
        success, client.getLastRuntime());

    success = !client.basicAuthentication(null, badPass);
    addResult("Basic authentication with null user and void pass",
        success, client.getLastRuntime());

    success = !client.basicAuthentication(goodUser, null);
    addResult("Basic authentication with good user and null pass",
        success, client.getLastRuntime());

    stopRunTests();
}
```

The addResult method is used to collect all the results and to print the details if requested. In the end of a test run a detailed report including a statistic is produced. The report for the example above looks approximately like this:

```
Running tests 'AA commands verification tests':
Basic authentication with valid user and valid pass ... (6ms): PASSED
Basic authentication with valid user and void pass ... (4ms): PASSED
Basic authentication with void user and void pass ... (3ms): PASSED
Basic authentication with null user and void pass ... (6ms): PASSED
Basic authentication with good user and null pass ... (4ms): PASSED

AA commands verification tests statistics:
  Number of tests performed: 5
  Number of tests passed   : 5
  Total runtime of tests   : 23ms
  Average runtime of tests : 4ms
```

4.3. Web application

To actually deploy the Diameter WebAuth framework in a web application, a small web blog application called MyBlog was developed. The MyBlog application also serves as a showcase the WebAuth framework and to validate the Diameter application in Section 5.2.

4.3.1. MyBlog application

As model for the web application a blog was chosen. A blog is a simple website where messages are posted and usually displayed in a chronological order. The web application, called MyBlog, implements a number of features which are used to demonstrate the possibilities of the WebAuth framework. To implement the MyBlog application, the JavaServer Pages (JSP) technology [56] was chosen. The JSP technology enables the efficient and quick development of web applications. It also provides access to Java objects which is used to connect the web application to the Diameter WebAuth client. The main page of the MyBlog application is shown in Figure 4.5. The messages are displayed in the center, together with a couple of informations about the person who created the message and the time of its posting. Right below the title is a navigation bar which allows access to the subpages of the application.

Each subpage attends to a certain task. For example the `login_basic.jsp` page handles user login using the basic authentication. Table 4.2 shows a listing of the implemented pages and their particular tasks. The goal of the MyBlog application is to showcase the features of the Diameter WebAuth framework. It should be understood as such a showcase rather than as a serious blog application.

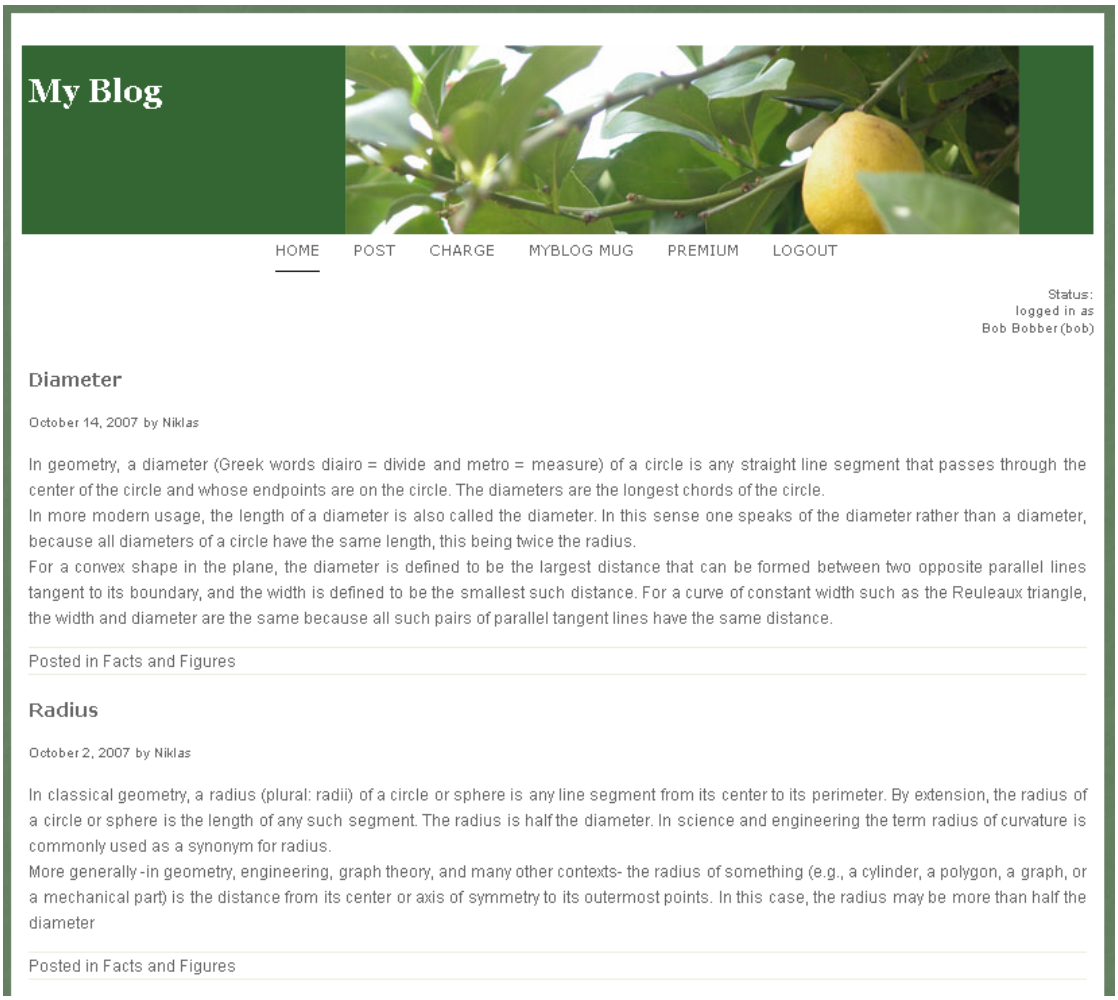


Figure 4.5.: MyBlog: Main page

Table 4.2.: MyBlog: JSP pages overview

JSP	Task
index.jsp	Shows main page with the posted messages.
buy_mug.jsp	Allows the user to buy the MyBlog mug.
charge.jsp	Allows the user to charge his account with credits.
login_basic.jsp	User login with basic authentication.
login_digest.jsp	User login using HTTP-Digest authentication.
login_form.jsp	User login using a custom form and plain text credentials.
logout.jsp	User logout.
post.jsp	Post a new blog message if the user is authorized to do it.
premium.jsp	Allows access to premium content if the user has enough credit.

4.3.2. JSP subpages

Most of the application logic is encapsulated in the JSPs. Besides offering XML-like tags with predefined functions, the JSP technology allows to mix HTML and Java code. This makes it well suited for rapidly developing web applications. If a user accesses a JSP with his web browser, the embedded code is executed by the web server and the result is send to the web browser. Usually the result is a HTML document that the web browser than presents to its user. However, a JSP also offers facilities to influence HTTP headers send from the web server to the user client. This allows for example to implement a HTTP digest authentication within a JSP.

The typical design of a MyBlog JSP is as follows. First, variables within session or application scope are declared. The next section contains Java code with the page logic. This code component is responsible for processing the request. It examines user input, conducts the necessary operations and deducts the page result depending on the outcome of these operations. Processing results and information that needs to be conveyed to the user are saved in page variables. In the end of the JSP is the HTML code located which produces the page result in a HTML document. The variables set by the page logic are used to adjust the output to the processing results. Figure 4.6 for example shows four different results for the digest login JSP. Depending on the outcome of the page operation, the user gets a different result document presented.

