The Jericho Challenge

Table of contents

1 Introduction

Moving to a Jericho World requires the ”provision of a low cost secure wide area network connectivity supporting a variety of geographically dispersed business units, with facilities of different kinds.”

And “de-perimeterisation” involves re-appraising where security controls are positioned, re-balancing cost and complexity. This may involve moving security controls from firewalls or proxies to internal end systems or applications”

By way on an introduction it is worth looking at the pros and cons of network layer security vs. application layer security and at the open standards available such as IPSec, SSL and SSH.

We will also look at the issues of how to approach delivering the Jericho vision in a practical way, which enables rapid partial adoption, combined with a defined route forward which leads to a way of achieving more complete adoption.

To achieve anything at all in the short term we need to look at the elements involved in the move to a more transactional relationship between end-point (users) and the servers (services). In the simplest model there are just 5 of these elements:

- The user (use cases, identity)
- The end-point (security, client software)
- The communication channel (security)
- The server (security)
- The application (software)

The communication channel is arguably more available and adopted than the other elements. So we need to explore the issues surrounding the two ends of the communication channel in order to develop a workable approach.

2 Moving away from the Firewall-centric view

Traditionally, firewalls have been used to secure networks by building a supposedly secure wall around the network. However, users on the inside need access to outside resources on the Internet and many users and services on the outside need access to internal resources, for example delivery of e-mail, IP telephony (VoIP) and remote users who need access to various internal applications. In addition, the use of wireless LANs extends networks beyond the physical network boundaries. All this together makes it very hard to maintain the firewall-centric view where security to all devices is controlled through a number of firewalls combined with one or more VPN systems.

Instead, a new architecture where each device is capable to protect itself is needed. Centrally defined policies should then be distributed that govern how each device on the network should behave. If the policy system not only include corporate servers but also end-user workstations, remote user’s workstations and network encryption, a much higher degree of security can be achieved. It also facilitates when offering secure remote access to various internal services to remote users such as business partners, home workers, mobile users, etc.

Firewalls can still be present but will in the long run be transformed into becoming central systems for data collection for intrusion detection systems (IDS) and intrusion prevention systems (IPS).

2.1 The view from the Server/Application end

There will be two different types of systems in the network: older legacy systems that cannot be modified but still have to function in the new environment; and new Jericho-aware systems that can be designed for the new infrastructure in the network.

Legacy Systems

The first point we have to recognise is that there are and will continue to be, numerous existing applications and legacy systems. These have 3 main problems: the applications were never written to work in a secure way; they may run on old operating systems which do not support security features; and the combination of application/OS/hardware may not be hardened to withstand exposure to external threats.

Today, new designs could potentially clear a sufficiently high security hurdle so that they could be thought of as “Jericho ready systems”, but these will have to co-exist with legacy systems for many years to come.

So, if there is no external firewall any more, then there is going to have to be some other form of supporting device (hardware and/or software) that can provide the security features (authentication, firewalling, access control, encryption, etc) which the legacy systems lack. The use of such devices could take two forms: distributed - an add-on to each and every legacy server or aggregated – a gateway that sits between the legacy servers and the great unknown. Whatever forms of device are used to add the missing security features, enterprises will be sure to demand some sort of central control. Security can only be managed effectively when there are few well-defined points of control that defines and distributes policies automatically.

Jericho-Ready Systems

Even in a world full of Jericho ready systems, the argument for a single point of control remains strong. As we have already alluded to, systems have to be written to work in a secure way, they must have built in security features and they must be able to withstand exposure to external threats. When an enterprise has many (distributed) Jericho ready systems, it is not going to be easy to manage aspects such as: who has access to which systems; security settings like which crypto is being used; ongoing maintenance like the latest patches; etc. This management overhead would be too much and would inhibit the implementation of a strong controlled security policy across the enterprise. So even in the Jericho world there is going to have to be a new generation of ‘Control Server’ to provide this single point of control.

So the view from the Server/Application end would suggest the need for a control device irrespective or whether we are looking at Legacy or Jericho-ready systems. Some of the functionality required will be the same control mechanisms but the Legacy systems will also need some extra security related functionality. So there is logic in suggesting that one control device should be able to do both jobs and in so doing would provide central control over both types of systems.

2.2 The view from the User end

The scope of the Jericho challenge is pretty wide. From a user viewpoint business scenarios like *Phoning home from a hostile environment* and *Application access by suppliers, distribution agents or business partners* might appear to be the same in a Jericho world, but actually encompass some different challenges.

The “One to One” transaction

By looking at, for example, *application access by suppliers, distribution agents or business partners,* this might represent a typical “one user to one system” transaction, for example an agent that just wants access to the ordering system to place an order. The technology to allow this exists today in the form of an HTTPS based connection to a web server at a known URL. In a completely Jericho-aware world there may be some improvements that need to be made, but there is no denying that this is a good model to start with.

