Providing secure remote access to legacy health care applications

Andrew Young, School of Sciences, University of Salford

David Chadwick, Information Systems Institute, University of Salford

John New, Salford Royal Hospitals NHS Trust

Abstract

While the widespread adoption of Internet and Intranet technology has been one of the exciting developments of recent years, many hospitals are finding that their data and legacy applications do not naturally fit into the new methods of dissemination. Existing applications often rely on isolation or trusted networks for their access control or security, whereas untrusted wide area networks pay little attention to the authenticity, integrity or confidentiality of the data they transport. Many hospitals do not have the resources to develop new "network-ready" versions of existing centralised applications.

In this paper, we examine the issues that must be considered when providing network access to an existing health care application, and we describe how we have implemented the proposed solution in one healthcare application – namely the diabetic register at Hope Hospital. We describe the architecture that allows remote access to the legacy application, providing it with encrypted communications and strongly authenticated access control but without requiring any modifications to the underlying application. As well as comparing alternative ways of implementing such a system, we also consider issues relating to usability and manageability, such as password management.

Keywords

Security, healthcare, remote access, public key cryptography, certification authorities, digital signatures, authorisation, authentication, X.509, PKI, CGI

Underlying problem to be tackled

As the proliferation of computer networks continues to increase, many health care organisations are finding that they are unable to realise the full benefit of such technology. While the networks allow data to flow, their existing data processing applications are not designed in a way that allows them to make use of this. This means that a user's expectations of mobility and near-instant access to information cannot be satisfied. Problems that will be encountered are:

? There may be rigorous access control relating to the data, legal requirements covering disclosure, or commercial considerations regarding sensitive data. While Network Operating Systems provide file-sharing facilities, the access control is usually based on simple authentication (name/password) and this is insufficient for protection of important data. Strong authentication (e.g. one-time-passwords, digital signatures) is required.

? There may be a need for confidentiality and integrity of the data whilst in transit, to ensure unauthorised users cannot view or modify data. Whilst media attention usually focuses on the lack of security of the Internet, packet sniffing on a Local Area Network is significantly easier than intercepting data on the Internet.

? Sharing data and applications across a network raises the possibility of multi-user access to the data. Many existing systems assume one user at a time, and do not prevent user operations interfering with each other

For these reasons, many health care applications operate on dedicated PCs (locked in a secure office or protected from the rest of the network by a firewall) or operate on a network with inadequate protection.

The above is certainly true in the UK National Health Service. Over the past decade, many chronic disease management programmes have been implemented, invariably supported by Information Technology, often as stand-alone or centralised applications. While these have usually been developed by secondary care (hospitals), much day-to-day treatment of chronically ill patients is performed by primary care (general practitioners - GPs) who have limited access to the data and limited opportunities to feed data into the

system. Tertiary care (specialists) is also likely to have limited access to the data if provided by a different hospital. Additionally, sharing of data within a hospital is often loosely controlled and usually unencrypted.

Sharing of data between all participants in the healthcare process will offer significant improvements in procedures. All physicians will have access to the latest information, leading to better patient care. If new data can be input directly, the time-consuming and error-prone task of re-keying can be eliminated. To achieve this, consideration must be given to strong authentication of participants, strong encryption of all communication between them, and audit trails to provide accountability.

The problem is not specific to the healthcare sector. Commercial organisations treat their internal company-confidential data (financial results, bids in preparation, board minutes) as carefully as healthcare professionals treat patient-specific data. Many organisations may find it useful to make such data available if they could ensure that:

? A well-defined group of users…

? …can perform a well-defined group of operations…

? …on a well-defined set of data…

? …and that other users can neither view nor modify the data, queries or results.

In this paper, we present an architecture that has been implemented to add these facilities to an existing centralised diabetic database application at Hope Hospital, but the architecture may be used by many different legacy applications in different public and private sectors.

The existing architecture

Salford Royal Hospitals NHS Trust operates a Clinical Information System, which holds current and historical data on 40000 patients with Asthma, Coronary Heart Disease, Diabetes and Haematology disorders. Until recently the system operated as a collection of standalone MS-DOS computers using the Smartware DBMS (a LAN based low security version with an SQL DBMS and Windows based fat-client

has recently been deployed). Users are centrally registered onto the system, and can be given rights to access patients for a particular hospital, consultant physician or GP.

