Introduction

For the term project, I designed and implemented an access control system for XML. I wanted the system to be powerful and easy to use from both an end-user and programming standpoint. In the project proposal, I planned to extend an existing (open-source) Web server to have built-in access control capabilities (similar to Apache's ability to control access at the document-level). After examining the design and other aspects of that approach, however, I decided to write a handler instead that could easily integrate with existing XML services and applications. These revisions were approved by Dr. Shi.

Design/Specification

The access control system I designed uses XML's own capabilities to define access controls. The actual configuration is defined through *allow* and *deny* policies. The model will interrogate users requesting access to a document by their username, group name, ip address, and symbolic address (hostname).

Since XML's markup is hierarchical (a tree-based data structure), its nodes are represented through nested elements. This allows access control at an extremely fine granularity level. To take advantage of this, I made it so that access policies are defined at the granularity of elements using an element's attributes. The handler will parse an XML document and check each element's attributes for the following access controls:

allow_user: list of users that have propagating access to the element
allow_group: list of groups that have propagating access to the element
allow_domain: list of networks that have propagating access to the element
allow_ip: list of IP addresses that have propagating access to the element
deny_user: list of denied users
deny_group: list of denied groups
deny_domain: list of denied networks
deny_ip: list of denied IP addresses

The access control engine will calculate the visibility that the user has to each element based on the access control policies found for each element, checking it against the requester's information (user, group, network domain, ip address). The document will be transformed accordingly. This is an example of how an element with access control attributes might look:

```
<chapter deny_domain="duke.edu,kansas.edu,cu.edu">
```

Obviously, for this to work it must be running in a multi-user, multi-group environment that can also resolve the hostname and ip address of a connection. Most systems that will have a need for an XML access control system will support this. Since my system is intended to be compatible with multiple platforms, it does not perform name server lookups or resolve usernames and group names. That is the responsibility of the calling application (e.g., Web server), and it must pass that information to the handler via command line arguments:

```
xmlhandle <xmlfile> <username> <groupname> <domain_name> <ip_address> [-
s style_sheet] [-a]
```

There are several limitations with demonstrating this system in class: it must use Apache, I don't have root access to the system it will be running on, and the username and group name will not vary. I came up with a work around—I will emulate a system via web forms and a simple CGI script that calls the handler. Due to the limitations stated above I believe (along with Dr. Shi) that this is the best solution.

One final thing worth mentioning is the importance of my decision to use XML's own capabilities to implement this system; since it does not extend XML itself, all it requires is creating the policies within the XML documents (adjusting any DTDs and schemas accordingly) and configuring the server to use the handler to transform the content.

Implementation

I have ample programming experience with both the Java and C++ programming language, and both languages are object oriented, but there are key differences that lead to my decision to implement this project in C++. Even though Java has lots of cool libraries and API's for managing, parsing, and processing XML, I felt that C++ was a better choice due to its speed, portability (more environments seem to have C++ compilers than JVMs), and most of all it would be more fun and challenging. Rather than write an XML parser from scratch, I used the following open-source/open-license libraries:

libxml2 (http://www.xmlsoft.org/) an xml processing toolkit written in C.
xmlwrapp (http://pmade.org/software/xmlwrapp/) a C++ wrapper for libxml.

Xmlwrapp is a C++ wrapper for libxml which means that it consists of C++ class interfaces that encapsulate libxml and take full advantage of all the additional features C++ has to offer. Libxml/Xmlwrapp support SAX, DOM, XSLT, XPath, XPointer, DTD validation, and uses only standard libraries to allow for easy embedding.

I decided to use DOM rather than SAX in the use of libxml/xmlwrapp because it allowed for a simpler and more intuitive design. The document is then processed recursively, filtering out nodes that the user is denied access to. Access to each element is allowed by default to minimize the configuration necessary.

Scott Baldwin
883985