4.3.3. Helper classes

A JSP is also capable of storing Java objects outside its own page scope. This allows to maintain statefull objects and to efficiently reuse objects on a higher scope. For example,

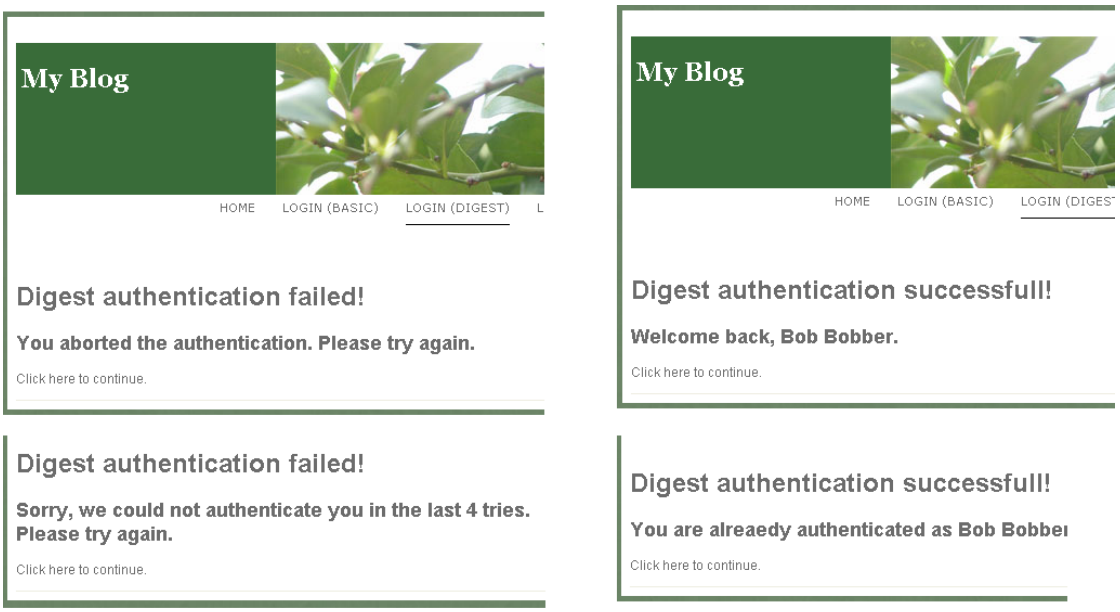


Figure 4.6.: Different results from the login JSP

every JSP in the MyBlog application accesses a user object which is maintained on a user session scope. This object allows each subpage to set and retrieve user related information. Such objects are common Java objects which are instantiated from Java classes.

Besides the User class which represents a single web application user, the MyBlog application also a class called WebAuthHelper to execute Diameter WebAuth functions. The WebAuthHelper class bridges the gap between the Diameter WebAuth client class and the MyBlog web application. It provides the JSP subpages with an interface to the WebAuthClient class that considers the web application environment. For example, it provides authentication methods that encode and decode authentication information into HTTP headers and take on the HTTP authentication exchange with the user client. As a result, the JSPs can request authentication from the helper and get a simple true or false as return value. Handling the Diameter server on the one side and the user client on the other side is done transparent to the JSP by the WebAuthHelper.

4.4. Summary

This chapter detailed the implementation work done on the Diameter WebAuth framework.

- A Diameter WebAuth server, a Diameter WebAuth client and a web application were implemented to substantiate the proposal.
- The server implementation uses dedicated classes to handle incoming messages of the three different command sets:

  the ServerAuthHandler class processes authentication and authorization requests (AA-Requests),

  the ServerCreditHandler class is responsible for credit-control requests (CC-Requests) and

  the ServerIdentityHandler class handles identity information requests (II-Requests).

- The client implementation provides methods for external applications to conveniently perform Diameter WebAuth operations. For example, this allows applications to use the method call `basicAuthentication("bob", "secretpass")` to execute a basic authentication via the Diameter WebAuth framework.

- A simple web blog application called MyBlog was implemented to showcase the framework.
- There is also a small test suite available to test and verify the implementation.

5. Evaluation

After presenting design and implementation of the Diameter WebAuth framework in the previous chapters, the approach will now be evaluated. The first part of the evaluation is a short verification of the Diameter server and client implementation. It will be followed by a validation using the requirements that were posed on the framework during the design phase. Then a couple of considerations concerning the overall performance of the framework will be made. And finally the Diameter WebAuth proposal will be compared to other web-based identity management approaches.

5.1. Implementation verification

During the implementation and after its completion, the test suite (cp. Section 4.2.6) was used for continuous testing and verification. The intention of software verification is to ensure that the developed software satisfies it requirements. This means that the software should behave as described in the specifications [62]. The software operations should yield the specified results and the software should run without unexpected errors or exceptions.

The test suite contains numerous tests that compare the results of different methods in the Diameter WebAuth client implementation. This is a dynamic verification approach and can be considered an integration test [3, 62]. Integration tests cover multiple modules of a system, in this case, several components of the Diameter client and server implementation. For example, the test cases for a particular command set, address the encapsulating client methods, the particular message implementation, the client message handling subsystem, the transport system, the server message handling system and the particular server backend functions.

The finished Diameter WebAuth implementation passed all the conducted verification tests. An example of the result statistics is shown in Figure 5.1.

```
AA commands verification tests statistics:
  Number of tests performed: 14
  Number of tests passed   : 14
  Total runtime of tests   : 65ms
  Average runtime of tests : 4ms

CC commands verification tests statistics:
  Number of tests performed: 12
  Number of tests passed   : 12
  Total runtime of tests   : 68ms
  Average runtime of tests : 5ms

II commands verification tests statistics:
  Number of tests performed: 15
  Number of tests passed   : 15
  Total runtime of tests   : 77ms
  Average runtime of tests : 5ms
```

Figure 5.1.: Verification test result statistics

5.2. Design validation

Validation means to establish that a software system meets its requirements and fulfils its intended purpose (cp. [62]). As basis for the validation of the Diameter WebAuth the framework, the use cases specified in the design phase (cp. Section 3.1.2) are chosen. Because those use case were a starting point in the initial design of the framework, it makes sense to compare the implementation results against them. In this section, therefore, a number of selected use cases, covering each of the Diameter WebAuth command sets, will be played through. This way it can be established that the framework indeed meets the requirements and purposes which were underlying to its design.

5.2.1. Authentication and authorization

The use cases involved in the authentication and authorization demonstration are:

- "A web site wants to authenticate a user."
- "A web site wants to authorize a user for a specific service."