Information is distributed to GPs on an annual basis as a printout of data for their patients. Data is returned to the hospital via hand-annotated "clinical review" forms, which are re-keyed by hospital staff. Within secondary care the situation is no better, and most consultations produce hand-annotated forms which are re-keyed. Specialist referrals rely on the hospital posting a copy of the printed form along with a referral letter, and re-keying information from a free-form letter returned after any treatment.

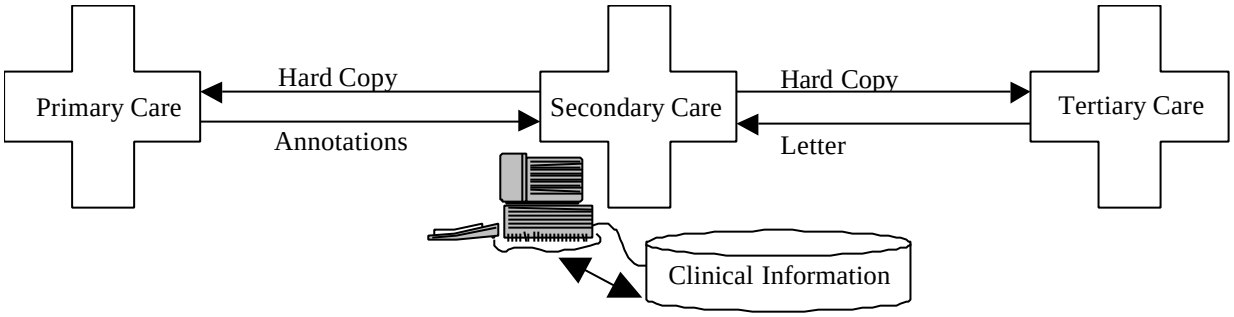

Figure 1 – current architecture

This architecture has a number of significant drawbacks:

? inflexibility - the information available to doctors is fixed, changes in requirements are hard to accommodate;

? inefficiency – someone in the hospital manually distributes information and enters any returned data;

? unreliability – data submitted to the database is hand-written, read and re-typed, each stage may introduce errors;

? latency – because hard copies are sent annually, it is likely GPs do not have the most current data.

While the current system gives access to a good quantity of data, there is scope for improving access to the data and therefore its quality. This should lead to an improvement in patient care.

An architecture with networking

Our goal was to give primary and tertiary care practitioners direct access to the database, thereby removing many manual processes currently taking place, including re-entry of data. It would remove the latency and would make it much easier to vary the set of data that doctors receive.

Initially, some hospital system administrators that we consulted preferred to allow external users to dial directly into the hospital network to access the database, as this approach was under their control. In practice, it suffers from several problems that apply equally to hospitals and commercial organisations:

? scalability - how many dialup lines should an organisation provide for its users;

? inefficiency – every organisation would need to provide their own pool of dialup lines, with no sharing of expensive resources or economy of scale;

? security - an external user would potentially have access to the entire organisational LAN, rather than just the systems they needed to access. It is unlikely an organisation's security policy (in our case, the NHS "code of connection") would permit such an arrangement.

We proposed that a wide area network provides an efficient way of allowing external access. Each hospital needs a connection between their secure LAN and the insecure WAN, and this is protected by a firewall configured according to local security policy (many hospitals and other organisations already have this infrastructure in place). The WAN provider takes care of dialup capacity and network management, and the firewall ensures that external users can only reach appropriate internal systems.

While hospitals are generally co-operative, never the less they have to be very careful about who they let onto their systems from an external network. Commercial organisations are usually even more suspicious. The firewall is a central place where an organisation's security policy is enforced, and this allows organisations to share information even if they don't have complete trust in each other as it allows them to control what they are sharing and who can gain access.

