Global Information Assurance Certification Paper

Securing Citrix NFuse and Metaframe using Appgate
Jerry Matt
December 31, 2002
Security Essentials (GSEC) - Version 1.4b – Option 2

Abstract

This paper is a case study on a project that provided browser based anytime, anywhere access via the Internet to a hospital application on Citrix Metaframe using AppGate SSH VPN. This paper begins with an introduction to the project requirements and defined risks. The paper then explains the research and design considerations leading to the products chosen to provide the secure access to the Metaframe application. Finally, this paper explains how the implementation process was completed and what the final result accomplished. The final result combined two core products, Citrix and Appgate, to create a secure and encrypted portal to deliver both web-based and legacy applications through a web browser with a zero-footprint client.

Before

In March of 2001, I was involved in a strategic project to deliver a new application to partners of our organization via the Internet. This application had security requirements beyond normal remote access requirements because it was delivering patient data through the Internet. The requirements and timelines set forth by management were very aggressive. Therefore, the delivery of the application had to be through a browser, which was secure and "anywhere, anytime." The project also required implementation in six months. The basic requirements are listed in the Table 1.

TABLE 1

Authentication	Authentication must support a variety of methods, more importantly support two-factor authentication.
Encryption	Data must be encrypted through the Internet. The encryption should be standards based.
Zero Footprint	Our organization cannot physically install any software on our partner's computers. Must be able to go anywhere there is an Internet connection and access the application.
Low Bandwidth	Performance must be adequate under all bandwidth conditions.
High Availability	Must be available 99.9%

With the high level requirements set, a project team was assembled to implement a solution that met the requirements. The team consisted of three groups: a technical team, project manager team, and management/leadership team. The

technical team was going to be responsible for the research, design, and implementation of the technology. The project management team was responsible for maintaining project deadlines and managing the resources from other teams outside the project. Finally, the management/leadership team was the sponsor of the project; they made any high level decisions and oversaw the decisions made by the project and technical teams.

I was an engineer on the technical team, which also consisted of a consultant, lead technical engineer, web developer, and part-time resources from other teams in the organization. These part time resources included workstation, operations (monitoring), and telecom support.

Since this was the first time our organization was using the Internet as a means of application delivery, the team assembled to implement this project had an advantage to build security from the beginning. As many security professionals are aware, it is much easier to incorporate security in the beginning of a project as opposed to reconfiguring something already built.

From the beginning security was of utmost importance. Anytime the Internet is used in a project, risk needs to be evaluated. Some of the major discussions of the project team dealt with how to mitigate risks dealing with authentication, encryption, and overall design. For example, when dealing with authentication, what is the appropriate level? Does the implementation require the need for just a single password, or should some type of two factor authentication be used? These types of decision are important due to what our organization is protecting - patient data. These decisions impact not only legal issues if the data was compromised, but also community trust.

The clinician application that needs to be delivered to our partners is the same application our organization uses internally. This application is very large and needs updating continuously; therefore, a thin client solution using Citrix Metaframe was used. Thin Client Computing is defined by the fact that the application is executed on the server and displayed on the client system. Therefore, a "thin client" terminal need only have sufficient power to render the display of the user session (http://www.thinclient.net/technology/history.htm#Thin%20Client/Server) .Thin Client computing normally uses a protocol such RDP (remote desktop protocol) or ICA (independent client architecture). These protocols send only keystrokes, mouse clicks, and screen prints from the client to the server.