The first exercise for the MyBlog application is to authenticate a user in order to establish his identity. It is assumed, that the user is known to the identity provider

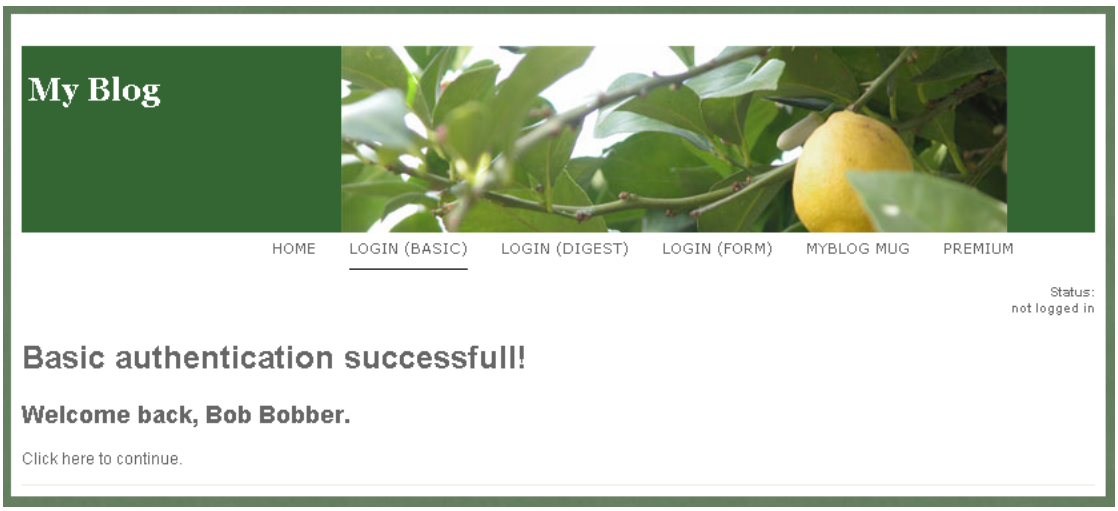


Figure 5.2.: MyBlog: Login successfull

(respectively the Diameter server) and that the user supplies his correct credentials. The Diameter WebAuth application provides means to perform a basic authentication or a http digest authentication via the Diameter protocol. Basic authentication means that a cleartext password is checked against the username, while the digest authentication employs a one-way hash function to secure the clear text credentials. The WebAuth demo application uses the WebAuth client on its login page to access the Diameter authentication facilities. When a user accesses the login page, he gets promted for his username and password. After the user entered his credentials, the web application passes them to the Diameter client. The client uses the Diameter AA commands to perform the authentication and returns a true or false value to the web application. As a result the web application knows whether the user could be authenticated or not and can finish processing the login page accordingly. Figure 5.2 shows the composed login page in case of a successfull authentication.[1]

After the user is identified as "bob," he wants to make a new post in the blog. The MyBlog application, however, poses some restrictions on the users allowed to post new messages. Only a number of approved users are allowed to do so[2]. The web application

[1] The personalized greeting is the result of a WebAuth identity information operation, which is covered in another use case (see below).

[2] How a user gets approved to post is outside the scope of this demonstration.

provider agreed with the identity provider to establish the service "2" within the standard service context as classification for users that are allowed to post on the blog. The web application, therefore, attempts to authorize the user "bob" for the service with the identifier "2" using the `authorize` method of the Diameter WebAuth client (cp. Section 4.2.5). The method returns a `true` value if the user is authorized for the "make blog posts" service in which case the application shows the post page. If the user is not authorized, an error page explaining the situation is shown.

5.2.2. Accounting

The use cases involved in the accounting demonstration are:

- "A web site wants to charge a user for a specific service."
- "A web site wants to credit a user for a specific service."
- "A web site wants to check if a user has a certain credit."

To showcase the credit-control features of the Diameter WebAuth framework, the the MyBlog web application sells the MyBlog mug as merchandise. On the "MyBlog Mug" page, shown in Figure 5.3, a user can order any number of MyBlog mugs. Since the prototype implementation only supports service specific units, one MyBlog mug is charged with 1500 "credits." The processing of the transaction is handled through the Diameter client, that sends a credit-control request to the Diameter server. The request contains a credit-control event request for the direct debit event and the value of requested service units. After the servers credit-control backend handled the request, the Diameter server will send a CC-Answer back to the WebAuth client which contains the amount of service units that actually could be charged to the user's account. If the the amount that was granted is the same amount that was requested, the number of ordered mugs is sold to the user. If it isn't, the application tries to rollback the transaction by refunding the amount that was granted. In any case, the user gets an adequate response.

To charge an account, the user can use the charge page. For the sake of the showcase implementation, this page just lets the user charge a certain amount of service units to his account. How the identity provider clears the charge with the user in the end, doesn't concern the web application provider and is not reconsidered here. Access to the "premium content" of the web application is intended only for user that have more than 10.000 service units available on their credit account. The web application, therefore, conducts a Diameter WebAuth balance check for 10.000 requested service units before it grants access to a user. Whatever the premium content may be, is not further considered in the showcase application.

Figure 5.3.: MyBlog: Buy mug page

5.2.3. Identity attributes

The use cases involved in the identity attributes demonstration are:

- "A web site wants to retrieve user specific identity data."
- "A web site wants to store user specific identity data."

Whenever a user makes a new blog post, the web application includes his full name and his email address in the post. Those identity attributes are fetched from the identity provider using a Diameter WebAuth identity information query. It is assumed, that the web application provider and the identity provider have agreed to use a private identity information scheme that uses simple key-value pairs. The schema is designated "myblog.key-value-pairs@id-provider.example.com" and specifies that an Identity-Attribute-Request AVP has to contain a keyword and is answered with an Identity-Attribute-Value AVP that contains the corresponding value. For example the query for "lastname" will fetch the value "Bobber" for the user "bob." Using this simple identity information scheme, the web application fetches the first name, the last name and the email address for every user when he makes a post. The values are stored and displayed together with the posting as shown in Figure 5.4.

5.3. Performance considerations

A dedicated performance evaluation is not intended for the Diameter WebAuth implementation. There are a number of reasons for that. First of all, the implementation is only a prototype implementation. It is intended as proof of concept work for demonstrating the proposed approach and thus was not implemented with an objective of optimized performance. Second, the implementation uses an external underlying Diameter implementation (cp. Section 4.2.1) which forms the major source of performance limitation, and the Diameter WebAuth application is just a slight add-on upon this implementation from the performance perspective. Third, performance does not seem to be an important issue in regards to identity management. Since it is a human user that is usually the subject of identity operations such as authentication or an automated login, the tolerable delay is comparatively high. Certainly, the approach itself should be scalable to be able to handle any number of concurrent users. In case of the Diameter protocol, this assumption is considered as true based on the fact that Diameter is widely deployed, including in large networks.