The “One to Many” transaction

A user who wants to phone home from a hostile environment is a typical “one user to many systems” transaction. Clearly, this is an area where much more work needs to be done to address the needs of a Jericho world. There are some immediate things to think about like the word ‘home’. A ‘home’ is ‘someone's or something's place of origin, or the place where a person feels he/she belongs’. The very act of knocking down the walls (firewalls) has made it much harder to define what home is. In a Jericho world what was ‘home’ may now be distributed and the Internet can be part of used to be a corporate network. For example, why should the mail server have to be in the same physical location as the CRM system if they no longer need the protection of a firewall? Instead, full flexibility for placement of both clients and servers is possible. So Phoning home from a hostile environment may now mean establishing connections to a number of geographically disbursed servers (services) at the same time. This immediately brings in a number of additional issues that need to be addressed:

- User authentication over insecure networks including distributed authentication systems through “trust domains”
- User authorization based on many different parameters such as user’s role, client system being used (type, physical location, level of security), date and time of day, etc.
- Traffic integrity (protection against spoofing and message modification) and confidentiality
- Server protection (firewalling) and means to receive user rights (roles) from central authorization servers

From a Jericho perspective it would be nice to define an architecture where these issues are addressed in a distributed way; where Jericho systems all working together and requiring no coordination. However, the ever present problem of the user is more difficult to deal with. For example, many users have no desire (or ability often) to remember the details of the different systems they want to access. So if the user has a choice of services then maybe some sort of user portal is required. For this reason alone, it is desirable to have a primary point of interface (PPOI) that understands what a user wants to do and then automatically deals with all the technical details without user intervention. And having made that leap, it then seems quite logical that that many of the issues are best handled through the PPOI.

The PPOI should from a users point of view do two things: Firstly it should be the door to the virtual transactional environment they seek access to; then after having got through the door, it should let the user know exactly what systems are available under the current circumstances.

3 System requirements and scenarios

3.1 Overview

Enterprises moving to the Jericho world need to change their thinking away from the 'edge' mentality based on controlled denial of access through firewalls, which was forced on them when the Internet became the de-facto means of connectivity. Instead they need to adopt 'core' mentality based on controlled access to Jericho ready systems.

From the server/application end we have already identified two elements:

- the need for devices to control/protect/secure access to legacy systems
- the need for central control over Jericho ready systems

And from the user end we have identified two elements that might form part of the PPOI:

- identity/authentication/transaction selection
- system availability/advertising

Some and maybe most of these requirements can be delivered today but not in a form which is immediately suited to the Jericho world. So inventing lots of new requirements is probably not the most effective way forward. Therefore we will start with taking as many elements as possible that exist today and reuse them in this new world. So what follows is a proposal for an architecture based around gateways that will be able to offer the desired service to both users and system owners.

3.2 A typical scenario

Before looking at each component, let's begin with investigating the process we are trying to support. Assume we have a user who needs to connect to one or more services (servers) securely. Then the system must be able to perform the following steps:

1. User finds a machine and connects to a PPOI (with a name that is that is easy to remember)
2. User identification/authentication; the user should only have to log in once to the system ('single sign-on')
3. System checks availability of possible services (authorisation) for the user. Availability may depend on many different parameters.
4. System provides user with information about service availability and the user selects what to do (this step should be optional and all access could be completely transparent for some users)
5. System provides application servers with information about the user's identity and current access rights ('role')
6. Application servers will allow authorised users access to requested services while blocking all kinds of service to unauthorised users. Traffic is optionally encrypted to provide message integrity and/or confidentiality

1. Finding the PPOI

The user should have a simple interface to a portal that becomes the PPOI. It could be accessed either through directing a web browser to a selected server or by starting a pre-installed client on the workstation that connects to the PPOI.

Users should be allowed to connect to multiple PPOI, for example if different organizations are involved. It can be one provided by their employer and another by a business partner that is required during normal corporate work. Preferably, both PPOI should be aware of each other's existence and be able to grant or deny particular services based on what the user currently are accessing.

2. User authentication

Ideally, users should only have to be authenticated once, i.e. when the user's identity is established it should be possible to use that in many systems. In reality, users may have to be re-authenticated by other servers only when there is no trust between two domains (a typical example is if a user needs to purchase a ticket over the Internet where the remote server uses its own authentication system). The system should, however, support creation of trust domains where it is sufficient to identify the user in one place and to base access to other systems based on that identification.

It is also important that authentication is separated from authorisation, i.e. that each system should be able to determine a user's right when the user accesses a particular service. This also implies that identification is a centrally administered service whereas authorisation can be delegated to individual service providers. One such system which is currently available and is conforming to open standards is Kerberos (RFC-1510). It allows creation of administrative realms and separates authentication from authorisation in very much this way.

The authentication system should provide support for many different authentication mechanisms such as username/passwords, LDAP passwords, token cards through Radius and certificates/smart cards following the X.509 v3 standard. By offloading application servers the task to authenticate users, it is much easier to implement support for many concurrent authentication systems.

3. Service availability check (authorization)

Authorisation should be based not only on user identity but also many other parameters such as time of day, user's location, the configuration of the workstation (existence of personal firewall, anti-virus protection, etc.) and what other services the user currently has requested access to should be accounted for. It must be possible to define a policy such as access to a corporate file server is granted using a password only during working hours and when connecting from the corporate network. In all other cases access should require authentication through a smart card a personal firewall to be installed on the client.

In addition, users should be allowed to take different roles. It should be supported by the system that roles are exclusive, i.e. that a particular role (e.g. accessing the patent database) requires that no other roles (services) are accessed at the same time.