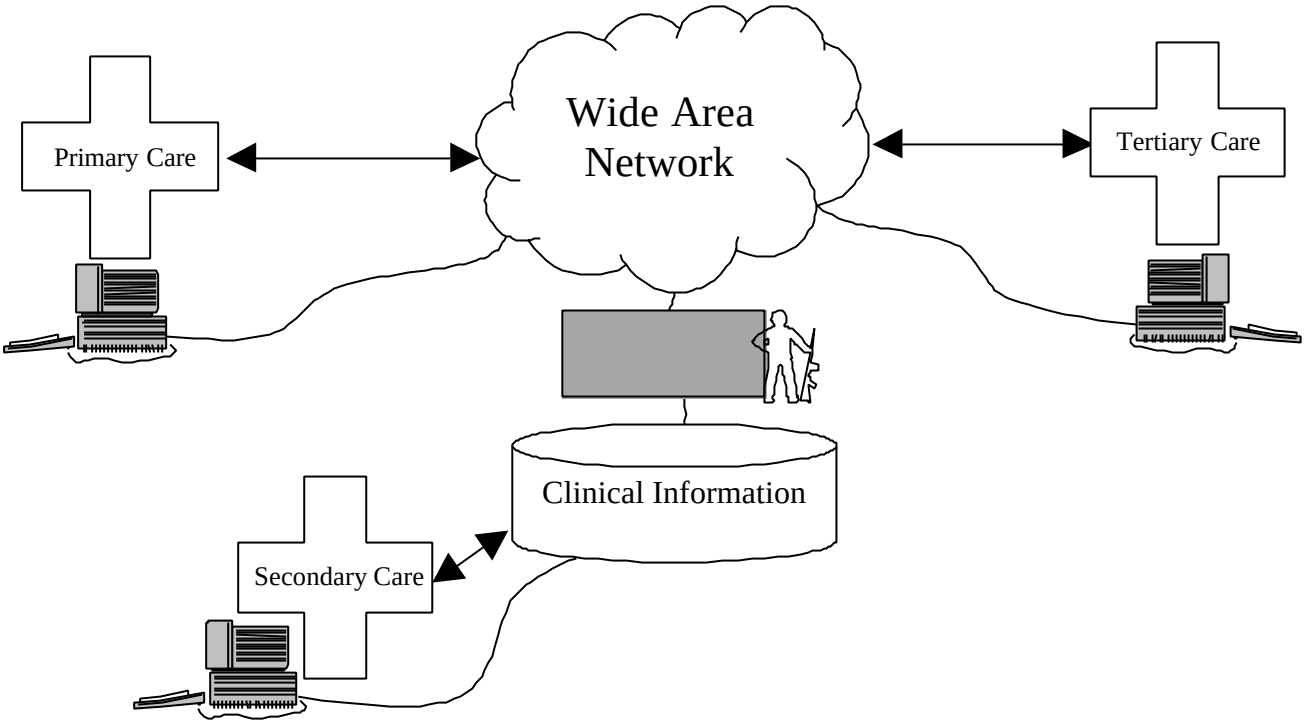

Figure 2 – architecture with networking

In the UK healthcare sector, there are two WANs which could be used – the Internet and NHS-Net (a trusted national TCP/IP based Intranet). While NHS-Net is intended to connect all GPs and hospitals by the end of 2001, this goal is far from being reached and many GPs currently use the Internet.

As both are TCP/IP based networks, any application developed will work in both environments. However the two networks have very different trust characteristics.

? The Internet is untrusted. Any application communicating over the Internet must take steps to protect the integrity and confidentiality of the data transmitted and to obtain assurance of the identity of communicating partners. This is possible through technologies such as one-time passwords, digital signatures and public key encryption.

? NHS-Net (and a hospital's private TCP/IP network) is deemed to be trustworthy due to the way it is managed. An application that operates solely on a trusted network will generally be deployed without

use of security software. However even this perception is slowly changing, as the realisation dawns that it is virtually impossible to fully protect all the access points onto the NHS-Net.

Given the common technology between the two, a secure application can clearly be deployed on a trusted network. A single implementation can be developed and used by all doctors, whether they connect via the Internet, NHS-Net or a private TCP/IP based LAN.

Designing the system for Internet use gives patients the capability to view their own medical records (subject, of course, to a rigorous system of authentication and a user interface that is easy to use and hard to abuse). This is in accordance with UK government policy [NHS2].

Adding security to the architecture

Deployment of security software needs to follow two basic principles:

1. Maximum use of commercial off-the-shelf software based on open standards. Security software and standards are evolving so quickly that it is impossible for a research project to implement proprietary software with any significant shelf life. Use of commercial software from an established vendor ensures an upgrade path in the event of future technological developments, while open standards provide interoperability.
2. Minimum inconvenience to end-users. Experience has shown [Whitten, Chadwick] that deployment of security software is more likely to succeed if it asks little of the users. Enthusiasts may be willing to experiment and accept inconvenience, but a significant number of users want to continue using their existing software and hardware with security added as transparently as possible.