Furthermore, other identity management approaches introduced in this thesis are intended to operate in networks too. They all use message transfers over IP networks and

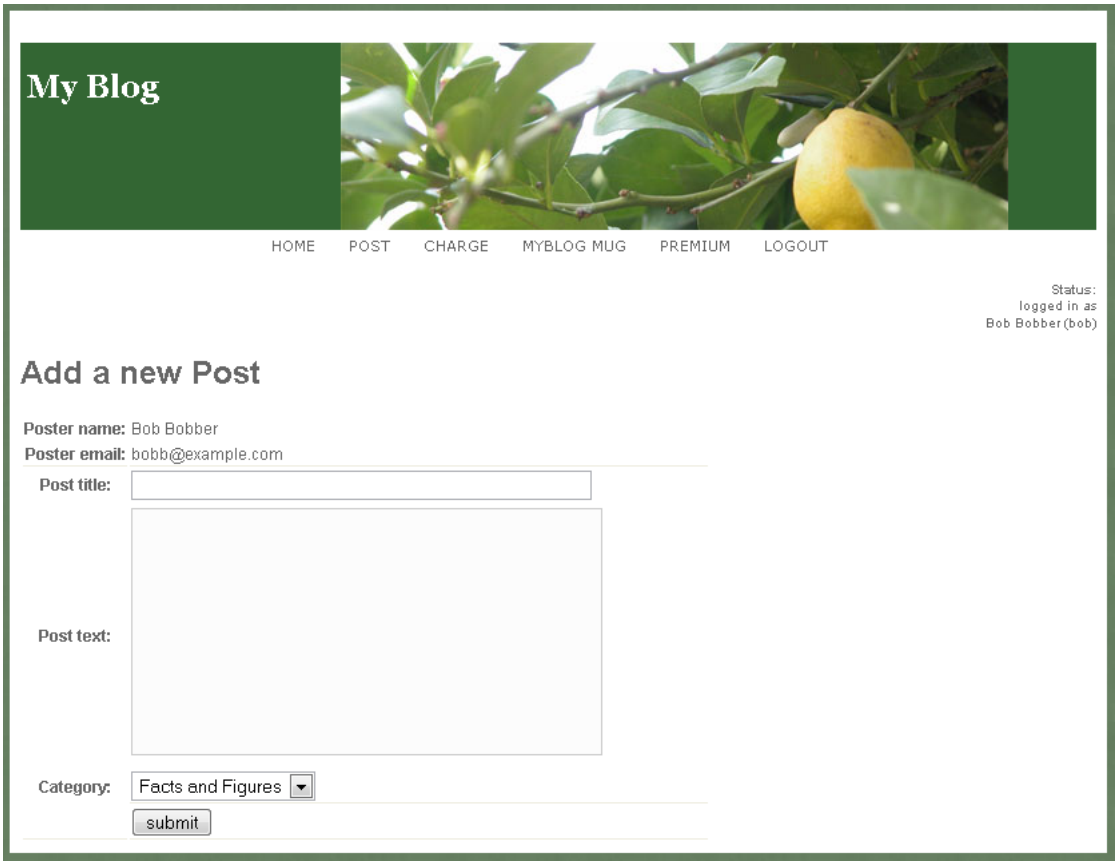


Figure 5.4.: MyBlog: Posting form with filled in identity information

don't operate locally without a network. Therefore, network charges and restrictions apply to all of them, including Diameter WebAuth. For future work regarding Diameter WebAuth, however, it might be interesting to compare different approaches with regard to some performance metrics like the number of message exchanges, the average data transfered or the number of user interactions necessary to process a certain operation.

5.4. Feature comparison

An important demand made on the design of Diameter WebAuth was that its features will be comparable with other web-based approaches to identity management. This section will now evaluate the features of Diameter WebAuth using the same criteria that were applied during the evaluation of the approaches in Section 2.6. The result of this evaluation will be used to directly compare Diameter WebAuth to the approaches to identity management in web applications introduced in Section 2.

End user authentication The Diameter WebAuth specification covers end user authentication using the HTTP authentication methods specified by RFC 2617. Those are basic authentication using cleartext credentials and digest authentication which uses a one-way hash function to protect the user's password. By implementing those methods, Diameter WebAuth provides username/password authentication which is propably the most common method used by web applications at the moment. HTTP digest authentication furthermore allows safe handling of the user's password even by the Diameter client without exposing its cleartext value.

As shown by the prototype implementation, Diameter WebAuth provides direct authentication support for web applications using either browser provided mechanisms (HTTP basic and digest access authentication) or form-based authentication. The later method allows the web application to use a completely customizeable login page[3]. Furthermore, Diameter WebAuth can be extended without any difficulty to support various other authentication methods by updating the specification.

Authorization Service differentiated user authentication is provided by Diameter WebAuth in order for web applications to support areas with different access restrictions. The particular service identifiers are freely allocable between the Diameter server provider and the web application provider. This allows for user authentication as fine grained as needed by the web application.

[3] Opposite to http digest access authentication, however, form-based authentication does not protect the password from exposure to the web application.

Single sign-on/sign-out Although single sign-on/sign-out is not implicitly supported by Diameter WebAuth, the HTTP digest access authentication scheme used, allows to define a protection space which can span multiple servers. This mechanism allows the end user client to determine the set of URIs for which the same authentication information may be used [18, Section 3.2.1.]. This can be used by the Diameter server to implement a crude single sign-on functionality for the Diameter clients it is responsible for. Also, HTTP authentication allows for the authentication header to be included in every request to a resource within the protection space. If a client omits the header, the application can understand this (if necessary after issuing another authentication request) as a logout. This would allow for a user client-based support of single sign-on and sign-out. A system immanent single sign-on/sign-out functionality can be realized with the support of another Diameter application which maybe the subject of future work (cp. Section 7).

Accounting Basic credit related functions are available in Diameter WebAuth as credit control operations, carried over from the Diameter Credit-Control application. They allow a web application to charge and refund its users with currency or application related service units. If and to what extend the Diameter service provider is taking monetary responsibilities for its users needs to be arranged between the web application provider and the Diameter server provider.

User client support The Diameter WebAuth application does not make any fundamental demands on the end user client. Only the HTTP authentication methods used by Diameter WebAuth need to be supported by the client. But first of all, they are more related to the HTTP protocol which support is a basic requirement for every web browser. This means, that Diameter WebAuth rather uses basic features of its end user's service protocol than expecting exceptional and incoherent traits. And second of all, even in the case HTTP authentication is not available in the end user's client, Diameter WebAuth can fallback on form-based authentication.

Technologies Since Diameter WebAuth is implemented on top of the Diameter Base Protocol, it employes the same technologies. Those are IP-based protocols like TCP, SCTP, TLS and IPSec. It is also valid to list HTTP as dependant technologies, because the authentication part of Diameter WebAuth relies on HTTP mechanisms.

Identity attributes Identity attribute exchange is supported by Diameter WebAuth by the implementation of the Identity-Information commands (cp. Sections 3.2.3, 3.3.5

and 3.3.6). They allow a web application to query the Diameter server for attributes describing a persons identity. The extend of the available attributes, their format and the query/response syntax to exchange them is not fixed and needs to be predefined between the web application provider and the Diameter server provider. This allows to use virtually any identity information schema with Diameter WebAuth.

Maturity level Although Diameter WebAuth itself is a new approach to identity management, it employs the Diameter protocol. The Diameter protocol is well established and mature. This means, that although the Diameter WebAuth framework can only be attributed a very low maturity level at the moment, it can be expected that its maturity can be advanced comparatively fast. This results from the facts that Diameter is a common technology and the WebAuth application is designed with regards to easy implementation.

Primary focus Diameter WebAuth focusses on bringing network-based AAA mechanisms into the realm of web applications. The intention is to provide means for web applications to use AAA infrastructures for identity management purposes. This is done by adapting network-based technology to be used in the application layer of web environments. Furthermore, the identity management aspect is emphasized by the means to transport identity information in addition to the classical AAA tasks.

5.4.1. Results

Table 5.1 shows a feature comparison of Diameter WebAuth with web-based identity management approaches. The result of that comparison is that Diameter WebAuth is able to offer most of the features that can be demanded from an identity management approach. Solely the support for single sign-on/sign-out can be enhanced, compared to solutions like OpenID and Liberty. However, OpenID and Liberty were designed with a special focus on single sign-on mechanisms. Compared to the Microsoft CardSpace technology, Diameter WebAuth does not rely on the support of the user client. This makes it easier to deploy and to maintain.

