Security Issues for Web-based Applications:  Issues and solutions for the safe transfer of Clinical Trials data over the Internet

Andrew Fagan
Chris Olinger
iBiomatics LLC, a SAS® company

ABSTRACT

When building any web-based application, the issue of security is always a concern. When the application is designed to allow the sharing of clinical trials data across the Internet, security becomes one of the major topics. This paper describes what approach iBiomatics, a SAS company, has taken in the construction of their iBiomatics Portal. The paper presents a basic overview of security for an Internet based application, describes what security features were built into the application, what software tools were used to implement various aspects of the security model, and how the actual hardware configuration can be a critical piece of the overall security of a web-based application.

While the iBiomatics Portal is used as the example for this paper, the concepts presented here apply to most Internet-based applications. The primary product of iBiomatics, the iBiomatics Portal provides a secure, Internet-based application for the warehousing, analysis and review of biomedical data. The figures used in this paper show the various components of the iBiomatics Portal where appropriate. You can read more about iBiomatics products at: http://www.ibiomatics.com/solutions.html

THE BASIC PRINCIPLE: THINK ABOUT SECURITY FIRST

Before sitting down and developing the functional pieces of any Internet based application, you need to think about how secure you want your application to be. Trying to develop a security model on top of an existing application, or retrofitting one after the majority of the application has been developed, can be disastrous. Understanding exactly what the security concerns are up front must be a basic part of your design.

The Internet is a different medium for application development. The decisions you make regarding what you want to secure and when will dictate major portions of your application design. Before you build your basic object model, ask yourself some questions:

- Do users need to be defined to the system, or can anyone access it?
- How will I store user ids and passwords?
- Should users within the application be able to share information?
- Are there parts of the application I want to restrict to only certain users?
- Should users have access to any of the files or tools on their desktops?
- Will the application reside only an intranet, or will it be available on the Internet?
- Will data be sent across the Internet?
- Will user information be sent across the Internet?
- Are there other security requirements due to the nature of the application?

There are many books on Internet security and many experts exist in the field to help you address each of these concerns. But as long as you are thinking about the issues, you are at least off to a good start in your design.

BREAKING THE PROBLEM DOWN

There are at least three basic areas where you can address security within your design to help you solve each of your concerns.

The first is hardware. Commonly overlooked, this is a critical piece of your design and will affect what you can do to help with security in each of the other two pieces.

Second is the language and tools used to develop the application. By choosing the right technology, you can buy yourself some security advantages with almost no additional work.

And finally, the application itself. By understanding what parts of the application are critical, you can design your strongest security features around them.

HARDWARE

How many times have you heard “we’ll put it behind a firewall so everything will be safe”? A firewall is but one small part of the overall hardware design for security, and a firewall alone is a pretty thin defense.

First, what exactly does a firewall do? A firewall is a piece of hardware, with associated software, that limits what types of connections are allowed. A firewall is typically used between an open Internet connection and a company network to restrict access to the company network. Firewalls can limit access in two ways: by restricting what port someone can connect on and by restricting what IP addresses can connect.

For example, you can configure a firewall between your company network and the Internet connection to only allow connections from a short list of IP addresses, such as your subsidiaries. A user logged on to a machine in one of your subsidiaries would be allowed through, while little Johnny, sitting in his garage, would not be allowed to because his machine would not be on the approved list. IP filtering is often used in this way to restrict access to only affiliated companies.

Port filtering can be used to prevent random network hits from getting through to your private network. By configuring the firewall with a specific list of ports to allow, a user trying to gain access to your network must know the exact port number to try to connect to. A random attempted connection on one of the default port numbers would fail.
Assuming you have protected who can get in to your network, you still have to deal with the issue of what happens to the information that goes in and out. Once it is on the other side of the firewall, your data is open to theft or corruption if left unprotected. This where a secure socket connection comes into play. Have you ever gone to a web site and when the time comes for you to provide some confidential information, the browser pops a little window telling you it is switching to a secure connection? And if you look, you will see that your URL now starts with https: rather than just the normal http:. That little “s” on the URL tells you that you are now transferring information across a secure socket connection. What this means is that all the data transferred between your web browser and the web server, wherever that may be, is encrypted. The data is encrypted at your browser, and decrypted at the server. Anyone trying to “listen in” on the data as it passes over the Internet will only get garbled bits and bytes. Without the corresponding decryption algorithm, the data is useless.