4. User interaction – role and service selection (optional)

Since roles can vary over time, it is essential that users always realize what services they currently have access to. If not, many helpdesk calls will just be to help explaining to users that a particular service is not available, for example because of the selected authentication method. If the users can have a portal-like user interface displaying available services, it becomes more obvious that a particular service is not available at this point in time.

Adding the possibility for users to enable access to particular applications when needed will help to enforce the “principle of least privilege”. In most environments it is a great advantage if systems are not available for traffic unless the user actually have requested access to it. If an attacker is able to spoof traffic from a user to a server, only ‘opened’ services will reach the application server.

This feature with user interaction should be optional. Not all users want to see exactly what is available or the number of services may always be static. If desired, system owners should be able to specify that some services are always available once the user is authenticated (and of course authorized to use it).

5. Provide information to application servers about user identity and access rights

Since user authentication and authorization is done by the central system, application servers should be able to give access to all users who can provide them with a (signed) certificate that shows that they should have access to a service. There is no need for the application server to check the users identity, nor to ask a central server for user authorization. Everything should be self-contained in the certificate including rules for communication such as requirement for encrypted access.

This kind of service is already provided for in the Kerberos system (although the ‘certificate’ is called a ‘ticket’ that is created by a central authority. Servers never need to check ‘tickets’ against any other server; everything needed for user access is contained in the ticket itself.

6. Application servers – providing access for authorized users

Authorised users can now connect to remote servers. Encryption should be possible to demand on a global scheme, i.e. if a server requests it, the clients (and users) must use encrypted communication to be able to use the service. In addition, global policies should demand client configurations for some (or all) applications, i.e. a policy may state that a user must be authenticated with a smart card or token device, use encrypted communication and have a particular firewall rule-set installed. If not, that service will not be available.

4 Architecture

In this section we will describe how a Jericho system can be built to conform to the requirements outlined in the previous section. The implementation details need to be refined much more in more detailed design documents, but the overall goal and architecture should be clear from this description.

4.1 Existing components and related work

The thoughts of having a distributed system that seamlessly can offer services to users is not new. One such project is the Distributed Computing Environment (DCE) by the Open group (formerly called Open Software Foundation, OSF) [1]. It is an open source project and one major contribution is the creation of a distributed file system, DFS. The development of the platform seems to have come to an end but many products and systems in use today are based on DCE. We believe that many thoughts can be picked up from this project when defining our architecture.

Another very useful, and open, component in this environment is Kerberos developed by MIT [2]. Kerberos allows systems to separate authentication from authorization as described in the previous paragraph. It also allows application servers to check user authorisation for a service just by inspecting a ‘service ticket’ the client hands over to the server. The server does not need to connect to a central service to verify the correctness of these tickets, thus it offloads application servers from all authentication and authorisation work and allows for central administration of users and user rights. The Kerberos protocol is open and also supported by many operating systems, it is even used internally in Windows 2000 and later. Kerberos uses strong cryptography and is designed to be functional in a hostile world such as on the Internet.

User credentials should preferably be stored in a central repository. One standard in this field is the use of LDAP (Lightweight Directory Protocol) protocol originally designed at the University of Michigan but now standardised by IETF [3]. LDAP is an open protocol and many products support it (Windows Active Directory can also function as an LDAP server). There is also an open implementation of and LDAP server database available which is used extensively [4]. The solution outlined here will be based on the use of LDAP but in a real working environment, other authentication systems such as Radius and X.509.v3 certificates must be supported as well.

For encryption of network traffic, we will propose the use of three standard protocols:

- SSL for web traffic encryption (de-facto standard) and newly developed Jericho-aware services.
- SSH for non-Jericho aware applications (such as legacy systems)
- IPsec for server to server encryption

4.2 Client and server protection

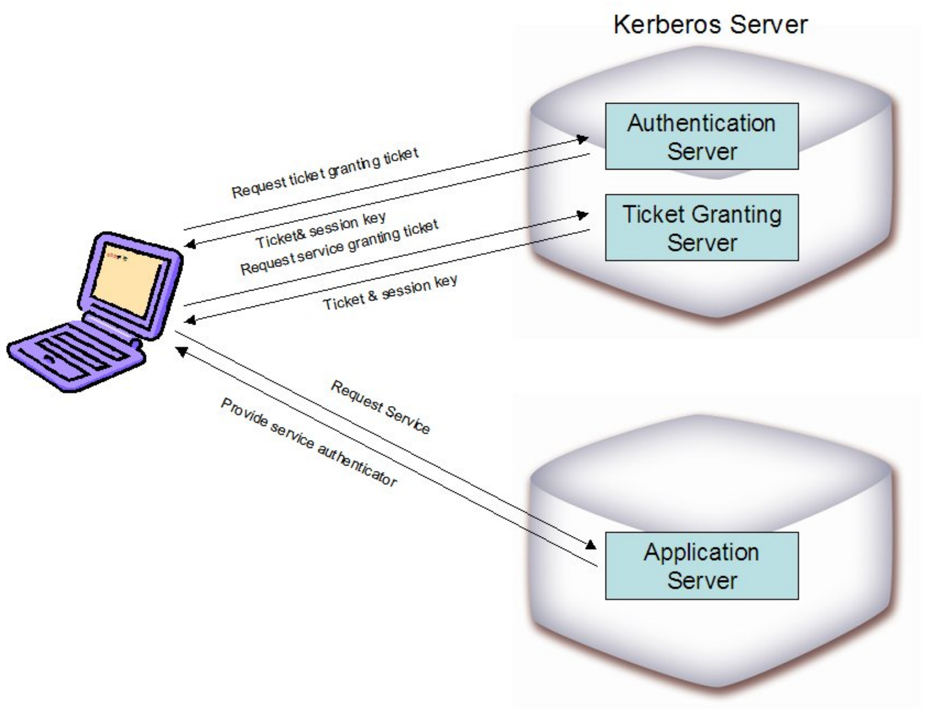
When tearing down the external firewall system in the Jericho world, protection must be moved to each component in the system, i.e. to servers and clients/workstations and to mobile devices. This means that instead of having one central firewall, a number of distributed (personal) firewalls will be introduced that protect each asset on the network. Qualified firewall administrators should manage the distributed firewall system from a central point in the network. Administration of personal firewalls, for example on user workstations, should never be handed over to individual users since they now are part of the corporate protection system.