Because Diameter WebAuth also explicitly covers end user authentication it can be considered much more versatile than the OpenID and Liberty approach. Unlike those approaches, Diameter WebAuth can be used as an authentication backend. This makes it viable in scenarios where no identity management but local authentication is the primary task to accomplish. For example, in the case where a company-internal webserver should be enabled to authenticate users against he existing AAA infrastructure. The authenti-

Table 5.1.: Feature comparison for identity management systems with Diameter WebAuth

Feature	OpenID	Liberty	CardSpace	Diameter WebAuth
Authentication	no	no	yes	yes
Authorization	no	yes	yes	yes
Single sign-on/	yes/	yes/	no/	depends[1]/
sign-out	no	yes	no	depends[1]
Accounting	no	no	no	yes
Requires user client support	no	no	yes	no
Technologies	HTTP	SAML	XML	IP, HTTP
Identity attributes	no	yes	yes	yes
Maturity	medium	medium	low	low[2]
Primary focus	Decentralized identities (Web)	Trust relations (Web)	Authentication (Web)	AAA + identity information

[1] HTTP Digest Authentication can provide some crude SSO mechanisms.
[2] Diameter WebAuth is based on the Diameter protocol which is well established and mature.

cation capabilities of Diameter WebAuth also allow for it to be employed to complement identity management approaches that do not cover authentication.

Unique selling points of Diameter WebAuth are that, because it superimposes on the Diameter protocol, it seamlessly integrates in existing AAA infrastructures and that it supports accounting mechanisms. While the later might not yet be excessively interesting for most web applications, the utilization of existing AAA structures, especially existing Diameter servers, brings a number of advantages. It reduces expenses and maintenace effort, avoids data redundancies or discrepancies and allows to consolidate know-how. Especially for organizations that provide different access controlled services, this makes Diameter WebAuth a very attractive approach to expand existing network-based identity management to web applications.

5.5. Summary

In this chapter the previous work done in the thesis was evaluated.

- The implementation of the Diameter WebAuth application was verified using a dedicated test suite.

- The design of the Diameter WebAuth framework was validated on the basis of specified use cases using the MyBlog web application.
  - It was shown how a web application can use Diameter WebAuth to authenticate and authorize its users.
  - The credit-control facilities were demonstrated by means of a merchandise sale included in the MyBlog application.
  - Throughout the application were identity information used to personalize the content.
- Performance optimization was not an objective of the implementation and a performance evaluation was, therefore, not made.
- The Diameter WebAuth framework was compared to other approaches for identity management in web applications.
  - The result was that Diameter WebAuth is compareable regarding the features to other approaches to identity management.
  - In addition, the proposal includes end user authentication specifications and accounting facilities.
  - Opposite to approaches like OpenID or the Liberty Alliance project, Diameter WebAuth, therefore, can be employed in authentication backend systems as well.

6. Conclusion

The more personal and personalized information are available on the Internet, especially the World Wide Web, the more popular identity management solutions will become. They offer the application providers a convenient way to handle user authentication without local authentication databases and they avoid tedious sign-up procedures as entrance barriers to their services. The users benefit from simplified login procedures with one set of access credentials and having the identity provider as trustworthy party to govern their account data. The proposed Diameter WebAuth is an AAA-based identity management framework that has been developed with the same requirements that are made to web-based solutions. It closes the gap between network authentication and application authentication by effectively bringing network-based access control concepts to the application layer. WebAuth it is based on the well established and mature Diameter protocol. It, therefore, benefits from the propagation of Diameter setups and the general experience with the protocol in terms of deployability implementability and maintainability.

The work presented in this thesis is suited to enable any Diameter system to be part of an identity management system. Especially network providers, which already operate Diameter setups can easily extend their systems to offer more services and consolidate their infrastructure respectively. Particularly in closed networks, Diameter WebAuth can also be established as common authentication method to implement a central and secure authentication system that is not limited to the web-environment. Since not all identity management solutions include authentication protocols, even identity providers offering service for other approaches, like OpenID or the Liberty Alliance, can employ Diameter servers for their backend authentication using the Diameter WebAuth proposal.

Authentication and authorization

Diameter WebAuth implements three functional facilities: authentication and authorization, credit control and identity attributes. The authentication and authorization functions of Diameter WebAuth include end user to identity provider authentication as well as the identity provider to application provider confirmation process about a successfull authentication. This distinguishes it from other web-based identity management

solutions that only cover one of those two aspects.

As result, Diameter WebAuth is a more holistic approach to identity management. More importantly it gives Diameter WebAuth the ability not only to substitute other approaches but also to complement them. For instance, Diameter WebAuth can be deployed as mechanism by OpenID identity providers to authenticate OpenID users. Also, it should be possible to extend Diameter WebAuth to support Microsoft CardSpace as an authentication method. This extent makes Diameter WebAuth a very versatile employable approach.

Credit control

In addition to the network-based access control, Diameter WebAuth also brings network-based credit control mechanisms to the web application layer which further unifies those two layers. The credit control functions allows to conduct charging operations over the same infrastructure that is used for identity management. This makes it easy to offer payable services or products even for application providers that only sell services or products in small quantities sporadicly. If a web application supports Diameter WebAuth it virtually takes no additional effort to also employ it for credit operations.

For the end user, credit control operations over the Diameter WebAuth framework have the advantage that he doesn't need another provider for his financial transactions. A fundamental idea of identity management is to make handling of identity aspects easier by integration. Diameter WebAuth consistently expands this concept to also include financial transactions to user's identities.

Identity attributes

Authentication and authorization as well as credit control operations in Diameter WebAuth are concepts that originate from the network layer and are adapted to the application layer. Identity attributes on the other hand, are a classical identity management concept that emerged from the application layer. Undoubtful, those attributes are an important aspect in identity management to provide further information about a person's identity. It, therefore, is not just an end in itself to support them in Diameter WebAuth. By doing so, however, the identity attribute facilities are made generally available on Diameter infrastructures. This means that Diameter WebAuth not only brings network-based functions into the application layer but also adapts concepts from the application layer to the network layer. Possible applications are to personalize network access control mechanisms or to transfer personal information like address books or schedules to devices that are authenticated using the Diameter protocol.

7. Future work

The work done in this thesis has initiated a number of topics that are closely related to the thesis but couldn't be explored further due to time constrains and the desire to hold up a primary focus. They are well suited to further advance the work in the scope of this thesis or to complement it.

Extending authentication capabilities

The integration of end user authentication capabilities is a property of Diameter WebAuth that distinguishes it from other approaches. The reason, end user authentication is included in the proposal, is that in a Diameter WebAuth setup the authentication credentials have to be transported using the Diameter protocol in order to reach the Diameter server. The details of the authentication operation, therefore, need to be specified in order to ensure interoperability between Diameter client and server. Furthermore, the specification of the supported authentication methods helps to choose those methods that are suited for the web environment.

Since the authentication capabilities are such an important part of Diameter WebAuth, further work should consider to extend these capabilities. Primarily this means to specify and implement additional authentication methods. Very suitable for that are authentication methods that are already widely supported by web browsers, for instance user certificate-based authentication. It also seems worthwile to explore the possibilities to integrate new authentication methods like Microsoft CardSpace. To enhance the available authentication methods it might be worth considering to add a number of AVPs to the specification to allow for some advanced features like the support of CAPTCHAs[1] or pre- and post-authentication messages. Especially in setups that use form-based authentication such features can provide additional functionality to make Diameter WebAuth more suitable for such scenarios.