This leads to the topic of encryption. Any encryption technique can be broken, right? Well, true, it’s just a matter of time and effort (except for the very newest encryption techniques which are mathematically proven to be unbreakable). To some degree, it is just like protecting your house or your car. The harder you make it on the thief, the less chance there is that he will pick on you. Currently, the best encryption techniques use a 128-bit cipher strength for encryption. It is commonly agreed that this type of encryption is so difficult for the common hacker to break as to be essentially impenetrable. And as far as you (the application developer) are concerned, it is almost trivial to implement.

There are currently laws restricting the use of 128-bit cipher strength outside of the United States, however. In cases where either the web browser or the web server will be located outside of the US, a 56-bit cipher strength is the highest level of encryption you will be able to use. Although this is markedly less difficult to crack than a 128-bit cipher strength, 56-bit cipher strength is still very strong and is more than sufficient for most applications.

**HARDWARE: A PICTURE OF THE PIECES**

So let’s put some of these pieces together to try to build the basics of a secure application, at least from the hardware standpoint. The figure shows the typical configuration for an iBiomatics Portal, which is accessible on the Internet.

Referring to Figure 1, the connection from the web browser to the web server is always done using https: This ensures that all the information transferred between the browser and the server is encrypted. Wherever possible, 128-bit encryption is used.

Between the web browser and the web server, a firewall is configured to do port filtering. The web browser must request connection to a specific port or the firewall will block the connection. Note: this firewall could also be configured to do IP filtering if the iBiomatics Portal was being deployed only to selected sites. For example, if a multinational pharmaceutical company had a set number of sites that were allowed to access the information, the IP addresses for the machines at those sites could be configured into the firewall, blocking access from all other Internet connections.

On the backside of the firewall, the connection to the web server is from the firewall to an internal IP address for the web server. Only the firewall knows this IP address, and the web server will only respond to a request from the firewall. This ensures that no one can talk to the web server without going through the firewall.

The web server then makes requests of the other hardware on the network, but only through a second firewall. This firewall is set up to do IP filtering and port filtering. Only the IP address of the web server will be allowed through the firewall. And once again, only the firewall knows the internal IP addresses of the other machines on the network. And the firewall will only respond to requests on a single port. In this way, we ensure that all communication with the internal network originates at the web server and all of the traffic goes through the second firewall.

![Figure 1: the iBiomatics Portal basic hardware configuration](image)

**THE TECHNOLOGY CHOICE**

You can develop a web (or Internet) based application using any number of technologies. SAS provides a number of choices itself. SAS/Intranet®, Design Time Controls, Integration Technologies in concert with other languages, etc. The choice you make, however, will dictate exactly what you can do in the application and how secure it is.

One of the primary fears of anyone running an Internet based application in their web browser is: what can this thing do to my data on my hard drive? Can the application see what is on my hard drive, and can it modify it, or even delete it?

Hardly a week goes by when you don’t read a notice about some new virus or another. While most of these are spread via email, the potential is just as great with a malicious Internet application.

The key here is to weigh the security concerns with your other design criteria. There is always a price for making an application more secure, whether it is in performance, user friendliness, or accessibility.

By choosing an application environment designed for deployment over the Internet, you can save yourself from having to implement much of the required security functionality. And by using technology your customers are familiar with and trust, you can more easily convince them of the security of your application.

The iBiomatics Portal is a collection of existing technologies, but the entire application is based on the Java 2 Enterprise Edition or J2EE. This paper is not an advertisement for Java, but there are a number of features of the Java language and architecture that should be pointed out.