As the data resides inside the hospital and the user is remote, we propose a client-server application to allow the two to interact. One option is to use an established client-server-pair (such as a WWW browser and server), another is to produce a new application-specific client-server-pair.

With a WWW solution, training is minimised as users are generally comfortable operating a browser. Development work is also minimised as browsers exist for all operating systems and platforms. However, a browser retrieves HTML formatted data from the server and is constrained by HTML in the ways it can be displayed and the ways a user can interact with it. The requirements to minimise inconvenience to end-users and to maximise use of commercial off-the-shelf software are met, but at the cost of restricting the flexibility of the user interface.

A customised client can retrieve data in arbitrary formats and use any available output and input devices to display the data and allow the user to interact with it. However, development work is required for all supported operating systems and platforms, and training will generally be required for inexpert users. This option allows the developer to provide a powerful user interface, but at the cost of inconvenience to end-users, extra development work and subsequent support to cope with evolving standards.

The choice depends on factors such as the nature of the data being transferred, the way the user will interact with it, and the development effort available. For our project, doctors wanted to access electronic versions of forms they already knew in paper-based format. The forms were easily representable in HTML, and users' familiarity with the existing forms and WWW browsers would minimise the training required. Therefore, a WWW solution would provide an acceptable user interface with minimum inconvenience and maximum re-use.

For both types of client-server architecture, security could be provided through the Secure Sockets Layer protocol [SSL3, RFC2246] or by a more sophisticated plug-in, such as Entrust/Direct [Entrust]. Both provide a secure application level connection between the client and server, based on digital signatures and X.509 [X509] certificates. Alternatively, virtual private network technology could be used to create a secure network level connection between the client and server networks. This is of limited interest to our project as it generally provides per-site security whereas we require per-user access control.

SSL and Entrust/Direct can be compared using a number of criteria:

? **Availability**: SSL is ubiquitously available for common web browsers and servers on all platforms. Source code and commercially supported toolkits are available. Entrust/Direct browser support is available for Windows and Macintosh, with server support for Windows NT and commercial Unix. While toolkits are available, source code is not.

? **Services**: SSL provides client and server authentication of http requests and responses. Entrust/Direct provides a permanent non-repudiable record of a transaction by digital signing of HTML forms using the PKCS#7 standard [PKCS7].

? **Trust**: Standard WWW browsers implement a user-centric trust model where end-users make trust decisions (often based on limited information). Entrust/Direct provides centralised trust management, where the Trusted Third Party makes trust decisions on behalf of the users, according to well-defined published policies.

? **Certificate management**: Standard WWW browsers require manual key backup and manual certificate renewal. Entrust/Direct provides a managed infrastructure with key recovery and automatic transparent renewal of keys

? **CRL management**: Standard WWW browsers requires users to manually check if a certificate has been revoked, though server plug-ins are available to allow servers to do this automatically. Entrust/Direct automatically checks CRLs in both client and server.

? **Encryption**: The standard WWW browsers available in the UK during this research provided only 40-bit encryption. 128-bit encryption was available via plug-ins [e.g. Fortify], and browsers with native 128-bit encryption could be obtained with a suitable export license. Entrust/Direct provided 128-bit encryption though was subject to Canadian export regulations (export licenses had always been straightforward to obtain for British companies, especially in market sectors such as healthcare; recent changes in regulations have made this easier).

? **Protocols**: While all standard WWW browsers contain a built-in implementation of HTTP-over-SSL, other client-server protocol can easily be secured using SSL toolkits. This allows customised clients to use proprietary protocols. Entrust/Direct only works with HTTP, though a customised client can make use of Entrust/Direct's security features if it uses HTTP as a transfer mechanism (Entrust provide a GSS-API conformant toolkit if proprietary protocols are required).

Whilst SSL and Entrust/Direct are both capable of providing good security to an application, each is suitable for a distinct type of user. The SSL protocol (actually the implementation provided in standard WWW browsers) is more suitable for a community of users that: are computer literate and educated in the concepts of security and trust; want to be in complete control of who they trust; can be relied upon to check CRLs when necessary; and like to tailor their security to their own requirements. It is also more suitable where the data that is being protected is merely sensitive rather than highly confidential. Entrust/Direct, on the other hand, is more suitable for a community of users that are: novice computer users; naïve about security and trust management issues; busy doing their everyday jobs; just want everything to "work properly" with minimum intervention; and who require a highly secure transport mechanism.