[1] "CAPTCHA" is a contrived acronym for "**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part" [12]. Very common are small pictures containing a confirmation code that cannot easily be read by a computer.

User feedback channel

A disadvantage, Diameter WebAuth has compared to several other identity management approaches is that it does not provide a direct communication channel between the identity provider (Diameter server) and the end user. All communication is relayed through the Diameter client. This means that the Diameter server cannot directly get confirmation from or send feedback to the user. Having such a feedback channel between the user and the Diameter server would most likely enhance security, privacy and overall suitability of Diameter WebAuth. The channel would allow to inform the user of ongoing operations or to request explicit user approval.

An objective of future work could be to establish the requirements for such a direct feedback channel and to develop a proposal to implement it. If end-to-end security and confidentiality can be achieved for this feedback channel, it might even be possible to use it for user authentication. This could effectively free Diameter WebAuth from all restrictions concerning authentication schemes and let the Diameter server handle authentication directly.

Single sign-on

As stated in Section 2.6, single sign-on is a valuable feature for identity management systems in general. The Diameter protocol on the whole lacks of single sign-on facilities. Future work, therefore, consider to develop a Diameter single sign-on solution. Either specifically for web environments or with a more general pretense. In the later case, a SSO solution could advance the Diameter protocol at large.

Performance evaluation of identity management approaches

Evaluating the performance of identity management approaches for web applications is not particularly straightforward. The approaches have different capabilities and use different techniques. Also the quantity to be measured is not implicitly clear. Is packet size, the number of message exchanges, server response times or the level and number of user interactions decisive? As a matter of fact, the issue whether this even is an interesting topic for future work needs to be assessed more extensive. It is feasible that an appropriate performance comparison of different approaches just cannot be done, because there is not enough common ground between them.

HTTP authentication

RFC 2617 which specifies the HTTP basic and digest access authentication methods dates from June 1999. Today other authentication methods, especially the HTML form-based authentication seems more and more popular. The biggest advantage of form-based authentication is that it allows unlimited adaptability of the login mask presented to the user. Its biggest disadvantage is that the login credentials are transmitted in cleartext. The HTTP digest authentication on the other hand is exactly reversed: it does not allow for any customization of its layout whatsoever but secures the authentication credentials.

Besides this most obvious flaw, its unadaptable layout, HTTP authentication has a number of additional problems that could be the subject of future work. For example, RFC 2617 does not specify any kind of logout facility or how to handle usernames or passwords in different character sets. Also it maybe worth to explore additional security considertations against phishing attacks or man-in-the-middle attacks. Especially the integration of some kind of server authentication seems to be interesting to solve a number of security issues with HTTP authentication. Last but not least, the behavior of most web browsers not to distinguish between the inherently insecure basic authentication and the reasonably more secure digest authentication is a security concern that should be addressed. Different security settings, perhaps even allowing the user to disable basic authentication under certain circumstances (for example, no SSL secure connection), and a noticeable, visible distinction between the two methods seem suited to increase the security of HTTP authentication on the client side.

Bibliography

[1] B. Aboba, P. Calhoun, S. Glass, T. Hiller, P. McCann, H. Shiino, G. Zorn, G. Dommety, C.Perkin, B.Pati, D.Mitto, S.Mannin, M.Beadle, P.Wals, X.Che, S.Sivalingham, A.Hamee, M.Munso, S.Jacob, B.Li, B.Hirschman, R.Hsu, Y.Xu, E.Campell, S.Baba, and E.Jaques. Criteria for Evaluating AAA Protocols for Network Access. RFC 2989 (Informational), November 2000. URL `http://www.ietf.org/rfc/rfc2989.txt`.

[2] B. Aboba, M. Beadles, J. Arkko, and P. Eronen. The Network Access Identifier. RFC 4282 (Proposed Standard), December 2005. URL `http://www.ietf.org/rfc/rfc4282.txt`.

[3] Alain Abran, James W. Moore, Pierre Bourque, Robert Dupuis, and Leonard L. Tripp. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE, 2004.

[4] bandit-project.org. Welcome to Bandit - Bandit-project.org. URL `http://www.bandit-project.org/index.php/Welcome_to_Bandit`.

[5] Marco Barulli and Giulio Cesare. Clipperz Crypto Library, 2007. URL `http://code.google.com/p/clipperz/`.

[6] Alan Berg and Bas Toeter. Single Sign on and Single Sign Off in a non homogeneous Portal front ended environment, 2005.

[7] Stefan Brands. The Identity Corner - The problem(s) with OpenID. URL `http://www.idcorner.org/?p=161`.

[8] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. Diameter Base Protocol. RFC 3588 (Proposed Standard), September 2003. URL `http://www.ietf.org/rfc/rfc3588.txt`.

[9] P. Calhoun, T. Johansson, C. Perkins, T. Hiller, and P. McCann. Diameter Mobile IPv4 Application. RFC 4004 (Proposed Standard), August 2005. URL `http://www.ietf.org/rfc/rfc4004.txt`.

[10] P. Calhoun, G. Zorn, D. Spence, and D. Mitton. Diameter Network Access Server Application. RFC 4005 (Proposed Standard), August 2005. URL http://www.ietf.org/rfc/rfc4005.txt.

[11] Scott Cantor, Jeff Hodges, John Kemp, and Peter Thompson. Liberty ID-FF Architecture Overview, 2005.

[12] Carnegie Mellon University. CAPTCHA: Telling Humans and Computers Apart Automatically, 2000-2007. URL http://www.captcha.net/.

[13] M. Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501 (Proposed Standard), March 2003. URL http://www.ietf.org/rfc/rfc3501.txt. Updated by RFCs 4466, 4469, 4551, 5032.

[14] D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. RFC 4234 (Draft Standard), October 2005. URL http://www.ietf.org/rfc/rfc4234.txt.

[15] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence. Generic AAA Architecture. RFC 2903 (Experimental), August 2000. URL http://www.ietf.org/rfc/rfc2903.txt.

[16] V. Fajardo, T. Asveren, H. Tschofenig, G. McGregor, and J. Loughney. Diameter Applications Design Guidelines, October 2007. URL http://www.ietf.org/internet-drafts/draft-ietf-dime-app-design-guide-04.txt.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. URL http://www.ietf.org/rfc/rfc2616.txt. Updated by RFC 2817.

[18] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999. URL http://www.ietf.org/rfc/rfc2617.txt.

[19] M. Garcia-Martin, M. Belinchon, M. Pallares-Lopez, C. Canales-Valenzuela, and K. Tammi. Diameter Session Initiation Protocol (SIP) Application. RFC 4740 (Proposed Standard), November 2006. URL http://www.ietf.org/rfc/rfc4740.txt.

[20] Jason Garman. Single Sign-on for Your Web Applications with Apache and Kerberos, November 2003. URL http://www.onlamp.com/pub/a/onlamp/2003/09/11/kerberos.html.

[21] Michael Graves. And then there were none - Zero Passwords with Client Certificates, April 2007. URL http://janrain.com/blog/2007/04/20/and-then-there-were-none-zero-passwords-with-client-certificates/.

