Security in OpenID

An overview of OpenID from a security perspective

MAX CHARAS
Security in OpenID

An overview of OpenID from a security perspective

M A X C H A R A S

Master’s Thesis in Computer Science (30 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2009
Supervisor at CSC was Douglas Wikström
Examiner was Johan Håstad

TRITA-CSC-E 2009:138
ISRN-KTH/CSC/E--09/138--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.csc.kth.se
Abstract

The goal of this thesis is to analyze some of the most pressing security vulnerabilities of the web based single sign-on system OpenID. The thesis assumes little or no previous knowledge of Single Sign-On (SSO) systems and will therefore cover the terminology needed to understand the inner workings of OpenID. After a technical walkthrough of the OpenID system the security vulnerabilities are presented and solutions are given.

The security vulnerabilities are primarily of importance to anyone implementing or maintaining any part of an OpenID system, but a few of the vulnerabilities are also important to the average end-user.

Referat

En överblick av OpenID i ett datasäkerhetsperspektiv

Målet med denna rapport är att belysa några av de allvarligaste säkerhetsbristerna i det internetbaserade single sign-on system OpenID. Eftersom att läsaren av rapporten inte förväntas besitta någon tidigare kunskap av single sign-on system täcker rapporten även den basala terminologi som behövs för att kunna diskutera ämnet.

Efter en teknisk genomgång av systemet belyses säkerhetsbristerna varpå även lösningar presenteras. Säkerhetsbristernas natur gör att de primärt är av intresse för systemadministratörer och utvecklare som kommer i kontakt med OpenID-system. Vissa av säkerhetsbristerna är dock också av vikt för vanliga slutanvändare.
## Contents

1 Introduction ................................. 1
  1.1 Outline of the Thesis ...................... 1
  1.2 Intended Audience ......................... 2

I Background ................................ 3

2 Terminology ................................ 5
  2.1 Identity .................................. 5
    2.1.1 Example: Passport ...................... 5
  2.2 Digital Identity ........................... 6
    2.2.1 Example: Username and Password ....... 6
  2.3 Reliable End User Identification ......... 6
  2.4 Single Sign-On ............................. 6
    2.4.1 SSO and Cookies ....................... 7
  2.5 Federated Identity ......................... 7

3 Technology Overview ......................... 9
  3.1 Kerberos .................................. 9
  3.2 Microsoft Live ID Passport and Cardspace 10
  3.3 Security Assertion Markup Language ....... 11
  3.4 OpenID .................................... 11
  3.5 Electronic ID ............................... 12
  3.6 Technology Overview Summary ............. 13
    3.6.1 Kerberos OpenID ....................... 13
    3.6.2 Microsoft Live ID and Microsoft Info Card OpenID 13
    3.6.3 SAML OpenID ............................ 13
    3.6.4 Electronic ID OpenID .................. 13

II OpenID .................................. 15

4 OpenID Protocol Overview ................... 17
  4.1 OpenID Subjects/Actors ..................... 17
List of Figures

4.1 Example OpenID provider. ................................................. 18
4.2 Example relying party. ................................................... 19
4.3 An example of a Joomla Content Management System OpenID sign in box. ......................................................... 19

5.1 Main sequence diagram provides a schematic chronological overview. ................................................. 22
5.2 Examples of normalization. ................................................ 23
5.3 Schematic overview of normalization. .................................... 24

6.1 Prototype schematic overview. ............................................. 34

7.1 Alices normal provider. .................................................... 40
7.2 A phishing attempt against Alice. Can you, the reader, identify the difference from the previous picture? ..................................................... 40
7.3 This is an example of a yahoo Sign in Seal. In this case the KTH-logo in the upper right corner is the seal for this user. A user should get suspicious if their personal seal is not shown when logging in. In this case the Seal is stored in an Flash Stored Object. ............................................. 41
Chapter 1

Introduction

The subject of this thesis is a Internet based single sign-on system that received much media attention during the last few years, namely OpenID. The objective of this thesis is to:

- Explain OpenID on a technical level
- Compare OpenID to existing similar technologies
- Explore the most pressing security issues in OpenID
- Suggest how to solve these problems
- And finally discuss other indirect benefits of OpenID

1.1 Outline of the Thesis

This thesis is divided into three primary parts. The first part, background, covers basic terminology required for the rest of the report. This is followed by a technology overview comparing different technologies more or less similar to OpenID. This part is intended for any reader who wants to get familiar with some of the terminology used when discussing single sign-on technologies.

OpenID, part two, intends to give a deeper understanding of the OpenID protocol. This part is divided into the chapters OpenID Protocol Overview and the more technical chapter OpenID protocol in-depth. The in depth step by step walkthrough in the second of the two chapters is intended for a reader who wants to understand OpenID on a more technical level. This understanding may be necessary to grasp the individual security threats presented in the last part.

Results, the final of the three parts presents a number of security threats to OpenID. All of the threats are discussed and solutions are presented. This final part is ended with a more open discussion surrounding trust in single sign-on systems and what other benefits OpenID might give the Internet community.
1.2 Intended Audience

This thesis is first of all intended for anyone who wants to understand OpenID and its current most pressing security threats.

The readers of the first part and the first chapter of part two should at least be familiar with modern day Internet usage.

The readers of the technical overview of and the discussion of the current security threats may require some knowledge of Internet architecture and XML/HTML.
Part I

Background
Chapter 2

Terminology

As identity is a very broad subject with many different definitions, I will start this report by defining and explaining some important concepts that are used throughout this thesis. Concepts that are explained include: identity, reliable end-user identification and single sign-on.

2.1 Identity

So let us start by defining the most important term in this report, identity. We will define identity as:

“A set of claims made by one subject about itself or another subject.”

[3]

• Ok, so what is a subject? More or less anything, a person, a government agency, a corporation and it may even be a computer.

• Well what is a claim then? An assertion of the truth of something, typically one which is disputed or in doubt.