Many of the meetings in the beginning of the project dealt with what type of authentication should be employed to protect the application. Passwords, One Time Passwords, RSA SecurID were all discussed as possible ways to authenticate our users. As a project team, we decided that that we would implement two factor authentication using RSA SecurID, which generates a new, unpredictable code every 60 seconds. The user combines this number with a

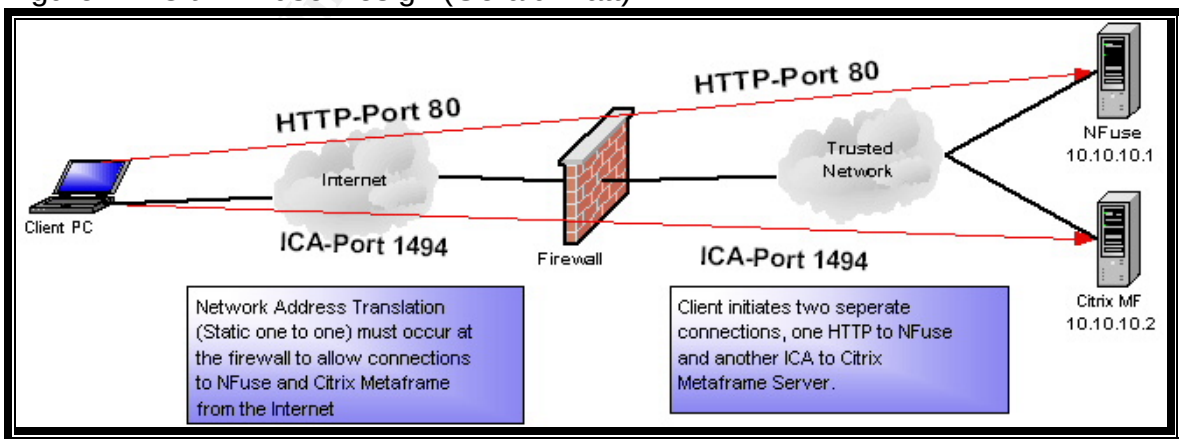
secret PIN to log into protected resources. Each authenticator has a unique 64-bit symmetric key that combines with a powerful algorithm to generate a new code every 60 seconds. Only the RSA ACE/server software knows which number is valid at that moment in time for that user/authenticator combination (www.rsasecurity.com/products/sehcurid/tokens.html). This would give our organization a high level of protection that we felt was needed to protect "the front door."

Once the authentication discussion ended, the project team started to research technology to deliver the application to our health partners. Since the application that provides patient data is installed on and implemented using Citrix Metaframe, the first step was to determine what products Citrix offers that could fit our requirements.

Researching Citrix product offerings, Citrix NFuse seemed to fit our requirements. Citrix NFuse enables organizations to integrate and publish applications into any standard browser and create enterprise portals that give users personalized, web based access to all the tools, information and applications they need (http://www.citrix.com/products/nfuse_classic.asp).

The initial research of Citrix NFuse showed much promise for the implementation of our project; however, there was a security concern in the design of NFuse. NFuse required that all the Metaframe servers to be used through the Internet be publicly addressed with port 1494 open through the firewall[1]. This caused security concerns because the current Metaframe farm was located on the trusted network. That being the case, any user on the Internet could try to attach to the Metaframe farm on 1494 using an ICA Client and get prompted for username and password. At this point any one could try to guess usernames and passwords to get a Metaframe session. This would bypass any front-end security that NFuse would provide.

Figure 1 – Citrix Nfuse Design (Gerald Matt)



[1] Citrix Secure Gateway eliminates the need to publicly address all of the Metaframe servers; however, this product was not available at the time the project was started and implemented.