Assuming each object can protect itself; the overall security level achieved in this system can be significantly higher than before. A major reason for this is that all systems are now protected against hostile traffic regardless of its origin. In the central firewall world, only external attacks were blocked by the firewall and all internal traffic was passed to servers. This is important to consider, especially in a larger corporation where thousands of users is given access to the network and application servers are virtually unprotected (the typical scenario is that a new employee gets full network access the moment he/she is employed.)

The overall assumption we make here when designing the system is that, by default, *no traffic to systems (clients and servers) is allowed except from authorised users*. The main problem to be discussed in the following sections is how to make sure that only authorised users can connect to the systems and what traffic should be considered authorised.

4.3 The Kerberos Authentication System

This description of the Kerberos system is intentionally kept short to avoid the details but still being able to highlight the benefits from using the Kerberos system as the core of the authentication/authorisation system.



Using Kerberos for Authentication and Authorisation

The Kerberos server separates authentication from authorisation and makes sure application servers do not have to participate in this process. When a user wants to be authenticated:

- The client contacts the Kerberos server to be authenticated. The Kerberos server can contact external servers for user authentication.
- The client receives a ticket from the Kerberos server that shows that the user is authenticated. The ticket is encrypted by the Kerberos server and is only useful in contact with the Kerberos server.

Now, when the user wants to access a specific service, the following scenario occurs:

- The user contacts the Kerberos server (it sends the ticket showing that he/she is authenticated).
- The Kerberos server makes an authorisation decision and if access to the service is granted, the user gets a ‘service granting ticket’ (SGT).

This scenario can be repeated for each new service the user wants to access. Note that authentication is only done once, each request to the Kerberos server is now about authorisation for a new service.

When the user wants to access the service, the following occurs:

- The SGT is sent to the application server at the first access. It contains encrypted information for example about who the user is and what encryption key should be used, if applicable.
- The application server does not have to contact any external system for user authentication or authorisation. This has already taken place and all necessary information was included in the ticket and was encrypted by the Kerberos server to the application server. The user cannot read or understand its contents.
- If the Kerberos server has submitted a key, the communication should be encrypted between the client and the server.

This scheme should be the basic foundation in the Jericho system. It offloads application servers from all work and interaction with external servers as well as determining the role each individual user should have, and it makes it possible to create central policies for who should be able to access what services.

4.4 Authorisation

Authorisation is inexorably tied in with identity and company policy.

As with Kerberos there is a ‘standard’ that is widely adopted by companies who supply authentication solutions; this is RADIUS. It would therefore seem appropriate that our Kerberos server should support this type of interface.

Level of Authentication

Any company hosting Jericho services will almost certainly have a policy in place with regard to authentication of the user. This will fall into one of 3 categories

- One factor
  Something you know (password)
- Two factor
  Something you know and something you have (token)
- Three factor
  Something know, something you have and something you are (biometric)

Decisions about what level of authentication is used should probably be left to an individual company to decide.

However our solution should be able to handle different levels of authentication at the same time. This would enable someone with only one factor (maybe an external company) to be granted limited access to services; and someone else with two factor (maybe an employee) access to several services.

Common Identity

In a Jericho world someone may have access through several different PPOIs. If each one has different authentication requirements then the user may need to remember lots of passwords and have a key chain covered in tokens. In a Jericho world there is therefore merit in trying to identify forms of user authentication which would work across different PPOIs.

Passwords can be made to work this way but if one PPOI requires a change every 2 weeks and another every 3 weeks then the solution gets out of control quite quickly. The ideal solution might be based on 2 factor authentication, where the 'something you know' can be fixed or within the domain of the user only and the 'something you have' is portable an can be recognised/used by any PPOI.

Again there are going to be a number of ways of doing this and different solutions will suit different users. A good summary of some of the solution alternatives can be found at http://securitypronews.com/securitypronews-24-20040115StrongAuthenticationAlternativesReportforCustomerX.html. However two examples that offer one means of connecting to multiple PPOIs are:

- Certificate
- OTP (one time passcode) via SIM Card

Certificate

Digital Certificates have to be installed on the device being used at the user end; this would restrict their freedom to access PPOIs from anywhere. The Certificate store may additionally be password protected on the users device. Whilst this is a pretty robust way of preventing third parties authenticating to a PPOI – it can only offer password protection if the users device is stolen or is available for others to use.