To make this definition more tangible I will here illustrate with an example.

2.1.1 Example: Passport

A Swedish passport can be issued by the Swedish National Police Board (Rikspolisstyrelsen, RPS) to a Swedish citizen. This document strengthens the claim of citizenship that the individual has about him or herself.

The identity in this case is the claim of citizenship that the individual has of him or herself.

One could also argue that there is an identity formed when the Swedish National Police Board also claims that the individual is a Swedish citizen.
2.2 Digital Identity

The definition of a digital identity is very close to the previous definition. In fact let us just add the word digital before the word subject to signify that the subject is in fact digital. The definition:

'A set of claims made by one digital subject about itself or another digital subject.'

2.2.1 Example: Username and Password

The most commonly used digital identity today is probably the username/password-combination used to access most service providing Internet sites today. You, the reader, probably have several of these identities for e-mail, Internet communities, forums and so on. With the username you claim to be the digital subject that has access to the service you want to use and then you use the password to authenticate yourself.

A physical identity document like a passport is most commonly verified through visual clues on the document itself while a digital identity is verified through mathematical algorithms. Most of us today tend to distinguish between the real world and the virtual world, which brings us to the next topic reliable end-user identification.

2.3 Reliable End User Identification

An interesting problem arises when one subject belongs to the virtual world and the other one is part of the real world. For example, say that we have a citizen, call him Bob. Bob wants to digitally identify himself to some government agency. Furthermore let us assume that Bob has some kind of digital identity that he uses towards the agency. For a government agency it is usually imperative that the digital identity really represents the citizen Bob. How can the government agency be sure that this digital identity really is Bob? Or to rephrase the question:

'How can the government agency be sure that digital Bob is the real Bob?'

This is reliable end-user identification in a nutshell, how do we connect the virtual world and the real world.

2.4 Single Sign-On

In a previous example we concluded that username and password is in fact a simple form of a digital identity, probably the most frequently used identity on the Internet today. Requiring a user to remember a unique password and username for each site often results in that the user reuses his or her password on many different locations,
2.5. FEDERATED IDENTITY

a problem that will be discussed later in the report. One solution to this problem is "Single Sign On" or SSO for short.

The SSO concept is very simple; only require the user to log in once. For example if I need to access my mail, a discussion forum and a social community a fully SSO-enabled system would only require me to log in once with one identity instead of logging into each service individually.

As you of course realize having only one password also has its downside, if somebody gets a hold of this master password they can access all your services.

2.4.1 SSO and Cookies

The most popular way to store login credentials though a web-browser is the so called cookie. The cookie is basically just a text file that can be stored on the user’s computer to be read later. But there is one very important restrictions set on cookies, they can only be read by the domain that created them. For example, if the domain www.example.org creates a cookie with my username and password on my computer, the domain www.example.com cannot read it.

A speculative guess why single sign-on has been so hard to design on the web could be because of this restriction set on cookies. Most web designers take for granted that login credentials should be stored in cookies, but if www.example.org needs to share these credentials with www.example.com a cookie is clearly an incorrect approach.

2.5 Federated Identity

The step beyond SSO is a concept called "Federated identities". The idea here is not only to be able to share authentication of users but also user information.

Say for example that you are flying to Berlin with your normal airline. You book a ticket to Berlin and intend to stay there a couple of days. The airline now suggests that you might want to book accommodations at an hotel.

If they both now support federated identities the airline would then be able to transmit data about you the customer to the hotel. The information could for example be when you arrive and when you intend to depart thereby simplifying your hotel booking.

SSO enables parties to have collaborative authentication systems. In this paradigm the identity is still stored at each separate part. While in federated identity system the parties can exchange the identity itself not only the authentication.

One could of course raise an ethical discussion sharing more or less personal information between business partners but this thesis will not enter into this discussion.
Chapter 3

Technology Overview

This chapter intends to give an overview of some well known technologies that solve single sign-on, federated identities and reliable end-user identification in some way or form. The goal of this is to be able to compare OpenID to similar existing technologies.

3.1 Kerberos

Kerberos is a computer network authentication protocol developed by MIT during the 1980’s and is today one of the most widely spread authentication solutions. Since Kerberos was developed as early as the 80’s, before the days of the Internet, there was no architectural vision to make the protocol browser/web oriented. Therefore Kerberos is primarily used as a single sign-on solutions for internal networks.

The design is centered on a shared secret, the user password. This secret is used to negotiate a session encryption which then can be used to give the user timestamped tickets called Kerberos tickets. These time-limited tickets are then used to gain access to the internal services. [9]

Aha you may say now, let us just put this ticket in a cookie, that way Kerberos becomes a SSO-solution! The problem arises when you have a system with multiple domains in your system. Then the cookie that one domain stores cannot be read by another domain in the system, as discussed in Section 2.4.1 SSO and Cookies.

Intended Use. Single sign-on solution for internal networks.

Pros and Cons

- **Pro**: Widely used.
- **Pro**: Well scrutinized.
- **Con**: Requires synchronization between the clocks of the involved hosts. [9]
3.2 Microsoft Live ID Passport and Cardspace

In 1999 Microsoft launched a new technology named .Net Passport (now called Microsoft Live ID). The goal of the technology was to become a widely spread single sign-on system both in the enterprise market and the web based consumer market. [7] [8] Unfortunately for Microsoft the .Net passport technology came under media limelight for having serious security flaws. [5]

Furthermore the author of the Laws of Identity[3], had a vision that was not compatible with the .Net Passport vision. One of the laws Pluralism of Operators and Technologies states:

'A universal identity system must channel and enable the inter-working of multiple identity technologies run by multiple identity providers. It would be nice if there were one way to express identity. But the numerous contexts in which identity is required won't allow it. One reason there will never be a single, centralized monolithic system is because the characteristics that would make any system ideal in one context will disqualify it in another.'