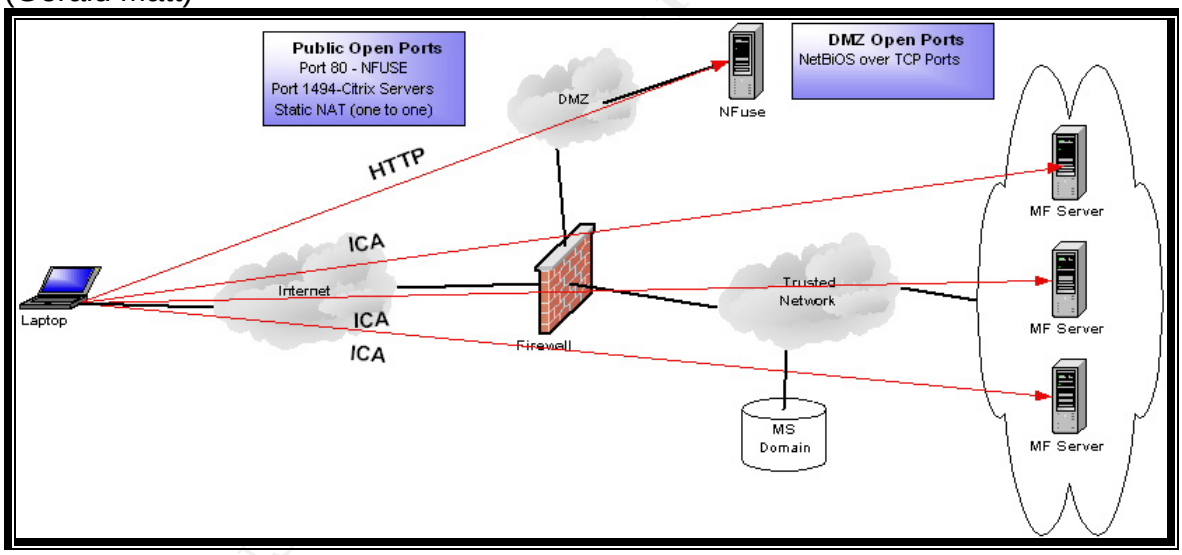
Discussions then lead down the path of where to place the Metaframe Farm. This was where the project team split. One design allowed users from the Internet to make a direct connection to the trusted network, citing there were no known vulnerabilities to Citrix on port 1494. Flaws in this design included the following:

1) Our organizational security policy did not normally allow direct access to network resources from the Internet without a DMZ or proxy.

2) In the case of a server being compromised, there is no logging since it is in the core of the trusted network. Also, there is limiting of services and destinations.

3) Future vulnerabilities could include Citrix ICA services.

Figure 2 – Implementation of Citrix NFuse with MF Farm in Trusted Network (Gerald Matt)