[22] H. Hakala, L. Mattila, J-P. Koskinen, M. Stura, and J. Loughney. Diameter Credit-Control Application. RFC 4006 (Proposed Standard), August 2005. URL http://www.ietf.org/rfc/rfc4006.txt.

[23] Marit Hansen. User-Controlled Identity Management the Future of Privacy. In *Identity in a Networked World*. FidiS consortium, August 2006.

[24] Marit Hansen, Henry Krasemann, Christian Krause, Martin Rost, and Dr. Riccardo Genghini. Identity Management Systems (IMS): Identification and Comparison Study, 2003.

[25] Dick Hardt, Johnny Bufu, and Josh Hoyt. OpenID Attribute Exchange 1.0 - Draft 07.

[26] S. Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648 (Proposed Standard), October 2006. URL http://www.ietf.org/rfc/rfc4648.txt.

[27] Sampo Kellomäk and Rob Lockhart. Liberty ID-SIS Personal Profile Service Specication. Liberty Alliance ID-SIS 1.0 Specification, November 2006. URL http://www.projectliberty.org/liberty/content/download/1028/7146/file/liberty-idsis-pp-v1.1.pdf.

[28] Daniel Kouril. Kerberos Module for Apache, . URL http://modauthkerb.sourceforge.net/.

[29] Daniel Kouril. Negotiateauth Project: HTTP Negotiate authentication for Mozilla-based browsers, . URL http://negotiateauth.mozdev.org/.

[30] Ben Laurie. OpenID: Phishing Heaven, January 2007. URL http://www.links.org/?p=187.

[31] E. Stewart Lee. Essays about Computer Security, 1999. URL http://www.cl.cam.ac.uk/~mgk25/lee-essays.pdf.

[32] Microsoft Corporation. .NET Framework Developer's Guide: Role-based Security, 2007. URL http://msdn2.microsoft.com/en-us/library/52kd59t0(VS.71).aspx.

[33] Microsoft Corporation. Windows CardSpace, 2006. URL http://cardspace.netfx3.com/.

[34] Microsoft Corporation. Windows Vista Technical Articles - Introducing Windows CardSpace, 2007. URL http://msdn2.microsoft.com/en-us/library/aa480189.aspx.

[35] Joaquin Miller. Yadis Specification Version 1.0, March 2006. URL http://yadis.org/papers/yadis-v1.0.pdf.

[36] National E-Health Transition Authority Ltd. Identity Management - Glossary of Terms (Version 1.0). Public Release, August 2007. URL http://www.nehta.gov.au/index.php?option=com_docman\&task=doc_download%&gid=320\&Itemid=139.

[37] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), July 2005. URL http://www.ietf.org/rfc/rfc4120.txt. Updated by RFCs 4537, 5021.

[38] OASIS. OASIS Security Services (SAML) TC, 1993-2007. URL http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.

[39] Object Mentor. JUnit.org: Resources for Test Driven Development, September 2007. URL http://www.junit.org/.

[40] Open Diameter Project. Open Diameter, 2007. URL http://opendiameter.org/.

[41] openid.net. OpenID. URL http://openid.net/.

[42] Joon S. Park, Ravi Sandhu, and Gail-Joon Ahn. Role-based access control on the web. *ACM Trans. Inf. Syst. Secur.*, 4(1):37–71, 2001. ISSN 1094-9224. doi: http://doi.acm.org/10.1145/383775.383777.

[43] PRIME Project. PRIME - Privacy and Identity Management for Europe, 2007. URL https://www.prime-project.eu/.

[44] projectliberty.org. Liberty Specs Tutorial. URL http://www.projectliberty.org/liberty/content/download/423/2832/file/tutorialv2.pdf.

[45] projectliberty.org. The Liberty Alliance, October 2007. URL http://www.projectliberty.org/.

[46] Nick Ragouzis, John Hughes, Rob Philpott, and Eve Maler. Security Assertion Markup Language 2 (SAML) V2.0 Technical Overview. Working Draft 10, October 2006.

[47] D. Recordon and B. Fitzpatrick. OpenID Authentication 1.1. Finalized OpenID Specification, May 2006. URL http://openid.net/specs/openid-authentication-1_1.html.

[48] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000. URL http://www.ietf.org/rfc/rfc2818.txt.

[49] Joyce K. Reynolds and Robert Braden. Instructions to Request for Comments (RFC) Authors. Internet-Draft, August 2004. URL ftp://ftp.rfc-editor.org/in-notes/rfc-editor/instructions2authors.txt.

[50] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial In User Service (RADIUS). RFC 2865 (Draft Standard), June 2000. URL http://www.ietf.org/rfc/rfc2865.txt. Updated by RFCs 2868, 3575.

[51] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. URL http://www.ietf.org/rfc/rfc3261.txt. Updated by RFCs 3265, 3853, 4320, 4916.

[52] Vishal Sharma. Secure Authentication in Web Based J2EE/JEE Development, October 2007. URL http://entips.blogspot.com/2007/10/secure-authentication-in-web-based.html.

[53] Marco Slot. Beginnerś guide to OpenID phishing. URL http://marcoslot.net/apps/openid/.

[54] specs@openid.net. OpenID Authentication 2.0 - Draft 12, August 2007. URL http://openid.net/specs/openid-authentication-2_0-12.html.

[55] B. Sterman, D. Sadolevsky, D. Schwartz, D. Williams, and W. Beck. RADIUS Extension for Digest Authentication. RFC 4590 (Proposed Standard), July 2006. URL http://www.ietf.org/rfc/rfc4590.txt.

[56] Sun Microsystems, Inc. JavaServer Pages Technology. URL http://java.sun.com/products/jsp/.

[57] The Eclipse Foundation. Higgins Project Home, 2007. URL http://www.eclipse.org/higgins/.

[58] The Internet Engineering Task Force (IETF). Diameter Maintenance and Extensions (dime). URL http://www.ietf.org/html.charters/dime-charter.html.

[59] The MIT Kerberos Consortium. MIT Kerberos Consortium, 2007. URL http://www.kerberos.org/index.html.

[60] The RFC Editor. How to Publish - RFC Editor Publication Process. URL http://www.rfc-editor.org/howtopub.html.

[61] The RFC Editor. RFC Editor Tutorial, July 2007. URL ftp://ftp.rfc-editor.org/in-notes/rfc-editor/tutorial.latest.pdf.

[62] Eushiuan Tran. Verification/Validation/Certification. In Philip Koopman, editor, *Topics in Dependable Embedded Systems*. Carnegie Mellon University, 1999. URL http://www.ece.cmu.edu/~koopman/des_s99/verification/index.html.

[63] Alex Vasin and Erick Svenson. JDiameter: Diameter Base Protocols written in Java, 2007. URL https://jdiameter.dev.java.net/.

[64] Vidoop LLC. Vidoop Secure, September 2007. URL http://www.vidoop.com/products.php?topic=vidsec.

[65] xmldap.org. xmldap.org. URL https://xmldap.org/.

[66] K. Zeilenga. Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map. RFC 4510 (Proposed Standard), June 2006. URL http://www.ietf.org/rfc/rfc4510.txt.

A. Diameter WebAuth AVPs

A.1. Diameter base protocol

Table A.1 shows a complete list of all Diameter Base Protocol (RFC 3588) AVPs that are used by the Diameter WebAuth specification.