It became clear that our users had several attributes/requirements:

? many were not computer experts and did not want to be asked questions relating to trust;

? most were extremely busy and would be unable to find time to manually renew their certificates (and would otherwise be denied access to live medical data); and

? most did not want to download new versions of WWW browsers or free plug-ins to obtain strong 128-bit SSL.

Additionally, from the hospital's perspective, it was important that CRLs were checked at every stage. The combination of technology-wary users and legal consequences for accidental data disclosure meant that a centrally managed security infrastructure was significantly more attractive than a user managed infrastructure.

Therefore we decided to use Entrust/Direct to provide a secure WWW connection between the GP and the hospital, and we obtained the necessary export licenses without problem.

Implementation

The following figure shows how the key components of the system (WWW browser and server, Entrust/Direct and the TTP) fit together to provide a securable architecture (it is a secure architecture only if correctly configured).

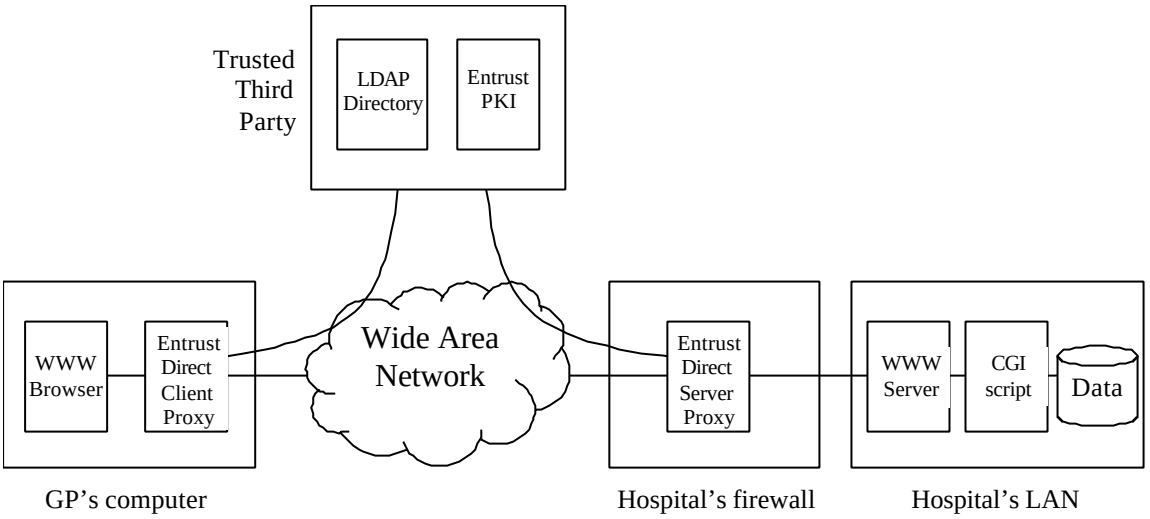

Figure 3 – components of the securable architecture

While Entrust/Direct is the key component of the application's security, the CGI script is the key component of its authorisation and functionality. The separation of the two allows a layered development and deployment process, where the application developers need not be aware of the strong authentication procedures and the security software providers need not be aware of the underlying application and its authorisation mechanism. The functionality of either component can be enhanced or replaced independently (for example, SSL instead of Entrust/Direct).

The CGI script acts as a proxy that sits on the hospital's LAN and makes requests to the database on behalf of the user. It is able to interface to a wide variety of programs, with the end-user being unaware of the details. For example, if a database application has a client-server implementation, the CGI script would simply act as a client. Alternatively, to interface with a standalone program with no external API, the CGI script could operate (albeit rather slowly) by feeding keystrokes to the program and taking a screen-scrape as the result.

The CGI script can also provide a number of useful extra facilities:

? authorisation/access control to an arbitrary granularity, even if the underlying application provides no access control.

? transaction management to give multi-user access to an underlying application with no support for such mechanisms.

? an audit trail of all modifications to the database and, under some circumstances, to make checks on data consistency. These mechanisms can be used to improve the quality of data (or to identify the source of incorrect data).