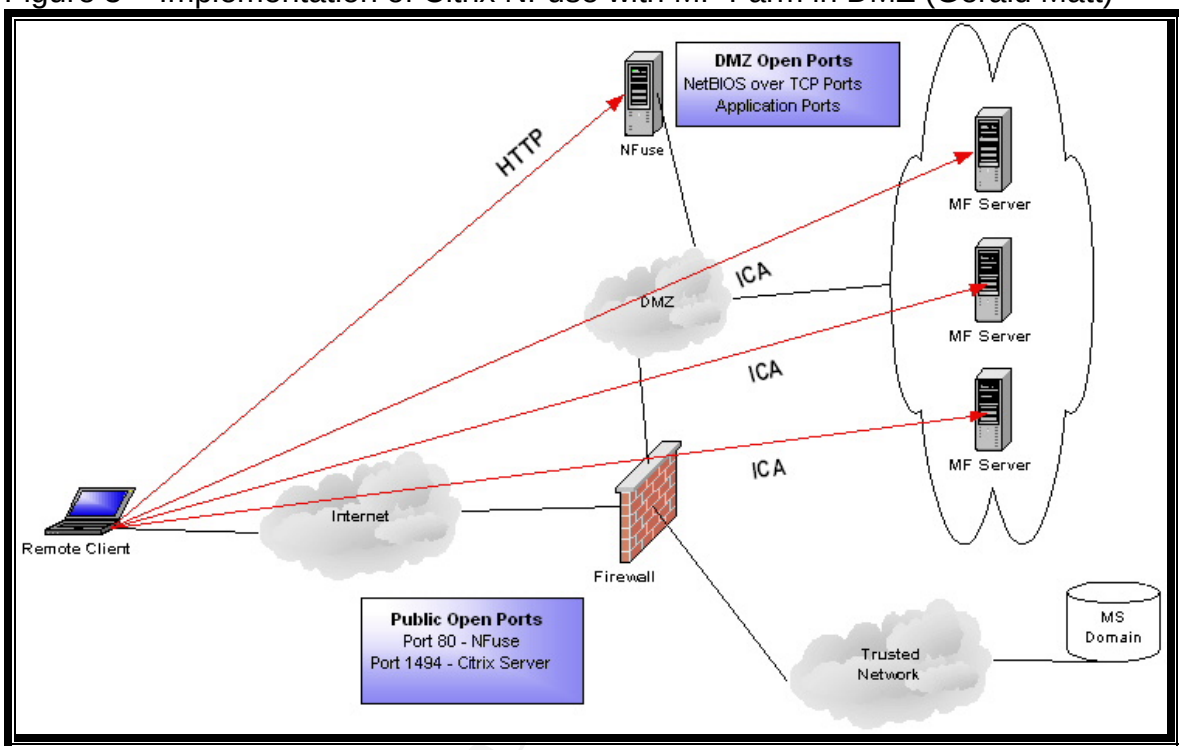


The other design considered an alternative placed not only NFuse, but also part of the Metaframe farm in the DMZ. This designed prevented users from making direct connections onto our trusted network, as well as providing logging and filtering to resources residing on the trusted network. Flaws in this design were as follows:

1) Splitting the Metaframe farm would cause our Citrix administrators to install an ICA gateway, which is not normally best practice.

2) There would be too many rules to open through the firewall because Citrix needs to communicate with our Microsoft domain as well as any server portions of client/server applications residing on the Citrix server.

Figure 3 – Implementation of Citrix NFuse with MF Farm in DMZ (Gerald Matt)



This issue became a deadlocked one and neither group could overcome it. Since security was a major concern, the project team agreed to look at other enabling technologies to deliver the application.

While pursuing other technologies, myself and the other engineer were implementing Secure Shell (SSH) for administration of our firewalls. Secure Shell, also known as SSH, is a protocol which permits secure remote access over a network from one computer to another. SSH negotiates and establishes an encrypted connection between the SSH client and an SSH Server, authenticating the client and server in a variety of ways. The connection can then be used for a variety of purposes, such as creating a secure remote login on a server or setting up a VPN (http://www.rsasecurity.com/rsalabs/faq/5-1-5.html).

While looking for a better SSH client, I ran across a Java SSH client called Mindterm. Using Mindterm gave the project team an idea on another possible design for our project. We could use a webpage to download the Java SSH client, then create SSH port forwards to give an authenticated session into our organization's trusted Citrix Metaframe network.

The initial testing was not very successful for the following reasons:

1) The download of the Mindterm Java SSH client was fairly large (over 500K) for dialup users. It also needed to be downloaded every time.

2) Users would have to manually create the SSH port forwards in the Mindterm client.

3) User administration could be very difficult.