A.2. Diameter Network Access Server application

Table A.2 shows a complete list of all Diameter Network Access Server Application (RFC 4005) AVPs that are used by the Diameter WebAuth specification.

A.3. HTTP-Digest authentication

The following section describes the AVPs used for the HTTP-Digest Authentication in Web-Auth-Request and Web-Auth-Response commands.

HTTP-Digest-Challenge AVP

The HTTP-Digest-Challenge AVP is identical to the SIP-Authenticate AVP specified in RFC 4740, Section 9.5.3. [19] and is renamed here for descriptive reasons.

The HTTP-Digest-Challenge AVP has the following ABNF grammar:

```
HTTP-Digest-Challenge ::= < AVP Header: 379 >
                          { Digest-Realm }
                          { Digest-Nonce }
                          [ Digest-Domain ]
                          [ Digest-Opaque ]
                          [ Digest-Stale ]
                          [ Digest-Algorithm ]
                          [ Digest-QoP ]
                          [ Digest-HA1]
                        * [ Digest-Auth-Param ]
                        * [ AVP ]
```

Table A.1.: Reused Diameter base protocol AVPs

Attribute Name	AVP Code	Value Type	Reference
Origin-Host	264	DiameterIdentity	RFC 3588, Section 6.3. [8]
Origin-Realm	296	DiameterIdentity	RFC 3588, Section 6.4. [8]
Destination-Host	293	DiameterIdentity	RFC 3588, Section 6.5. [8]
Destination-Realm	283	DiameterIdentity	RFC 3588, Section 6.6. [8]
Auth-Application-Id	258	Unsigned32	RFC 3588, Section 6.8. [8]
Acct-Application-Id	259	Unsigned32	RFC 3588, Section 6.9. [8]
Result-Code	268	Unsigned32	RFC 3588, Section 7.1. [8]
Auth-Request-Type	274	Enumerated	RFC 3588, Section 8.7. [8]
Session-Id	263	UTF8String	RFC 3588, Section 8.8. [8]
Authorization-Lifetime	291	Unsigned32	RFC 3588, Section 8.9. [8]
Auth-Grace-Period	276	Unsigned32	RFC 3588, Section 8.10. [8]
Auth-Session-State	277	Enumerated	RFC 3588, Section 8.11. [8]
User-Name	1	UTF8String	RFC 3588, Section 8.14. [8]
Event-Timestamp	55	Time	RFC 3588, Section 8.21. [8]

Table A.2.: Reused Diameter Network Access Server application AVPs

Attribute Name	AVP Code	Value Type	Reference
User-Password	2	OctetString	RFC 4005, Section 5.1. [10]

HTTP-Digest-Response AVP

The HTTP-Digest-Response AVP is identical to the SIP-Authorization AVP specified in RFC 4740, Section 9.5.4. [19] and is renamed here for descriptive reasons.

The HTTP-Digest-Response AVP has the following ABNF grammar:

```
HTTP-Digest-Response ::= < AVP Header: 380 >
                         { Digest-Username }
                         { Digest-Realm }
                         { Digest-Nonce }
                         { Digest-URI }
                         { Digest-Response }
                         [ Digest-Algorithm ]
                         [ Digest-CNonce ]
                         [ Digest-Opaque ]
                         [ Digest-QoP ]
                         [ Digest-Nonce-Count ]
                         [ Digest-Method]
                         [ Digest-Entity-Body-Hash ]
                       * [ Digest-Auth-Param ]
                       * [ AVP ]
```

HTTP-Authentication-Info AVP

The HTTP-Digest-Info AVP is identical to the SIP-Authentication-Info AVP specified in RFC 4740, Section 9.5.5. [19] and is renamed here for descriptive reasons.

The HTTP-Digest-Info AVP has the following ABNF grammar:

```
HTTP-Digest-Info ::= < AVP Header: 381 >
                     [ Digest-Nextnonce ]
                     [ Digest-QoP ]
                     [ Digest-Response-Auth ]
                     [ Digest-CNonce ]
                     [ Digest-Nonce-Count ]
                   * [ AVP ]
```

HTTP-Digest AVPs

Table A.3 lists all AVPS that are RADIUS attributes defined in RFC 4590 [55] and that are imported by RFC 4740 (cp. Section 9.5.6.) [19] to be used for the HTTP-Digest authentication.

Table A.3.: Reused HTTP-Digest authentication attributes

Attribute Name	RADIUS Type
Digest-Response	103
Digest-Realm	104
Digest-Nonce	105
Digest-Response-Auth	106
Digest-Nextnonce	107
Digest-Method	108
Digest-URI	109
Digest-QoP	110
Digest-Algorithm	111
Digest-Entity-Body-Hash	112
Digest-CNonce	113
Digest-Nonce-Count	114
Digest-Username	115
Digest-Opaque	116
Digest-Auth-Param	117
Digest-AKA-Auts	118
Digest-Domain	119
Digest-Stale	120
Digest-HA1	121

Table A.4.: Reused credit-control application AVPs

Attribute Name	AVP Code	Value Type	Reference
CC-Request-Number	415	Unsigned32	RFC 4006, Section 8.2. [22]
CC-Request-Type	416	Enumerated	RFC 4006, Section 8.3. [22]
Check-Balance-Result	422	Enumerated	RFC 4006, Section 8.6. [22]
Unit-Value	445	Grouped	RFC 4006, Section 8.8. [22]
Exponent	429	Integer32	RFC 4006, Section 8.9. [22]
Value-Digits	447	Integer64	RFC 4006, Section 8.10. [22]
Currency-Code	425	Unsigned32	RFC 4006, Section 8.11. [22]
Granted-Service-Unit	431	Grouped	RFC 4006, Section 8.17. [22]
Requested-Service-Unit	437	Grouped	RFC 4006, Section 8.18. [22]
CC-Time	420	Unsigned32	RFC 4006, Section 8.21. [22]
CC-Money	413	Grouped	RFC 4006, Section 8.22. [22]
CC-Service-Specific-Units	417	Unsigned64	RFC 4006, Section 8.26. [22]
Service-Identifier	439	Unsigned32	RFC 4006, Section 8.28. [22]
Requested-Action	436	Enumerated	RFC 4006, Section 8.41. [22]
Service-Context-Id	461	UTF8String	RFC 4006, Section 8.42. [22]
Subscription-Id	443	Grouped	RFC 4006, Section 8.46. [22]
Subscription-Id-Type	450	Enumerated	RFC 4006, Section 8.47. [22]
Subscription-Id-Data	444	UTF8String.	RFC 4006, Section 8.48. [22]

A.4. Diameter credit-control application

Table A.4 shows a complete list of all Diameter credit-control application (RFC 4006) AVPs that are used by the Diameter WebAuth specification.

The following AVPs need further explanations:

CC-Request-Number Since this application only supports credit control one time events, the CC-Request-Number can simply be set to 0 for every request.

CC-Request-Type The CC-Request-Type is EVENT_REQUEST for all one time credit control events.

Subscription-Id For a number of requests and responses in RFC 4006 [22] a Subscription-Id AVP is required or recommended. For compatibility reasons it therefore SHOULD be included in such requests/responses even if the web application does

not use subscription ids for their users. In such case, the Subscription- Id-Type AVP SHOULD be set to END_USER_PRIVATE and the Subscription-Id-Data AVP SHOULD be set to the same value as the User-Name AVP.