? a log of all users reading the database. As well as providing useful statistics about which database facilities are used most often, it is also able to highlight special cases of database access. For example, in the healthcare field a facility is required to over-ride access control in the event of a medical emergency. This requires a non-repudiable request to prevent abuse.

Issues regarding authorisation and access control

The CGI script acts as an authorisation proxy, and makes requests to the database on behalf of the user who has been strongly authenticated by the PKI. The way it does this depends on the access control provided by the underlying application. Some typical cases are as follows.

1. Some legacy applications exist as single-user centralised applications on isolated PCs that rely for authorisation on the control of access to their physical location (e.g. the staff database lives on a PC in the Personnel manager's locked office and, if there is any additional security, it is a single password giving unrestricted access). If a requirement existed to make this information available to a controlled set of users, or to the normal user at a remote location, it would be possible to add the isolated PC to the organisation's network without compromising security as long as it is configured correctly:

a) ensure the WWW server will only accept HTTP requests from the machine hosting the Entrust/Direct server proxy, or will only accept SSL requests;

b) ensure the CGI script terminates if not provided with a PKI authenticated identity;

c) ensure that access to the PC's TCP ports, other than those used by the secure protocol, is denied for all network users;

d) ensure that other network protocols are not installed on the PC.

These ensure that Entrust/Direct or SSL is the only means by which the PC can be contacted over the network. A simple way to achieve this is to use the Windows NT network control panel (protocols ✍ TCP-IP ✍ properties ✍ advanced ✍ enable security ✍ configure), while a cheap auditable way is to interpose a Linux firewall [IPFWADM].

In this case, the CGI script needs to know:

? which strongly authenticated names should be given access to the application;

? any password needed to access the application;

? a means of associating the authenticated name of the user with the privileges that are to be enforced on their behalf (possibly the same full access they would get if they were physically sat at the central computer, possibly a subset such as read-only access or access to staff in a certain part of the company).

2. Some applications will be centralised but multi-user, and will require separate authorisations for each user, usually through allocation of user-ids and passwords to those who can access the application. Authorisation is based on the user-id, and weak authentication is based on the password. The privileges and access control decisions will be implemented in the application. If there were a requirement to make this information available to remote users, it would be possible to add the isolated PC to the network without compromising security (with the same discussion as case 1 above). In this case, the CGI script needs either:

2a. a means of associating the strongly authenticated name of the user with their authorisation token (user-id) and weak authentication token (password); or

2b. authority to perform sufficient actions on the database as required by all users (e.g. a super-user id and password), knowledge of the access control provided by the application, replication of that functionality to

enforce user authorisations, and a means of associating the strongly authenticated name of the user to their individual authorisations.

We do not see any advantage of 2b over 2a (on the contrary), and would expect 2a to be the usual implementation design, but include 2b primarily for completeness.

3. Some applications will be implemented using a client-server model with a thin client, so the privileges and access control decisions are implemented in the server. In this case, the CGI script needs:

? a means of associating the strongly authenticated name of the user with their authorisation (user-id) and weak authentication tokens (password).

4. Some client-server applications will be implemented using a fat client, so privilege and access control decisions are implemented in the client (which will only issue allowable queries). In this case, the CGI script needs:

? sufficient privileges to access the server for all possible user queries (often complete read/write privileges);

? to know the privileges and access control decisions implemented by the fat client and to replicate that functionality; and

? a means of associating the strongly authenticated name of the user with their privileges.

In cases 2a and 3, the CGI script is accessing the application as though it were the remote user. In cases 1, 2b and 4, it is accessing the application as a generic super-user and limiting the query (or filtering the results) to what the remote user is allowed to do.

The distributed implementation requires a data store where it can securely hold the users' database authorisation and authentication tokens, plus a mapping of these to strongly authenticated PKI names (let this be stored in a structure called SEC_TABLE). In addition, for cases 1, 2b and 4, the data store needs to hold the privileges that are allocated to each user (let this be stored in a structure called PRIV_TABLE). We need a security mechanism for controlling access to these data stores.

Note that some users may have multiple privileges (user-ids) for the application (if they operate in a number of distinct roles) but may not wish to hold multiple PKI identities and key pairs. Therefore, the strongly authenticated identity of a remote user cannot be regarded as a unique primary key into SEC_TABLE.