Microsoft has now given birth to a more agile identity metasystem more or less based on Laws of Identity.

The primary goal of an identity metasystem is to encapsulate several underlying identity technologies such SAML, OpenID and .Net Passport. This metasystem together with an intuitive user friendly graphical interface rids the end-user from having to learn several different technologies. Combining these two ideas is the new Microsoft product Cardspace.

Microsoft Live ID, Intended use. Microsoft Live ID (.Net Passport) is a single sign-on system for applications and web applications.

Pros and Cons

- **Pro**: Widely used in Microsoft systems.
- **Pro**: Web based infrastructure.
- **Con**: Bad reputation for having security flaws. [5]

Microsoft Cardspace, Intended use. Microsoft Cardspace is a metasystem intended to simplify digital identity handling for the end-user.

Pros and Cons

- **Pro**: Identity meta system encapsulating several underlying technologies.
- **Pro**: User friendly environment.
3.3 Security Assertion Markup Language

Security Assertion Markup Language (SAML) is an XML standard for exchanging authentication and authorization data between services. This XML standard solves two problems, single sign-on and Federated identities. SAML is a product of the OASIS Security Services Technical Committee but the current version of SAML has also had a large contribution from Liberty Alliance.

This XML standard is intended mostly for business partners that want a standard for exchanging security information. As it is intended to be used by large businesses the OASIS seem to have chosen a more feature rich standard but at a cost, its complexity.

**Intended use.** Exchanging authentication and authorization data between business partners thereby solving single sign-on and notably Federated identities problems.

**Pros and Cons**
- **Pro:** Widely used.
- **Pro:** Web based infrastructure, through XML, SOAP and HTTP.
- **Pro:** Supports Federated identities.
- **Con:** Complex.

3.4 OpenID

OpenID is a relatively new protocol whose sole purpose is to solve the single sign-on problem for web-services. It started development in early 2005 [4] and has since gained massive attention both from the Internet developer community and large corporations. Today Google, IBM, Microsoft, Verisign, and Yahoo! are OpenID Foundation corporate board members. [6]

Unlike SAML, OpenID focuses on solving one problem, namely single sign-on. This made it possible to make the OpenID protocol shorter and easier to understand. One could speculate whether OpenID’s popularity may be a reaction to SAML’s complexity.

**Intended use.** Web based single sign-on for web-services.

**Pros and Cons**
- **Pro:** Easy to learn.
- **Pro:** Web based infrastructure, through XML, and HTTP.
- **Con:** New.
CHAPTER 3. TECHNOLOGY OVERVIEW

3.5 Electronic ID

Electronic ID or, E-Legitimation as it is known in Sweden, is an umbrella term for x.509 certificate based technologies that solve the reliable end-user problem. Of course the reliable end-user problem can’t be solved purely though technology, after all as discussed before we need to connect the virtual world and the real world.

A good way to connect “physical realm” and “digital realm” with the intention to solve reliable end-user identification is to use an existing business that has both a well established real world identity infrastructure and some kind of website. A good example of this is a bank. Banks generally require solid identification of customers and most banks today also have some kind of web-interface to access their services.

With this in mind it’s not surprising that the leading Electronic ID actor in Sweden called Bank ID is a joint venture between 9 of Sweden’s largest banks namely [2]:

- Danske Bank i Sverige
- Handelsbanken
- Ikanobanken
- Länsförsäkringar Bank
- SEB
- Skandiabanken
- Sparbanken Finn
- Sparbanken Gripen
- Swedbank

Electronic ID is now widely used by the Swedish public to access and use services from e.g. Rikskatteverket (The National Tax Board), Riksförsäkringsverket (The National Social Insurance Board) and Centrala studiestödsnämnden (Swedish agency in charge of financial aid for studies).

The downside of using a solid identity infrastructure is its cost. Bank ID charges a fix price SEK every time an identity is checked against their system.

**Intended use.** x.509 certificate based identity with a java based client side web application for authentication.

**Pros and Cons**

- **Pro:** Based on the proven x.509 certificate standard.
- **Pro:** Widely used.
• **Pro:** Supports Reliable end-user identification.

• **Con:** Expensive.

### 3.6 Technology Overview Summary

So by now you should have a view of some different technologies, so what makes OpenID different? let us compare them one by one

#### 3.6.1 Kerberos OpenID

Kerberos early design makes it unsuitable for the Internet. Since Kerberos is designed around the user’s password it also lacks the authentication flexibility of OpenID. Forcing all hosts to have synchronized clocks is also an architectural flaw that does not fit for a modern web based single sign-on system.

#### 3.6.2 Microsoft Live ID and Microsoft Info Card OpenID

Microsoft Live ID is a similar to OpenID but most end-users will not accept a system where the only identity provider is Microsoft.

#### 3.6.3 SAML OpenID

SAML’s support for both SSO and Federated Identities makes it a very complex standard intended for big businesses that can invest large sums of money. But for a single sign-on system to spread and get widely accepted it has to be inexpensive to implement and easy to learn.

#### 3.6.4 Electronic ID OpenID

Even though OpenID is not designed to be a system that solves reliable end-user identification it is still compared to Bank ID in this report, this to highlight the differences between the two technologies. The enormous infrastructural costs of securely identifying a physical citizen in a country forces bank ID to have a rigid payment system where every transaction is laden with a cost.

Though as you progress in this report you as a reader can think about the possibilities of creating an OpenID provider that guarantees reliable end-user identification.
Part II

OpenID
Chapter 4

OpenID Protocol Overview

As described before, OpenID is designed as a single sign-on solution for web-services. This chapter will give a more comprehensive overview the OpenID 2.0 authentication protocol. OpenID will first be presented from a users perspective through a two case examples. After this, a more detailed analysis of the protocol itself will be presented.

All examples containing a domain name in this report will use the example domains specified by the RFC 2606. [11]

4.1 OpenID Subjects/Actors

First of all let us look at the three actors of an OpenID system.