First and foremost, Java was designed to be a development environment for web based applications. The designers of the language envisioned how a web application would be deployed and what the concerns might be. They thought...
about restricting the application’s access to data on the hard drive and
the language enforces it. That is the key point: the application
developer does not have to take on the role of enforcing when data is
accessible because the language itself dictates the rules. Java is very
unforgiving to the application developer in this respect: the language
gives you access to only what you need and only what the actual user
allows you. Your application cannot step outside of its bounds without
the user explicitly (and knowingly) allowing it.

For example, an applet running within a web browser is actually
downloaded to the user’s machine when it is run. By default, the
applet has the minimum amount of authority allowed on the local
machine. It has read access to the directory where the applet is
running from, and nothing else. The only way an applet can have
greater access to local resources is if the user edits the local policy file
and grants the applet more authority. So unless you go out of your
way to let an applet do more than it can by default, it can’t do any
harm.

Contrast this with other technologies which were designed for general
applications development but have been amended for web
applications development. An ActiveX control, for example, written in
either C++ or VB, by default has complete access to every resource
on the local machine. Whenever an ActiveX control is run, you trust
the developer of that control to have good intentions and not be
malicious. The responsibility here is on the developer of the control,
ot the person running it.

There are many other features of the Java language that make it well
suited for developing secure web-based applications. If you want to
read more on this topic, Sun maintains an excellent web site about
Java at http://developer.java.sun.com. A good starting point is the
white paper on Java security at

There are any number of ways to use technology to develop a secure
web-based application. Java is one of them, and it does much of the
work for you. By using it effectively, a secure application can be
developed quickly and easily, while giving the end user control over
what resources are available and which are not. When developing
your application, think about the technology you will use to develop the
application and what implications that will have on the security of your
final product.

THE APPLICATION ITSELF

By now, we have determined what type of hardware we are going to
use and how the network will be configured. We have also chosen a
development language and associated technologies with which to
implement the application. Now we have the final piece of the security
puzzle: how the application itself deals with security.

Going back to the series of questions at the beginning of the paper, at
this point the answers to many of those questions come into play. If
the application is available to the world and doesn’t need to know who
individual users are, your job is half way finished already. Of course,
that wouldn’t make for much of a paper on security so we’ll look at a
bit more realistic example.

For the iBiomatics Portal, every user must be known to the application
before they can use it, and within the application itself, there are
various levels of access. The application was designed to be run
across the Internet, although deployment within a company’s own
intranet is an option. The answers to the specific questions are thus:

- Do users need to be defined to the system, or can anyone
  access it
  - Users must be defined ahead of time
- How will I store user ids and passwords
  - Not known but they must be accessible both when the application is first
    invoked and at various times during its execution
- Should users within the application be able to share information
  - Yes, each user must be able to decide what information is “private” and what
    should be visible to others
- Are there parts of the application I want to restrict
  - to only certain users
  - Yes
- Should users have access to any of the files or tools on their desktop
  - Yes, in some cases
- Will the application reside only an intranet, or will it be
  available on the Internet
  - Designed for the Internet, available on an intranet as well
- Will data be sent across the Internet
  - Yes, confidential data
- Will user information be sent across the Internet
  - Yes, in some cases
- Are there other security requirements due to the
  nature of the application
  - Yes, given that the application handles clinical data, government regulations
    (specifically, 21 CFR Part 11) must be followed

These answers dictate in many ways what the application
must do in terms of security. First, since every user must be
known prior to using any of the features within the
application, the first thing a user sees has to be a login
screen. And the user should not be able to get to anything
else before they provide a valid user id and password.

For the iBiomatics Portal, this means that upon entering the
correct URL, the user is presented with a very non-descript
login screen. The screen itself gives no indication of the
content of the Portal, nor who the actual owner of the
content is. The login screen itself is stored in a directory
apart from the majority of the application so that only the
bare minimum of icons and files are accessible prior to
authentication.