OTP via SIM Card

The SIM card is something one has (in ones phone). And clearly the user must have network coverage at the time of authentication. The user logs in say using a Radius compatible service, entering their username and password. A one-time passcode is sent by by SMS to the phone. Costs are kept low and you get genuine two factor authentication. A good example of such a system is can be found at http://www.mideye.com/

4.5 Access and Traffic Encryption

If decided by the central authorisation system that traffic should be encrypted, servers must enforce that this is done. A client refusing to encrypt traffic must be disconnected. Now, what system should be used for encryption? We basically have three candidates, all open standards: SSL/TLS [5], SSH (Secure Shell) [6] and IPsec [7]. The main difference being that IPsec works on the network level and must be either pre-installed in the system (as a system-level driver) or built-in to the system. SSL/TLS main advantage is that some applications such as web browsers and e-mail clients already support it. Finally, SSH is easy to use when securing existing applications. It supports tunnelling of traffic and can in many environments be completely transparent for the applications being secured.

SSL-based services

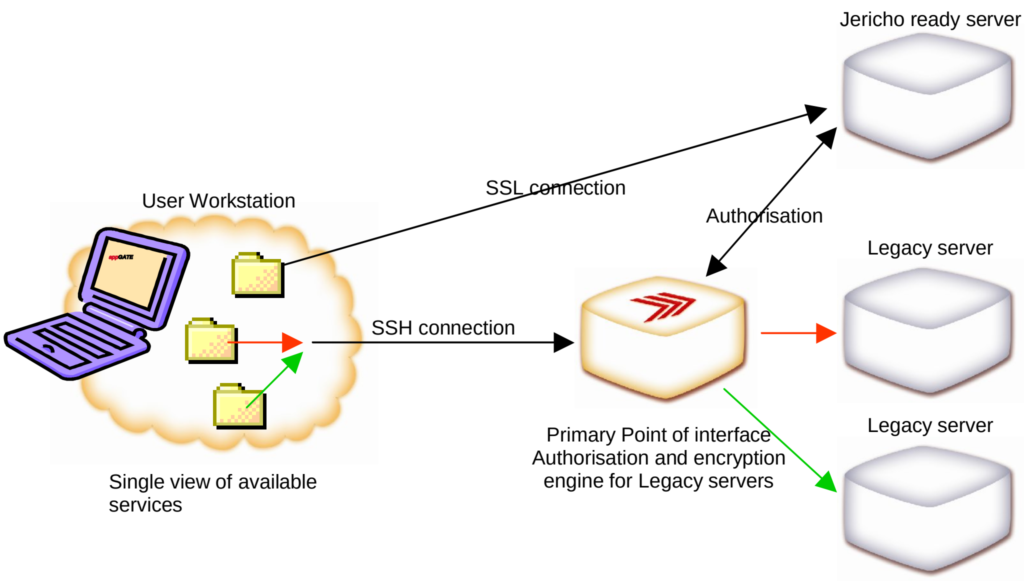
SSL[1] is built-in to many applications today and will in the Jericho world be the preferred protocol to use when new applications are designed. SSL is easy to implement when writing new applications and it offers a high level of security. It should therefore be the preferred protocol when one or a few TCP connections between a client and server should be encrypted. This is also the reason why all web-browsers today support SSL and a major argument for why the Jericho system must support SSL-enabled services.

When built-in to applications, no additional software is needed to perform encryption; we will have a true zero-footprint client. However, currently few SSL implementations support Kerberos for authentication and authorisation and therefore, the server must be able to handle this separately. However, this is not a problem for new applications that should be adopted to the Jericho environment, since the combination of SSL and Kerberos is described in an RFC standard [8].

Transparent encryption of traffic

Other applications that are not Jericho-aware must be forced into using encryption if necessary. This is possible to do in different ways, but the best and simplest method is to use the SSH protocol as discussed above. SSH supports tunnelling of arbitrary (TCP) connections (including multiplexing multiple connections into one tunnel) between the client system and a server. It can invisibly intercept traffic from “unknowing” applications, encrypting it and sending it to a remote server[2].

Many SSH products today also support Kerberos as authentication method, such as the Solaris 10 server.



Using SSL for web-based services and SSH for all other services

Since most people are more familiar with the name SSL than TLS, we will use it in this text. TLS is SSL version 3.1 which is the preferred version to use.
[2] Another method done in some VPN systems is to use the SSL/TLS protocol. However, multiplexing of multiple connections is not automatically done but would require special vendor-specific (proprietary) software solutions to this problem.

Server to Server traffic

In all environments, there will be servers that need to communicate sending non-user related traffic. Therefore, servers should be allowed to automatically set up connections between them (after authorisation by the central rule system) and the setup process should be automatic without any user intervention. There are two choices of protocol we can do here, either deal with this communication in a way similar to user initiated traffic, or more easily, to allow these servers communicate using the IPsec protocol. We propose the first method using IPsec in this environment mainly because it is completely transparent for application-level protocols and it is already supported by most server operating systems.