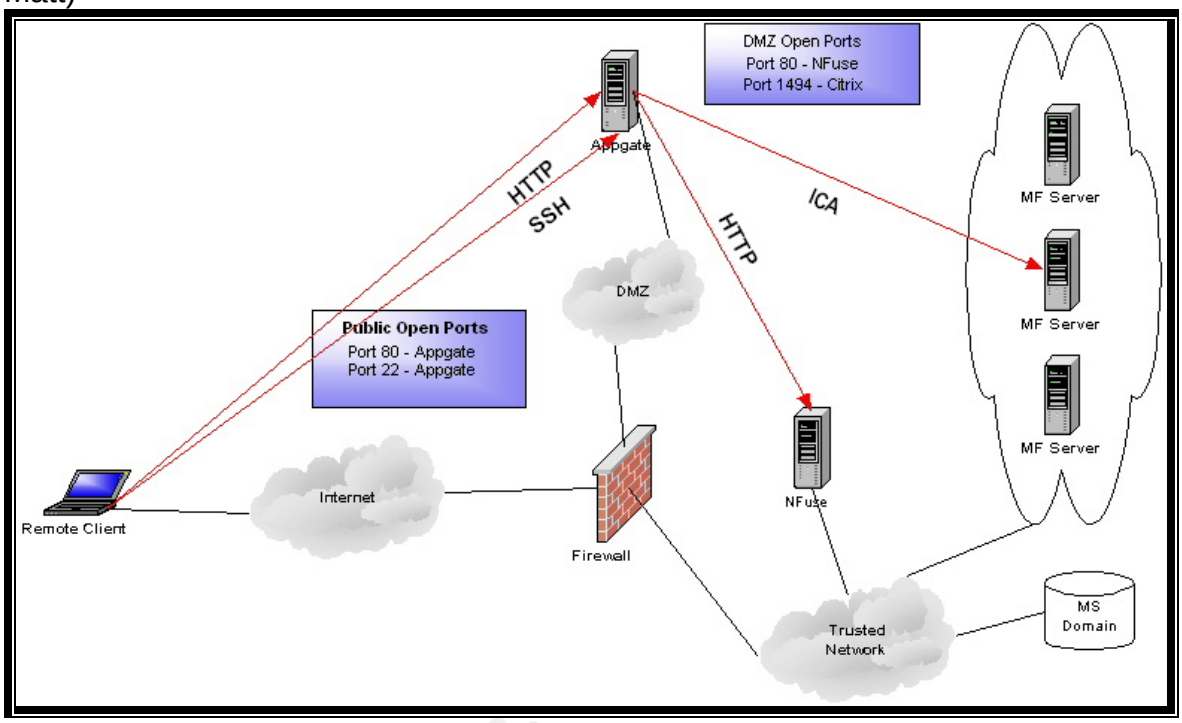
We contacted the vendor, AppGate, who supported Mindterm SSH client to see if they could help us resolve some of the issues. While explaining some of our technical issues and concerns, they pointed us in the direction of their flagship product, also called AppGate. They explained that using AppGate could solve the majority of our concerns.

AppGate is comprised of two main components, the AppGate Client and the AppGate Server, and establishes secure communication between client and the server. The AppGate client creates a secure tunnel between the user's workstation and the AppGate server, using the SSH protocol. The AppGate server requires the user to provide authentication, which supports a variety of authentication methods. Once authenticated, the user is presented with a list of applications and services, based on predetermined criteria. All traffic between the AppGate Client and the AppGate server is encrypted, no matter where the user is located (http://www.appgate.com/products/appgate.htm).

The project team was very excited to research further the possibilities of using AppGate as a front end to NFuse to solve our solution. AppGate had many positive features in our initial research. First, the Java SSH applet was delivered via HTTP through a web browser, thus allowing a zero-footprint on our partner's workstations. The SSH Client applet was also cacheable, meaning after the first time the applet is downloaded in resided locally on the machine. The AppGate application also runs on a Solaris OS platform, which fit our organization's direction for Internet (DMZ) devices. This was important because of the wealth of information on OS hardening procedures related to Solaris. Using AppGate eases Internet firewall administration because the firewall administrator just needs to manage two ports on the Internet (80-HTTP, 22-SSH), eliminating the need to use NAT for all the Citrix servers. All other services are hidden behind the SSH port. Finally, AppGate provided a Java-based front-end for ease of administration for all the port forwards based on custom defined roles.

The project team felt that AppGate with a combination of Citrix NFuse could help us provide a secure solution. AppGate SSH proxy could sit in the DMZ, and proxy/forward requests to and from Citrix Metaframe and NFuse, thus eliminating many of the security nuances of just implementing Citrix NFuse.

Figure 4 – Implementation of Citrix NFuse with AppGate as a front-end (Gerald Matt)



The table below (Table 2) shows how AppGate fit our basic requirements. The next step was to get a proof of concept built to verify the technology research could work.