There are several alternative locations and implementations of the data stores. The simplest is for the users to store the token information (their own entry in SEC_TABLE) in their own heads, and for the CGI script to ask them to present the tokens each time they have been strongly authenticated across the network. This is very secure, since the information is never stored on any computer (except that running the application). The disadvantage is that it is very inconvenient from the user's perspective, since the user now has to use PKI authentication tokens at session start up and repeatedly use application tokens as different CGI scripts are invoked. Consequently we discount this as a realistic mechanism, since it could annoy users by overburdening them with security hurdles. However, it is still useful in combination with other storage mechanisms.

The CGI scripts could also maintain the data store on their local server. This needs to be initialised with each user's authorisation and authentication tokens, so we ask the user to present their tokens (user id and password) at session start up. If the data store is persistent, this need only be done when the first time the user uses the application. Otherwise, it must be done every time the user starts a session.

There are several places where the data store can be located:

1. in a file (maybe hidden) on the WWW server's disk
2. in the Windows NT Registry
3. in a hardware token attached to the WWW server machine
4. in a database table separate to the server application

5. inside the server application (if the application is a database that permits structural modifications - this is useful in case 4, where the database may already contain tables that allow the fat client to manage users and privileges)

The information in SEC_TABLE is sensitive, and must be protected from disclosure. In the first two cases, the information should be encrypted, and the CGI script must know the decryption key. In the other cases, a password or PIN number is required by the CGI script to access the information. We have redefined the problem from “protecting the data” to “protecting the means for access to the data”.

The stateless nature of the World Wide Web provides a number of difficulties with secret/password management. As every incoming query results in a new instance of the CGI script being spawned, each interaction with the central application requires a separate bind-query-result-unbind sequence (or an instantiate-query-result-close sequence for a centralised application). As well as being inefficient, each bind or instantiate operation requires access to SEC_TABLE to retrieve the required user passwords, and the CGI script must have access to this table. As human intervention is impossible, SEC_TABLE must be unprotected or the CGI script must know the credentials (symmetric key, password) required for access. These can be hidden (in the executable file, the Windows NT registry or hardware tokens), but encryption is no solution as the CGI script would need to access the decryption key and the problem is merely deferred. This approach provides obscurity rather than security.

These problems can be solved if a Session Manager is introduced to the architecture. This is a permanently running service that sits between the CGI script and the application and performs the following services:

? It provides efficient management of binding and unbinding to the application. A fresh bind will be initiated the first time a user accesses the system, and this will stay in place until the user explicitly unbinds or a timeout occurs (the PKI authenticated identity is used to associate a particular user with a particular session). This reduces overheads on multiple queries.

? It allows access to SEC_TABLE to be controlled, as a human operator can provide a password (or something that can be converted to a symmetric key) when the service starts.

The CGI script passes the http query and the authenticated identity to the Session Manager, which converts the http query into an application specific query, binds to the application if required and initiates the query. The CGI script must be trusted to relay PKI authenticated identities correctly, and the WWW server must be managed to ensure this.

Implementation issues

Our healthcare application uses a client-server architecture with a fat client. As described in case 4 above, the Session Manager must know a password to access the database (as used by the fat client) and the access control mechanisms governing client access. The secure data store must contain knowledge of the users' security tokens mapped to authenticated identities (SEC_TABLE) and knowledge about which privileges are associated with each authorised identity (PRIV_TABLE). The latter already exists in a database table holding privileges associated with each user-id for the fat client. The Session Manager maps the authenticated identity to a database user-id using SEC_TABLE, and uses PRIV_TABLE to determine the privileges corresponding to that user-id. Therefore, a user has the same privileges for remote and local access, as PRIV_TABLE is used by both to construct well-formed queries and filter results. The user has potentially full access to the database, because the Session Manager has full privileges and the user is limited only by the restrictions in PRIV_TABLE (full access is required for production of global anonymised statistics and audit information).

For construction of SEC_TABLE, mechanisms for distribution of certificates, database user-ids and passwords must be defined. We found the potentially complex structure of the LDAP distinguished name (DN) [RFC1799] meant entry of these into SEC_TABLE was error-prone. The hospital preferred not to worry about DNs but to work with familiar database user-ids and use the same user-ids and passwords for remote access and their fat client (this precludes the obvious enhancement of using the certificate DN as the database user-id).