Once the user is properly authenticated and granted access
to the Portal, the content and functionality they have access to is determined based on their user privileges. And
because of industry regulations, particularly 21 CFR Part 11,
the way they access certain functionality is also restricted. If
your application only requires that a user be authenticated at
login, and there are no distinctions within the application
concerning who can do what, you could probably opt for a
simple LDAP server to hold your user names and
passwords. But the requirements on the the iBiomatics
Portal make a robust user and function security model within
the application necessary. And this places some constraints
on the design and implementation.

All of the details of 21 CFR Part 11 cannot be captured here
(see References below), but a few of the major points that
impact the security model are that user ids and passwords
must be stored forever, the passwords must be changed on
a regular basis, and the system must be able to trace an
action back to a user even after that user is no longer active
in the system. In addition, the guideline states that certain
actions within the application must require subsequent
authentication (a single user id/password at login is not sufficient).

The iBiomatics Portal stores all of the user and password information in a database table that is accessed by the webserver directly and also by the application. This solution allows for a single copy of the information to be accessible by both and ensures that any updates to the information are instantaneously reflected everywhere in the application. For example, if a user changes her password during a session, the next time she attempts to save changes to an object, she will be prompted for her password. All of the passwords are of course encrypted in the database table, and only a single system user id has access to the table. If you really want the technical details on how security is handled within the WebLogic webserver as used in the iBiomatics Portal, see the references below.

In addition to basic user authentication throughout the application, the iBiomatics Portal restricts a user’s access to objects and functionality based on his role and privileges. Every object, each time it is accessed, checks whether the user is allowed to perform the requested action (read, write, delete, etc.). The object security model is analogous to a tree structure, where a user can be given access to all objects “below” a certain point in the tree, or just to a specific branch of the tree. Even if a user could somehow get to a portion of the hierarchy she was not allowed to access, she could never open any of the objects because each object checks the user’s permissions. And even if a user could somehow defeat the interface and get to some set of functionality that was off limits to them, the functionality itself would prevent the user from performing the action. All security failures of any sort are stored in the audit trail for the iBiomatics Portal and can be reviewed at any time.

User authentication, object security and even functionality restrictions are fairly obvious. But it’s the little things that will burn you. For instance, putting a user id or password into the HTML for a web page, where any user who views the source can see it. Or placing more than just the login screen in a directory that is accessible without authentication. Or what if your application allows the user to invoke the SAS system, including display manager, as the iBiomatics Portal does? Once SAS is up, a user can type “X” on the command line and get a command prompt on the server where SAS is running. That’s probably not what you want. Check out the list of options you can specify upon SAS invocation to restrict what the user can do. NOXCMD is certainly one to note.

Figure 2: User authentication within the iBiomatics Portal
CONCLUSION

Building a web application that enforces all of your security concerns is possible - although more difficult - than for a conventional client/server application. The web introduces a host of security concerns, but the real issues are in how you design your application. Take the time up front to consider all of the security constraints you require and then do some research on how best to solve them. There are enough businesses doing real work on the web now that most of the issues you come up with will have already been faced by someone else. Learn from their experience, and mistakes, and you will end up with an application that is as secure as you want it to be.

REFERENCES

Java Language References:
http://developer.java.sun.com


WebLogic Security:

CONTACT INFORMATION

Andrew Fagan is Director, Research and Development at iBiomatics, LLC, a SAS company. He has a Bachelor of Science in Computer Science degree from the Pennsylvania State University, University Park, PA. He has worked at iBiomatics and SAS Institute for more than 13 years, with over 10 years dedicated to software for the pharmaceutical industry.

Your comments and questions are valued and encouraged. Contact the author at:

Andrew Fagan
11000 Weston Parkway
Suite 110
Cary, NC 27513

Phone:  +1 (919) 653-2700
Fax:  +1 (919) 653-2701
Email:  Andrew.Fagan@iBiomatics.com
Website:  http://www.ibiomatics.com

ibiomatics and SAS are registered trademarks of SAS Institute Inc. in the USA and other countries.

Other brand and product names are registered trademarks or trademarks of their respective companies.