SecuringDspace

Contents

Security Models for DSpace Servers

This page discusses how *security concepts* apply to a network service like DSpace and what you need to consider when setting up your site.

About Security

What do we mean by "Security"?

For our discussion, information security involves the following concepts:

- Confidentiality
  This means keeping information *confidential*, i.e. so it is only seen by authorized people. For example, it includes not allowing your password to be seen as you enter it when logging in. Confidentiality is usually implemented by *encryption*.
- Authentication
  Authentication is the process of proving your identity to the system, e.g. logging in as an EPerson in a DSpace session. It is performed by the Stackable Authentication modules.
- Authorization
  This means granting permissions - *authorizing* access - to resources. It relies on *authentication* to provide your identity; it has to know who you are to tell what you should be allowed to do.
- Encryption
  A system of encoding that lets two parties (e.g. your Web browser and the DSpace server) exchange data without it being readable to anyone else. In the context of Web applications, encryption usually implies the **HTTPS** (HTTP over SSL) protocol.

Why do we need security if DSpace is open?

Even if all of the digital materials stored in your DSpace archive are available for open access by the world, you still need *some* security. At the very least, you probably don't want everyone visiting
the site to be able to add to, modify, and delete the contents of the archive.

Some archives contain materials that should only be visible to a select audience. Those sites can use the the usual DSpace access-control mechanism to grant "read" privileges to certain users and groups.

Even if your archive is relatively open, *you still need to consider the security of the login process for your EPeople*, as described in the next section.

Practical Security Solutions

Threat Model

Security people have a term, *threat model*, which describes the threats and risks of *compromise* (disclosure of confidential information) you expect to encounter. In the case of a network service like a Web server, the threats you should realistically expect include:

- Attackers attempting to gain unauthorized privileges by some method; if they get access to a DSpace Administrator login, they can do whatever they like to your archive.
  - Monitoring of network traffic ("sniffing packets") is commonplace, especially in university environments; this can reveal confidential information if it was not encrypted.
  - Session stealing (enabled by packet sniffing), using the credentials of an established web session.
  - Guessing passwords and "social engineering" are, unfortunately, legitimate risks, but we cannot address them here.
- Reading confidential materials as they are accessed by an authorized user.
- Subverting IP-based authentication through a proxy or IP spoofing.

Note that I'm not mentioning the many other types of threats because the solutions are not discussed here – e.g. denial-of-service attacks, SQL injection, etc.

Addressing the Threat Models

Packet Sniffing

When do we have to worry about the contents of packets on the network being seen by a third party? Typically, only when the user is entering a password, such as on the login page and the administrative page
where the password is changed.

The solution is to force the Web transactions containing a password to be encrypted. Assuming we are *only* concerned about pages with HTML forms, the simplest way to force encryption is to force the pages containing the password forms to be downloaded with HTTPS. If the browser tries to load that page with HTTP, the server should *redirect* it to the equivalent HTTPS URL.

Note that it isn't strictly necessary to load the *form page* with HTTPS; the form itself can be transmitted unencrypted since it doesn't contain a password at that time, only when the form is filled in and POSTed to the server. However, the form tag typically has an `action` attribute containing a *relative URL*, so on submission, it gets sent to whatever host and port were on the page that got loaded. Thus the page has to be loaded with HTTPS.

If you change the UI code so the password forms always use HTTPS URLs, then it is not necessary to use the redirect technique above.

Session Stealing

DSpace establishes a login session, which identifies your EPerson, by handing your browser a cookie. The cookie is not marked secure so it gets transmitted on non-encrypted connections, so it is subject to being observed and "stolen". So, the threat model is that an attacker could steal your cookie and get your level of access to DSpace, which is significant if you're an Administrator.

As of DSpace Release 1.4.1, this risk has been greatly mitigated because the `Authenticate` class now binds a session to the client's IP address. If a client on a different IP address presents that same session cookie, it is rejected with an error. This is not foolproof, since it *is* possible to spoof IP addresses, but that is a very sophisticated attack that you may not ever expect to have deployed against your DSpace archive.

If you *are* that concerned about session stealing, it's probably worth requiring encryption on all transactions on your web server. We expect this to be an unusual case.

Implementing Security

This really comes down to answering the question "where do I need to enforce HTTPS on Web transactions".

The answers are, for the DSpace 1.4 Web UI:

- If you don't use client-certificate (X.509) authentication, then only the servlets where password forms are presented need to be encrypted.
- If you *do* use client-certificate (X.509) authentication, then all pages requiring authentication must be redirected to HTTPS.
- If you are extremely security-conscious or dealing with very confidential materials, then only allow HTTPS connections.

For actual solutions, see these other pages:

- ServletSecurity – implementing security in the Java Servlet container (e.g. Tomcat)
- ModJk – How to use Apache and mod_jk, and mod_rewrite to enforce HTTPS.