The credentials are distributed as follows. The Trusted Third Party (TTP) uses its procedures to issue certificates to participating doctors, using the NHS Directory schema [NHS1] for DN construction. The hospital creates user-ids and passwords for each doctor, stores them in SEC_TABLE and sends the user-id and password directly to the doctor. The hospital assigns privileges and stores these in PRIV_TABLE with the database user-id as primary key. The first time a doctor accesses the system, the Session Manager cannot find an entry containing their DN in SEC_TABLE, and asks them to provide a valid user-id and password. If the user-id and password match a pair stored in SEC_TABLE, the Session Manager will store the user's DN alongside the database user-id, and will use the user-id to control access to the database.

On further use by the same doctor, the Session Manager uses SEC_TABLE to look up the user's DN, and finds it stored alongside a database user-id. The Session Manager uses that user-id to look up the user's privileges in PRIV_TABLE. The user is only required to authenticate themselves to Entrust/Direct, and that strongly authenticated identity determines their privileges.

In this arrangement, it is important to separate strong authentication and authorisation. If the hospital distributes user-ids (authorisations) and passwords (weak authentication) to doctors, the TTP can certify users (strong authentication) without having the possibility of access to the database. This allows a TTP external to the NHS to operate as effectively as one that was internal. If the TTP distributes both certificates and database credentials, it has opportunities for data access that it should not have. Separation of duties and roles is always good security practice.

Finally, it should be noted that the Session Manager and the secure server components both require the trade-off between security and availability to be defined. This depends on the application and the nature of the data being protected. In both cases, a shared secret needs to be provided to give access to server's private keys or to SEC_TABLE. If the strictest security is required, human users must provide this, and system availability is compromised if someone is unavailable. If availability is paramount then the processes may be self-restarting, in which case the shared secret must be accessible and could become vulnerable to reverse-engineering.

Conclusions

In this paper we have shown how an existing healthcare application, a diabetic database, can be made available over the Internet, even if it is not network-aware and does not consider the inherent untrustworthiness of public networks. This system has been implemented as part of the EC 4th Framework "TrustHealth2" project and is being used as the underlying data distribution mechanism for a Clinical Decision Support System being implemented as part of the EPSRC "Distributed Diabetic Dietician" project.

We have considered how alternative technologies such as Entrust/Direct and SSL can be used to implement the strong authentication component of such a system, and have looked at how authorisation can be provided to a variety of different types of end user applications. Where there is a need to manage both strong authentication credentials and simple authentication passwords, we have examined how these may be handled from a user perspective as well as how the users will be managed.

We believe the issues described here are generic and are applicable across a wide range of applications, and that the legal constraints on disclosure of patient-specific information in the healthcare sector are similar to the commercial constraints on disclosure of company confidential information in the commercial sector. We therefore believe that this architecture has wide applicability.

References

[Chadwick] Chadwick, D., Tassabehji, R., Young, A.J. “Experiences of Using a Public Key Infrastructure for the Preparation of Examination Papers”, Computers in Education, To be published 2000

[NHS1] National Health Service Standard NHS 0001, “Directory Services”, 1997.

[NHS2] “Information for health: an information strategy for the modern NHS 1998-2005” NHS Executive, 1998

[PKCS7] RSA Laboratories, "PKCS #7: RSA Cryptographic Message Syntax Standard," version 1.5, November 1993.

[RFC1779] Kille, S. “A String Representation of Distinguished Names”, March 1995

[RFC2246] Dierks, T., Allen, C. “The TLS Protocol Version 1.0”. January 1999

[SSL3] Frier, A., Karlton, P., Kocher, P. "The SSL 3.0 Protocol", Netscape Communications Corp., Nov 18, 1996

[Whitten] Whitten, A., Tygar, J.D. “Why Johnny Can’t Encrypt. A Usability Evaluation of PGP5.0” Proceedings of the Eighth USENIX Security Symposium, August 23-26, 1999, pp. 169-184

[X509] ITU-T Recommendation X.509: "The Directory - Authentication Framework". 1997

URLs

[Entrust] see www.entrust.com

[Fortify] see www.fortify.net

[IPFWADM] see www.dreamwvr.com/ipfwadm/ipfwadm-faq.html

Acknowledgements

This research was funded by the European Commission IV Framework Programme Trusthealth 2 Project (Contract No: HC 4023) and the UK EPSRC (grant number GR/L60548).The authors would also like to thank Entrust Technologies for making their security software available to the research on preferential terms.