The use of certificates makes it possible to secure identify communicating parties without involving a normal user authentication procedure. Each server must also have a (personal) firewall in place that prevents unauthorised traffic to be sent to it. The firewall policy should be distributed globally when services are defined. We will come back to this topic later in the paper.

UDP and other protocols

Protocols based on UPD (and ICMP) cannot be tunnelled using SSL or SSH without some additional help. In situations where support for all IP protocols is required, a small piece of software (a virtual Ethernet driver) can be downloaded to the client that intercepts all IP traffic to a remote network. Currently some VPN vendors (such as AppGate) support this solution and it requires a vendor-specific solution. Nothing however prevents us in this forum to create a standard for how this should be done.

Temporary systems, mobile and hand-held devices

Mobile devices and temporary systems must be supported in this environment. It must be possible to use an arbitrary client to connect to at least some important applications. The system should be flexible enough to allow systems administrators/owners to decide what should be possible to access and technological limitations should be kept to a minimum.

Web based services (http and https) should be supported out of the box using a built-in web client together with proper user authentication. Other applications can be securely used if a small Java applet is downloaded to the client that supports SSH tunnelling. This applet then takes care of user authentication and authorisation and can be a full Jericho-enabled client. Java is available for almost all desktop systems and many Java implementations can be used in PDAs, telephones and other hand-held devices.

Client protection in these systems may not be entirely sufficient. As a corporation, we might not want to rely on the personal firewall administration on a client found on and airport or an Internet café. Therefore, the authorisation system must be aware of the client's configuration (or lack thereof) when determining access rights to various resources.

4.6 Application Servers and Services

Jericho Ready Systems

Applications (or services[3]) that are Jericho enabled should be adopted to use Kerberos tickets for authorisation to the service and should be prepared to encrypt traffic with SSL if required by the central security policy (i.e. according to what it finds in the Service-granting Ticket). The number of Jericho-aware applications will increase over time, but since we've selected Kerberos and SSL as

[3] Note that we try to distinguish between the physical 'server' and 'service(s)' running on the application server.

the major components, several applications would already today support this environment and more will definitely come.

Jericho-enabled Servers with older applications

Applications running on Jericho-enabled servers, i.e. applications that are not or cannot be made aware of the Jericho system, require a front-end to be present to handle interaction with the Jericho system. This front-end is a small piece of software that takes care of data encryption and makes sure the users are authorised to connect to the application service. The purpose of the front-end is to make non-Jericho aware applications become Jericho-compliant without having to make any modifications to them.

The Jericho-enabled servers also need a centrally managed (personal) firewall for protection.

Non-Jericho-enabled servers

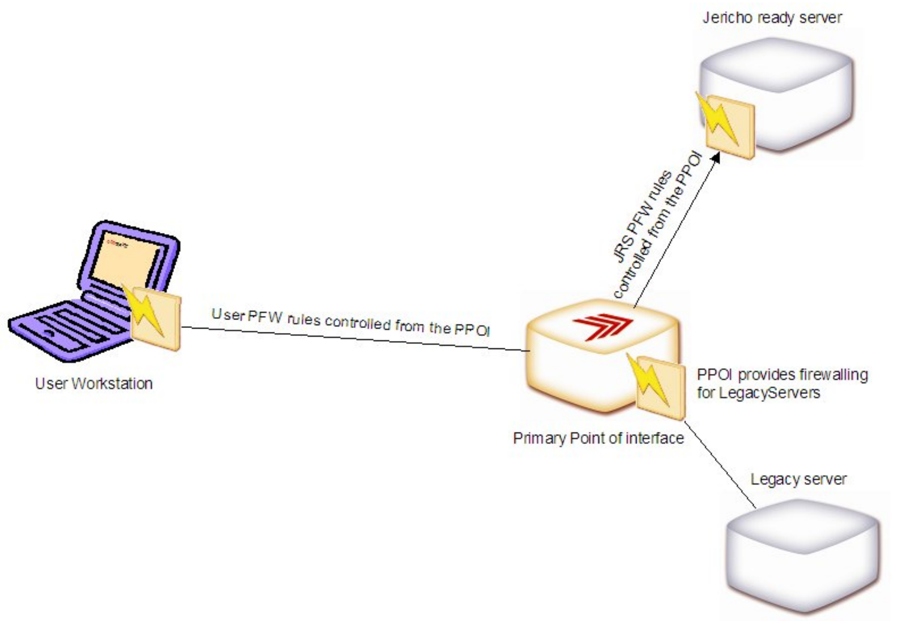
There will always be servers that do not support or understand the Jericho environment and which cannot have an additional Jericho module (i.e. the front-end as discussed above) installed. It may be to do with technical issues (such as unsupported platforms) or reasons relating to policies or server ownership. Either way, support for such servers must be an integral part of the system solution. The simplest solution is to protect the application server with a small box in front of it that acts as a firewall, encrypts traffic from users and only lets authorised traffic through to applications running on the server. This ensures that only authorised users can connect to the server and see what applications it offers to the network.

4.7 Firewall and network protection

Each application server should be equipped with a firewall, either an internal firewall or an external appliance that protects this specific server. Most systems should do with using a good personal firewall (PFW) that is centrally administered.