The user. The end-user or user, let us call her Alice, has one goal, to login and use one more services on the Internet. In the following example Alice wants to post a comment on her friends Bob’s blog, at www.example.com/bob.

The OpenID provider. The second actor is called the OpenID provider and exists to provide a stable identity for its users. This is accomplished by giving all users a unique URL identifier, called the OpenID. Let’s say that Alice uses www.example.org as her OpenID provider.

The provider could then have given Alice the unique identifier alice.example.org in OpenID called OpenID identifier or just identifier. In the following use case we will assume that Alice already has an account at www.example.org and that her identifier is alice.example.org.

The relying party. The third and final actor is the relying party in this case www.example.com/bob. It’s called a relying party because it relies on the OpenID provider to authenticate its users.
CHAPTER 4. OPENID PROTOCOL OVERVIEW

4.2 How does it work?

OpenID architecture is built around the unique URL identifier, the OpenID, that every end-user possesses. The process of authentication is in theory a fairly straightforward process which can be summarized in a few key steps:

4.2.1 Case 1

1. Alice is visiting www.example.com/bob and intends to post a comment, to do this she has to login to Bob’s blog.

2. Alice notices the icon in one of the sign in fields. This tells her that she can use her OpenID to sign in. So she enters alice.example.org and presses Sign in, she is now claiming to own the OpenID: alice.example.org (see Figure 4.2 and 4.3)

3. Alice’s web browser is then redirected to her OpenID provider example.org.

4. The OpenID provider example.org prompts Alice to identify herself. How the provider chooses to identify its users is not in any way specified by OpenID, though most providers use the usual username and password combination. (It is also common that the provider also prompts the user if they trust the site they are about to sign in to. In this case the relying party example.org would ask Alice whether she trusts www.example.com/bob.)

5. Finally example.org redirects Alice back to Bob’s blog and at the same time telling the relying party (Bob’s blog) www.example.com/bob that Alice owns alice.example.org as she claimed.

6. Alice is now logged in and Bob’s blog knows Alice as alice.example.org.

Open ID provider

Hello alice.example.org please identify yourself

Username: [blank]
Password: [blank]

Submit

Figure 4.1. Example OpenID provider.
4.2. HOW DOES IT WORK?

4.2.2 Case 2

1. After reading Bob’s blog Alice decides that she wants to post on her own blog. She enters www.example.net/aliceblog which is the url to her blog.

2. Well there she again notices the icon in one of the sign in fields. This tells her that she can use her OpenID identifier to sign in here so she enters alice.example.org and presses Sign in.

3. Alice’s web browser is then redirected to her OpenID provider example.org.

4. The OpenID provider www.example.org now notices that Alice is already logged in. Instead of prompting her to login once again her web browser is redirected directly back to her blog. She hardly ever notices leaving the blog.

Case 2 illustrates how OpenID is a single sign-on technology. Alice only has to enter her password once even though signing in to two different services.

Note also that the services are at two different domains, example.net and example.com. This illustrates that OpenID has solved the multi-domain cookie problem discussed in the Terminology chapter.
Chapter 5

OpenID Protocol In-depth

5.1 Schematic Overview

The previous use case is a simplification and to understand the protocol in-depth we will have to look at the information flow between the three actors step by step. The main sequence diagram, Figure 5.1, provides a schematic chronological overview.

The figure has been divided into 8 categories seen along the left-hand side. These categories or events will each be explained in a separate following chapter beginning with Acquiring Identifier.

5.2 Acquiring Identifier

The OpenID process starts with the relying party asking for the end-user’s unique identifier. This is usually done through a HTML form.

The OpenID Authentication 2.0 specification states that the name attribute of the text box should be openid_identifier. This is so that User-Agents (web browsers) automatically can determine that this is an OpenID form.

The identifier can be one of two things:

1. Uniform Resource Locator (URL) (either http or https)
2. eXtensible Resource Identifier (XRI)

Here are some examples of how OpenID identifiers look. These are all URLs:

- **Name Identifier**
  - AOL openid.aol.com/screenname
  - Blogger blogname.blogspot.com
  - Flickr www.flickr.com/photos/username
  - LiveDoor profile.livedoor.com/username
Figure 5.1. Main sequence diagram provides a schematic chronological overview.

- LiveJournal username.livejournal.com
- SmugMug username.smugmug.com
- Technorati technorati.com/people/technorati/username
5.3. NORMALIZATION

- Vox member.vox.com
- Yahoo http://openid.yahoo.com/username
- WordPress username.wordpress.com

The identifier is the first bit of information sent from the end-user to the relying party.

5.3 Normalization

After acquiring the identifier from the end-user the relying party must normalize it. The normalization takes the user supplied identifier and returns either a XRI identifier or a URL identifier. The main goal of the normalization is to remove any irregularities that the user might have sent with the input. (see Figure 5.3)

5.3.1 Normalization Examples

Figure 5.3 presents a flowchart describing the steps of normalization in detail but to clarify here are some examples to clarify what the normalization does:

<table>
<thead>
<tr>
<th>User’s Input</th>
<th>Identifier</th>
<th>Type</th>
<th>Discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice.example.org</td>
<td><a href="http://alice.example.org/">http://alice.example.org/</a></td>
<td>URL</td>
<td>A URL with a missing scheme is normalized to a http</td>
</tr>
<tr>
<td><a href="https://alice.example.org/">https://alice.example.org/</a></td>
<td><a href="https://alice.example.org/">https://alice.example.org/</a></td>
<td>URL</td>
<td>https URLs remain https URLs</td>
</tr>
<tr>
<td>=example</td>
<td>=example</td>
<td>XRI</td>
<td>Normalized XRIIs start with a global context symbol</td>
</tr>
<tr>
<td>xri://=example</td>
<td>=example</td>
<td>XRI</td>
<td>Normalized XRIIs start with a global context symbol</td>
</tr>
</tbody>
</table>

Figure 5.2. Examples of normalization.