The PFW rules should make sure that access is limited regarding who can see or communicate with servers. Only connections that can show a valid ticket should be allowed to continue. The firewall filters can even be extended in the future in such a way that a central policy manager knows exactly what systems are allowed and expected to connect and therefore distributes this information to the PFWs. This way, any non-authorised (or non-expected) clients cannot even detect the presence of the server since its traffic is dropped before it is even opened by the receiving system.

Workstations in the network should also be given the opportunity to be protected by PFWs. Again; the policy should be controlled by the central authority that governs access to applications in the network. Administration of personal firewalls should never be handed over to individual users since they now are part of the distributed protection system around a truly ‘distributed’ organisation.


Jericho ready server
JRS PFW rules controlled from the PPOI
User PFW rules controlled from the PPOI
User Workstation
PPOI provides firewalling for LegacyServers
Primary Point of interface
Legacy server

Firewall protection of clients and servers

5 Conclusions

The proposed solution is rather simple, at least from a conceptual point of view. The central firewall complex is replaced by a set of distributed firewalls that are placed on all clients and servers. These firewalls are centrally controlled and can dynamically be configured to allow or deny traffic in the network. They also make sure that applications and application servers become invisible to unauthorised users! This is an important first step and will also increase internal security tremendously. The distributed firewall concept virtually introduces the same protection as we today are used from Internet users to internal users as well.

We have also introduced a distributed authorisation system based on Kerberos. A central Kerberos server will take care of all authentication and authorisation requests. Application servers will not have to bother with these issues but only to deliver service to its users. The system is also able to specify whether encryption for confidentiality and/or integrity should be demanded before granting access.

Users using the system will have clients with a GUI that shows them what services are available at all time. This is important since the security system can be configured to grant access based on many different parameters such as authentication method being used, time of day, system being used, presence of a personal firewall, etc., and therefore users must be able to see what resources they currently can and are allowed to access.

The system is also designed in such a way that everything from web-based (SSL) applications to all kinds of legacy applications can be used. Clients must be supported on a large number of devices from desktop systems to small hand-held devices and telephones. Therefore a generic client in Java is proposed (this concept is already tested and in extensive use by AppGate customers).

The proposed system is able to handle not only newly designed applications but also to be able to integrate older and legacy applications into the architecture. This is important before a larger deployment is possible.

Security and Administration

Client and server security do not depend on link-level technology or on individual users and their actions. Everything is centrally managed and administration can be divided into different domains where trust (to a certain degree) can be established between domains. Identity management can also be split into different levels of administration.

The system supports extensive logging and the use of distributed firewalls and a distributed authorisation system also makes it possible, and necessary, to globally collect data from all equipment. Since protection is deployed in the end-points, very close to the applications, it is possible to have a very detailed knowledge of events and the centrally collected logs can be very accurate.

The use of distributed components as opposed of using large central firewalls and devices, makes this system much easier to evaluate and possible to certify according to different standards such as common criteria (CC). The OpenSSL libraries that are proposed here (can be used both in SSL and SSH) are currently undergoing FIPS certification and therefore the resulting system should be very easy to certify as well.

Traditional VPN systems will not needed in this architecture since secure remote access to devices is already in place. WLAN security is not different either since the proposed system will be independent of the transmission technology being used.

Future work

There are obviously many details that need to be worked out before it can be deployed in a larger scale. However, the purpose of this document is to leave implementation details to a later stage and let the overall architecture and the relative simplicity of the solution to shine through. We have also intentionally tried to use standard protocols and standard solutions as much as possible, although new protocols and products will be required at various places (such as firewall rule deployment, user interaction with the system, etc.) and all that need to be addressed in future more detailed design document if enough interest exists in going forward with this solution.

The distributed firewall concept proposed here already exists to a certain degree within the AppGate solution. However it must be extended to support virtually all kinds of devices and other firewalls (such as IPfilter, IPchains, etc.) and the protocols for controlling firewalls and collecting logs and sending alarms in a secure way must be investigated further to make them vendor-independent and universally usable.

Redundancy is another field that need to be further analyzed. Although the architecture makes it possible and easy to add individual components, redundancy issues must be addressed at a more detailed scheme than what has been done here.

6 About AppGate Security

6.1 The AppGate Concept

The AppGate concept was borne in the second half of the 90s. It came about because of a realisation that there was a new way of designing networks apart from the unsecured client/server based networks and the portal/terminal emulation model. The AppGate concept realised that users often prefer to use the client/server model but that the portal/terminal emulation model offers enterprises more control and security (keeping data and servers away from user machines). So the AppGate concept was borne; a managed portal, which provides users with easy access to either client/server or terminal emulation, based applications, and provides enterprises with the highest levels of availability, control and security.