Removal of xri protocol prefixes. The first step of normalization is to remove any xri protocol prefixes such as: xri:// xri://$ip xri://$dns#

Starts with XRI Global Content Symbol? If what’s left of the identifier starts with a XRI Global Content Symbol (‘=’, ‘@’, ‘+’, ‘$’, ‘!’) or ‘(’, as defined in Section 2.2.1 of XRI Syntax 2.0, then the input should be treated as an XRI identifier and the normalization is complete.
Figure 5.3. Schematic overview of normalization.
5.4. DISCOVERY

Has http/https prefix? If the identifier does not have an XRI Global Content Symbol but has an http:// or an https:// prefix, then keep it and proceed to the next step. If not then append an http:// prefix.

Follow possible http redirects. The relying party has to follow any HTTP redirects it might encounter until it comes to a unresponsive dead end or OpenID server.

5.4 Discovery

After normalizing the user supplied identifier the relying party has one thing, an address to contact the identity provider. The address might for example be http://alice.example.org/. The OpenID protocol does not require the OpenID providers server to run on this exact address. This would be problematic for the ID provider since it might have several thousand subscribers all having their own unique address or identifier.

Instead in OpenID the relying party must ask the OpenID provider on what address the server is located. For example Alice’s normalized identifier is http://alice.example.org/ but the server might be running at http://www.example.org/openidserver/. In addition to this the relying party has to know what version of OpenID protocol the server supports. So the relying party now needs two pieces of information:

1. The OpenID server endpoint address.
2. And what version of OpenID authentication this server supports.

In the chapter Normalization we concluded that the resultant normalized identifier could be either XRI or a URL. Depending on what identifier the relying party obtained different protocols will be used to get the necessary information:

- If the normalized identifier is a XRI identifier a protocol called XRI Resolution Protocol will be used to obtain a so called eXtensible Resource DescriptorS (XRDS). The XRI and the XRI resolution protocol are out of the scope of this document but there will be an example of how a XRDS document might look. If you, the reader, have no previous knowledge about XRI you can see it as the next generation URL.

- If the normalized identifier is a URL the relying party will first try to obtain a XRDS through the Yadis protocol which also will be out of the scope of this document.

- And last but not least if the Yadis protocol did not yield a XRDS the relying party will try to acquire the endpoint address and supported OpenID version though HTML discovery. Further down an example HTML document can be found.
5.4.1 Example eXtensible Resource DescriptionS

Below is an example of a XRDS document. In this case it would tell a relying party that there are two available OpenID provider servers one located at http://www.example.org/openid/server and the other at http://www.example.org/openid/backup.

In addition to this it tells the relying party what version of the OpenID Authentication protocol is supported for each server. The first of the two servers has a higher priority (lower priority index) and will therefore be used before attempting the second backup server.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS xmlns:xrds="xri://\$xrds" xmlns="xri://$xrd*(v*2.0)"
xrds:openid="http://openid.net/xmlns/1.0">
  <XRD>
    <Service priority="1">
      <Type>http://specs.openid.net/auth/2.0/server</Type>
      <URI>http://www.example.org/openid/server</URI>
    </Service>
    <Service priority="2">
      <Type>http://openid.net/signon/1.0</Type>
      <URI>http://www.example.org/openid/backup</URI>
    </Service>
  </XRD>
</xrds:XRDS>
```

5.4.2 Example HTML discovery

```html
<link rel="openid2.provider" href="http://www.example.org/openid/server"/>
```

As described before, if the Yadis protocol fails to retrieve a valid XRDS document the last resort is HTML based discovery. This is a tag located in the `<head>` section of the HTML document and would tell a relying party that there is a OpenID server running at http://www.example.org/openid/server.

In addition openid2.provider tells the relying party that the server accepts OpenID Authentication version 2.0.
5.4. DISCOVERY

5.4.3 Provider Delegation

Say that Alice owns her own domain www.example.net and wants to use this as her unique OpenID identifier. Let's go on and assume that she still wants her OpenID provider www.example.org to take care of her authentication. How can this be accomplished with OpenID?

OpenID Authentication supports so called provider delegation. What Alice then does is basically to say to her relying party that she owns www.example.net but will let www.example.org authenticate her. This is accomplished by putting a small tag of HTML at the address Alice intends to use as her identity. At her relying party Alice would now enter her own url www.example.net as her OpenID but when the relying party performs discovery on this address it will point to www.example.org.

Just as before the relying party will need the OpenID server endpoint address which in this example would be http://www.example.org/openid/server, notice that the server is located at the provider. And again the relying party of course needs to know what version of OpenID authentication this server supports. But when using delegation the relying party needs additional information.

The first new piece of information will be a so called Claimed Identifier which is basically just the same as the previously normalized identifier, in this example it would be www.example.net. This is what Alice enters at the provider, in this case instead of the previously used http://alice.example.org.

The second piece of information is OpenID Provider-Local Identifier. This is the name by which the OpenID provider knows the user, in this example it would be http://alice.example.org.

So when using delegation the relying party now needs 4 pieces of information:

1. The OpenID server endpoint address.
2. What version of OpenID authentication this server supports.
3. Claimed Identifier, the identifier that the user claims to own.
4. OpenID Provider-Local Identifier, how the OpenID provider knows the user.

The relying party is only provided with the claimed identifier, the rest is retrieved through discovery at the claimed identifier address. Examples are provided below.

5.4.4 Example HTML discovery with Delegation

Just as a normal HTML discovery but with the added information about OpenID Provider-Local Identifier http://alice.example.org. This piece of HTML is located at www.example.net and its used to tell the relying party that this identity is in fact authenticated by www.example.org.

```html
<link rel="openid2.provider" href="http://www.example.org/openid/server"/>
<link rel="openid2.local_id" href="http://alice.example.org"/>
```
XRDS also supports delegation just like in the above HTML example. The delegation information is contained in the <openid:Delegate> tag. Example can be found below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS xmlns:xrds="xri://\$xrds" xmlns="xri://$xrd*($v*2.0)"
xmlns:openid="http://openid.net/xmlns/1.0">

<XRD>

<Service priority="1">
  <Type> http://specs.openid.net/auth/2.0/server </Type>
  <URI>http://www.example.org/openid/server</URI>
  <openid:Delegate>
    http://alice.example.org
  </openid:Delegate>
</Service>

<Service priority="2">
  <Type>http://openid.net/signon/1.0</Type>
  <URI>http://www.example.org/openid/backup</URI>
</Service>

</XRD>
</xrds:XRDS>
```

5.5 Request Authentication

The relying party and the OpenID provider now have all the necessary information needed to begin the communication. The relying party now requests authentication from the OpenID provider. This is done by redirecting the end-users browser to the ID provider. The necessary information is transmitted through HTTP when the end-users browser redirects. All of the request parameters are listed in Request authentication table in the appendix.

5.5.1 Realm

When the relying party requests authentication it should also specify a so called realm. The realm indicates to the end-user to what domain or url he or she is being authenticated to. Example realms:

- www.example.com - indicating that the authentication is intended to be used at www.example.com
5.6 Authentication

How the OpenID provider authenticates the end-user is out of the OpenID specifications scope. This is a matter of controversy and will be discussed in subsequent chapters.

5.7 Authentication Response

After the authentication request the OpenID provider redirects the end-user back to the relying party. If the openid-return_to parameter is specified the end-user will be directed to this URL. The authentication response parameters are transmitted through HTTP just as in the case of the request authentication.

Note here that the OpenID provider sends the user back to the relying party through a redirect in the end-users web browser. A malicious end-user could easily tamper with this request.

To make this redirect tamper-resistant it can be signed by the OpenID provider. But if this signature is to be verified by the relying party, the two actors (the relying party and the provider) have to share a common secret. This secret is exchanged during the association phase and can be exchanged through the Diffie-Hellman key exchange. This shared secret enables the relying party and the open provider to relay signed messages through the end-user. If the end-user tampers with the message the signature will no longer be valid.

The association step is considered optional but is recommended since it seriously reduces the end-users ability to tamper with messages relayed through him or her. The signature is generated with HMAC SHA1 or HMAC SHA256.

5.8 Authentication Verification

After the authentication response is received by the relying party it should be verified with the signature provided.
Chapter 6

Prototype

To be able to better understand OpenID I decided to create a test environment. The purpose of this environment was twofold. Firstly I wanted to get a feeling of how difficult it would be to become any of the three previously named actors, end-user, relying party and finally OpenID provider. And secondly I needed a platform where I could test the security flaws and problems of OpenID.

6.1 Design

The first step in creating the testing environment was system architecture. I set up a number of goals that the prototype environment had to fulfill, these were:

1. Contain a working relying party
2. Contain a working OpenID Provider
3. Successfully authenticate the relying party and the OpenID provider
4. Successfully use my OpenID provider to create an id that I could register at a foreign relying party

And to be able to accomplish these goals I needed an OpenID library that fulfilled the following criteria:

- The dependant library had to be open source.
  
  So that I could access code and review it for better understanding.
- The library had to support both OpenID providers and relying parties.
The second step in creating the testing environment was to find a suitable open source library. Java OpenID or JOID for short, a library that "lets you create both OpenID 2.0 relying parties and identity providers" is distributed under the Apache License 2.0 license was the one that I found most suitable. It was one of the libraries that supported both OpenID providers and relying parties. I decided to run JOID within the Apache Tomcat 5.5 application server.

During the design process I discovered that the JOID’s OpenID provider by default stored its users in the ram memory and therefore had no persistence. Because of this I decided to install a MySQL server to be able to get persistence in the users. The system design can be found in Figure 6.1.
6.2 Implementation Results

Installation and implementation of the JOID based combined relying party and provider took about 3 weeks. These were the results:

6.2.1 Prototype Provider

The OpenID provider, which was the hardest to implement, had the following features:

- A user could create a password protected account
- A DynDns [1] Domain was bound to the server to be able to provide an OpenID with a DNS-Host name.
- This account name and (hashed & salted) password was stored in the MySQL database.
- If a user logged in a cookie was placed in their client so they did not have to log in for one hour.

With this implementation goal 1 was completed.

6.3 Prototype Relying Party

The relying party had one simple feature. An input box in where one would enter the OpenID that one wished to use. If the provided OpenID passed all steps in the authentication protocol the provider printed a message indicating whether or not the login was successful. With this implementation goal 2 was completed.

6.3.1 Joomla Relying Party

To test the prototype provider more thoroughly an additional relying party was tested. For this I decided to use the Content Management System Joomla which has OpenID login support.

6.4 Interoperability Results

The prototype provider successfully authenticated with the prototype relying party, the Joomla content management system and the foreign relying party Plaxo. Thereby completing requirement 3 and 4 previously described.
Chapter 7

The Current Security Threats

I have analyzed the security of OpenID though three scenarios, one were each actor 'becomes evil' and tries to exploit the system in some way. In the following chapter I will present the most serious security issue that has been found for each scenario starting with the evil end-user.

7.1 The Evil End-User

In OpenID the end-user can cause problems for a relying party. In the acquiring identifier step discussed in the chapter OpenID the relying party asks the end-user for his or her OpenID identifier.

This identifier is, as previously mentioned, most commonly a URL. It is used in the discovery phase mentioned in chapter 5 section 4. This identifier is acquired through a free text box where the end-user can enter anything he or she wants. If you, the reader, are observant here you will notice that the end-user can force the relying party’s server to contact a URL specified by them.

So what can an evil user do with this?