The concept was based around several key points:

- the need to separate users from the servers
- the need to protect servers (which often contain IPR, sensitive data, etc)
- the need to provide users with fine grain access control
- the need to secure user connections
- the need to support most types of network traffic (printing, UDP, ICMP, etc)
- the need to be able to connect from anything anywhere anytime
- the need to use open standards

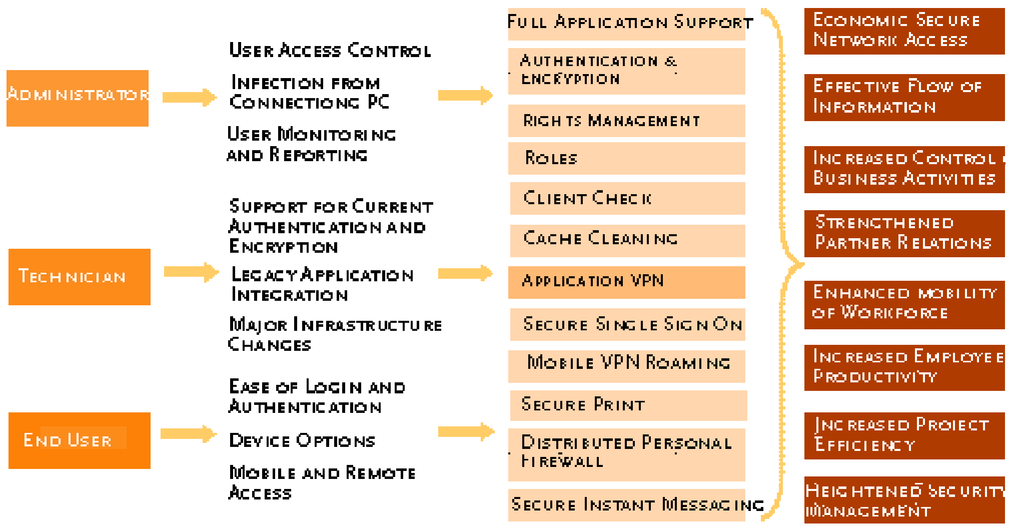
Many of these key points align with the sort of issues critical to the Jericho forum and the debate they are encouraging. This also *means that many of the features as suggested to the Jericho solution here actually are tested in live environments*, both for internal as well as external access to sensitive services.

6.2 The AppGate Solution

Highly dynamic, scalable and flexible, the AppGate technology easily integrates with any system or operating environment. This flexibility allows an enterprise to consolidate existing security tools, such as firewalls and PKI, while at the same time reduce complexity.

The AppGate solution ties together all the pieces of security technology in one easy-to-use system. The solutions work in both fixed and wireless network environments, with the broad range of client systems and scales from small user systems up to enterprise level customers, supporting thousands of users.

AppGate customers range from doctors who want to access patient information from outside the hospital to pharmaceutical companies that wants to control who can access sensitive research data internally. Sales people who travel frequently can access information with a mobile phone or a PDA and companies can invite partners and customers to share data in projects. In the government area, customers use AppGate to share resources internally.


Administrator
Technician
End User
User Access Control
Infection from Connecting PC
User Monitoring and Reporting
Support for Current Authentication and Encryption
Legacy Application Integration
Major Infrastructure Changes
Ease of Login and Authentication
Device Options
Mobile and Remote Access
Full Application Support
Authentication & Encryption
Rights Management
Roles
Client Check
Cache Cleaning
Application VPN
Secure Single Sign On
Mobile VPN Roaming
Secure Print
Distributed Personal Firewall
Secure Instant Messaging
Economic Secure Network Access
Effective Flow of Information
Increased Control Business Activities
Strengthened Partner Relations
Enhanced Mobility of Workforce
Increased Employee Productivity
Increased Project Efficiency
Heightened Security Management

Necessary functions in an industry-strength solution

6.3 The missing components

If we try to map the Jericho vision on to the AppGate solution as it is today, then there are many things that need to be done. We have already identified a few key areas that we consider the most important in terms of defining work packages aimed at filling in some of the missing functionality pieces in the Jericho jigsaw. The work packages would probably comprise:

- An element of discovery (how to find the PPOI)
- A definition phase (where a standard is proposed including places where AppGate uses proprietary solutions)
- An evaluation period (where ease of implementation and user experience is tested)

AppGate Network Security would be pleased to be invited to join the process of helping build some of the missing pieces.

6.4 AppGate contacts

This paper has been prepared by:

Tomas Olovsson & Jamie Bodley-Scott
CTO — UK Operations Manager
+46 730 667730 — +44 870 4460223
tomas@appgate.com — jamie@appgate.com

AppGate work exclusively through partners in the UK. This paper has been prepared in partnership with Somerford Associated whose contact details are shown below.

Andy Davies
+44 1793 698047
andy.davies@somerfordassociates.com

7 References

[1] *Distributed Computing Environment (DCE):* http://www.opengroup.com

[2] *Kerberos*: The Network Authentication Protocol: http://web.mit.edu/kerberos

[3] *Lightweight Directory Access Protocol (v3): Technical Specification.* RFC 3377 – J. Hodges, R. Morgan. September 2002, http://www.ietf.org/rfc/rfc3377.txt

[4] *LDAP* – open source implementation: http://www.openldap.org

[5] *The TLS protocol*, version 1.0, RFC 2246, http://www.ietf.org/rfc/rfc2246.txt

[6] The Secure Shell (SSH) protocol, The SSH Working group: *http://www.ietf.org/ids.by.wg/secsh.html*

[7] *Security Architecture for the Internet Protocol, (IPsec),* RFC 2401 etc, Kent and Atkinson, November 1998: http://www.ietf.org/rfc/rfc2401.txt

[8] Addition of Kerberos Cipher Suites to Transport Layer Security (TLS) – RFC 2712, October 1999: http://www.ietf.org/rfc/rfc2712.txt