Get Access to Internal Hosts. First of all one could force the relying party server to contact internal hosts/servers inside the external firewall. Consider if an end-user would enter:

localhost/admin.php?action=sql&query=DROP TABLE user

as his or her OpenID identifier. This would cause the relying party’s server to contact itself through a HTTP-Get request, a serious security threat since most servers have less security precautions against incoming local connections. (Note the url above is just an illustration)

Force Contact to External Hosts. An end-user could also force the relying party to contact other hosts and servers.
CHAPTER 7. THE CURRENT SECURITY THREATS

www.example.com/securityflaw.php?attack=true

If an evil end-user found a security flaw in some website this exploit could possibly be perpetrated remotely from the relying party, thereby possibly placing the blame on relying party.

**Data Flooding.** A relying party would also have to protect itself from unusually large amounts of incoming data via the end-user OpenID URL. For example the OpenID www.example.com/largefile.bin may cause problems for the relying party if the file largefile.bin is very big.

**Denial-of-Service Attack.** Finally a relying party would have to be aware of specialized Denial-of-Service (DoS) attacks. A denial of service attack is when a server is flooded by more requests than it can handle and thus becomes unavailable.

In the case of an OpenID provider a Denial of Service attack would probably not be perpetrated though the normal ICMP (Internet Control Message Protocol) flood. A more effective approach would be to flood the provider with requests of authentication, see chapter 5.2.

This would cause the provider to initiate the normal procedure of authentication. Acquiring identifier, Normalization, Discovery, Association and finally Request Authentication would be processed before the provider would drop the fake authentication procedure.

This can have a large performance impact since the phases after discovery are not stateless. Moreover these steps require the relying party to wait for response from the OpenID provider. This can then cause the relying party to allocate a lot of memory and threads to keep the state information and it cannot be dropped until the authentication is completed or fails.

### 7.1.1 Solution: ID Filters

One way for the relying party to tackle the evil end-user is to filter their input. First of all disallowing all private ip-addresses, specified in RFC 1918 [10], (and also their possible internal domain names) since they should probably not be used as OpenID’s. One could discuss also banning all ip-addresses altogether but this is probably not advisable since some users might actually use their ip-address as their OpenID.

This filtering solves the previously discussed problems where an end-user can force the relying party to contact internal hosts.

Secondly the relying party should consider banning hosts that repeatedly enter identifiers that do not successfully pass the Discovery phase. Not passing the discovery phase indicates that the supplied identifier does not lead to a running OpenID provider. Moreover the relying party should ban hosts that try to initiate too many authentication requests in a short time interval, this to prevent a dos attack.
Implementing such filtering rules protects the relying party from most malicious attacks that the end-user can perpetrate. These precautions are easy to implement and a must for any serious relying party.

7.1.2 Prototype results

Neither JOID nor Joomla did any relevant filtering on internal ip-addresses. This was tested by installing a Tomcat with high logging on the same machine as the OpenID relying party. The Get access to internal hosts is a quiet serious security issue since it enables an outside user to send a more or less uncontrolled HTTP Get request to the server itself or any server on the same intranet.

Some in data validation was done on the Joomla System, for example the > and the < characters were filtered. But yet the local host ip-address should not be allowed as an OpenID.

7.2 The Evil Relying Party

One of the most serious security threats can be realized by the OpenID relying party. The concept is simple, instead of redirecting the end-users to their provider, as described in Request Authentication in the OpenID chapter, the relying party redirects the user to some other web-server.

If this web server presents a page that looks like the provider that the end-user is used to using he or she might try to log into this fake provider. If a weak login method is used, such as a username and password, the end-user has then unknowingly given away his or her login credentials to the fake provider.

This type of attack is called a phishing attack, the relying party is fishing for the end-users login credentials to their OpenID provider. This kind of attack is exceptionally hard to discover for an end-user, the only real telltale sign is the URL. This is the only part of the appearance of the provider that is hard to fake.

7.2.1 Solution: Diverting Phishing

Phishing is a security issue for both the end-user and the id provider therefore I will list solutions for both of them.

End-user

There are two primary solutions for the end-user. The first of them is manual, and the second automated. The manual one was actually mentioned briefly in the problem description, check the url. What I mean by this is that the end-user should briefly check the url of the OpenID provider when attempting to log in. Thare is an example of phising in figure 7.1 and 7.2.

If the url looks suspicious, like a misspelling or a totally different domain then this is a indication that someone may be attempting a phishing attack.
CHAPTER 7. THE CURRENT SECURITY THREATS

The second method is automated and consists of browser add-ons that assist in phishing detection. These automated systems use the same method as the user, they check the domain/url. A good example of such an application is VeriSign’s OpenID SeatBelt Plugin.

While these automated tools are good I would still recommend that everyone starts being more aware of their url’s since this greatly reduces the risk of being the victim for phishing whether it is for getting your bank login credentials or for your OpenID account.

Provider

The solution for the provider is making the login page easy to recognize for the end-user yet hard to duplicate for phishing. There are several ways of doing this, one of the most successful and interesting concepts is the Yahoo Sign in Seal.

The seal typically consists of a small image or a custom text. This so called sign in seal is then bound to a computer so every time that the user logs in form this specific computer their unique seal is presented to them before they sign in. Technically this is done through a cookie or a similar technology called Flash Stored Object. As previously discussed in this report cookies (and also Flash Stored Ob-
7.2. THE EVIL RELYING PARTY

jects) can only be read by the domain that created them. Therefore the phishing party in this case cannot read what user has chosen as his or her custom seal.

If the end-user does not see his or her seal when signing in this should arouse suspicion and hopefully they will then notice a phishing attempt before giving away the user credentials.

The downside of this technology is that it only works for the computer that you created the seal on. Therefore users who constantly change computers won’t benefit from the technology. To clarify there is an example of the Sign in seal as an image. (see Figure 7.3)

Figure 7.3. This is an example of a yahoo Sign in Seal. In this case the KTH-logo in the upper right corner is the seal for this user. A user should get suspicious if their personal seal is not shown when logging in. In this case the Seal is stored in an Flash Stored Object.
CHAPTER 7. THE CURRENT SECURITY THREATS

7.2.2 Prototype Results

If one is familiar with the OpenID library one is working with making a phishing relying party is a quite easy task. For example in JOID this would result only in a minor code change. This is indicated below.

String s = OpenIdFilter.joid().getAuthUrl(id, returnTo, trustRoot);  
response.sendRedirect(s); // Here you can change to have only one static provider.

index.jsp - trunk/examples/server  This is the example JOID relying party. In this implementation making a static provider redirect is a one line code change indicated in the comment.

7.3 The Evil Provider

After reading the technical overview in the previous part of the thesis it should be quiet obvious that the OpenID provider has substantial power. This since this the part that holds the privilege of user authentication.

This means that the provider easily can masquerade itself as one of its users. This can be a serious security problem since one OpenID might be linked to many different services for the end-user.

7.3.1 Choosing the Right Provider

The evil provider is a highly problematic problem in the world of OpenID. Or rather it is a technically unsolvable problem since the architecture is built around a unilaterally authenticating provider. If this provider decides to 'go bad' no one can stop it.

If the problem is unsolvable can one then avoid it? Yes, the key here is the inherent flexibility of the user centric OpenID architecture. The system relies on its users to choose a provider that they trust or if they wish, start their own provider authenticating only them self's. Though this requires the end-users to be enlightened enough to realize how potentially destructive an evil provider can become.

The provider can easily masquerade itself as one of its users, a serious security problem since one OpenID might be linked to many different services for the end-user.

But a system where my only protection against an evil provider is 'choosing the right provider' and hoping it won't go bad isn't very good now is it. So what more can be done?

Delegation of Authentication

One of the keys to protecting yourself against evil providers is to asking the question:
7.3. THE EVIL PROVIDER

Who owns my identity?

In the case of OpenID this question is answer by realizing that the term provider in 'OpenID provider' refers to providing a means of authentication not providing an identity. In chapter 5.4.3 Provider delegation describes how an end-user can use his or her own url as an identity which is in turn redirected (or delegates) to a provider.

This is a pretty unique system since the end-user in this case owns his or her own identity (the url) and only delegates the means of authentication. This is extremely important since it solves the previous problem with an evil provider.

If the provider goes bad the end-user simply re-delegates his or her identity to a new provider with a means of authentication that fits her.

7.3.2 Prototype Results

The key concept in tackling an evil provider is as discussed delegation. This was successfully tested in the testing environment. This was one of the most satisfying steps in the implementation because it’s a very simple solution to a complex problem.

Though it easy for a person with good knowledge about the workings of OpenID it may be hard for your casual Internet user.

On a side note, I sincerely hope that the OpenID community successfully educates its users to realize the importance of delegation. It is basically the only means the end-user has to protect him or herself from the evil provider scenario. (With the exception of starting your own OpenID provider.)
Chapter 8

Discussion

The preceding chapter covered some of the most pressing security vulnerabilities of OpenID today. The first part of this chapter will try to convey some of the indirect benefits that OpenID brings to the Internet community. The second and final part of the chapter will discuss trust in OpenID.

8.1 Fewer Front-End Login Pages

Building a secure login system for a web based service is difficult and often costly. OpenID rids the relying party of having to design and implement a login system. The owner of the relying party can instead implement one of the pre packaged relying party modules that exist in most programming languages today. What are the benefits of this approach?

Instead of each web service having their own designed and implemented authentication solution users will use their id provides probably well tested and well designed solution.

It is also beneficial for end-users since they can choose the authentication system that suits them best, since they can choose the id provider that suits them best. A blind user might for example use a fingerprint reader instead of a keyboard typed password.

8.2 Reduction of Email Address Abuse

A common, and arguably misused, identifier on the Internet today is the email address. Users are commonly asked to supply their email address when creating any kind of Internet based service account. So why is the email such a commonly used identity? Probably since the email is something that users seldom change, they are bound to remember it for a long time. Unfortunately such a promiscuous spreading of the email address has no doubt made it easier for spammers to get hold of target addresses.
CHAPTER 8. DISCUSSION

How does OpenID help this unfortunate situation? OpenID effectively separates the end-users digital identity and their postbox! The OpenID identifier provides the end-user with an identifier that they seldom change, brilliant for use as a username.

The OpenID community hopes that Internet users start to question why web service providers need their email address. Why do they need my email-address? They already have my OpenID.

This might in turn lead to that Internet users might get less promiscuous when it comes to handing out their email address thereby reducing the number of known active spamable email addresses.

8.3 Closing Words

I personally believe that the Internet would benefit from a widespread SSO system, and I think that most security experts would agree with me. OpenID aims to standardize the way of authorization while giving the end-user the power to choose the authenticating party and the means of authentication. In my view this would undoubtedly be a step forward from today’s "one identity per service" view. The goal of this report was to summarize OpenID and highlight some of the problems that are inherent within the protocol itself. Some of the problems are mostly behavioral (such as phishing for user credentials) whilst some are purely a matter of security for the system administrators.

Whether you are an end-user, a relying party or a provider having some awareness of the most common security issues may be key to protecting your system or your own identity.

8.3.1 Future development

One very interesting problem would be to create an OpenID provider that supports reliable end-user identification. Could one create an OpenID provider that utilizes BankID to authentication, and if so, how could the provider guarantee that it uses this strong level of authentication of its users?

A second interesting problem concerns one of the key features in OpenID is provider delegation. This is, as discussed, the feature that enables the end-user to own his or her own identity whilst delegating the authentication to a trusted party. An interesting question is how does one get end-users enlightened enough to realize the benefits of provider delegation.
Bibliography