TABLE 2

Authentication	Supports Unix Passwords, RSA SecurID, Public Key, Entrust, Cryptocard, Baltimore, SmartTrust, RADIUS
Encryption	Supports up to 256 AES, Blowfish, 3DES, and ArcFour
Zero Footprint	Java SSH client delivered via HTTP on a web server
Low Bandwidth	Java applet is cacheable on the workstation. Also ICA is a low bandwidth protocol
High Availability	AppGate supports application server clustering

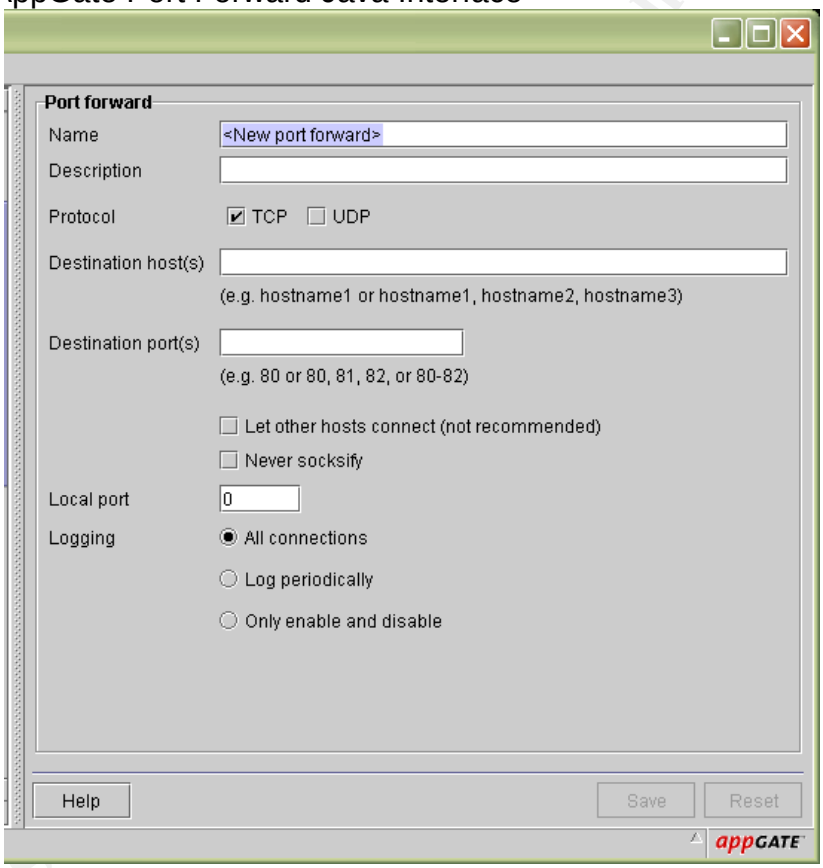
During

The proof of concept needed to be built by the technology team to determine if the design would work. The Citrix Metaframe farm running the clinician application was already built for our organization. Our team just needed to build an AppGate server and Citrix NFuse server.

Since our organization did not have any Sun hardware available, we decided to build the proof of concept using Solaris 5.7 for X86, since AppGate supported Solaris X86. Although the hardware would not be used in production, Solaris X86 was very similar in operation and function to Solaris 2.7 built on Sun hardware. Citrix NFuse runs on the Windows Platform; therefore, plenty of hardware was available also.

Installing AppGate was very easy since it came in an installation package used by Solaris. Once AppGate was installed, we started to configure the application to create the port forwards. Port forwards using AppGate were easy to create using the Java administration console. Figure 5 is an example of the interface:

Figure 5 – AppGate Port Forward Java Interface



Since our organization was a prospective client, AppGate gave us the use of their support resources to help us with installation and configuration. The technical team ran into a few small issues, but none that were not resolved fairly quickly with a call to support. After the configuration was completed, we hardened the OS based on SANS step by step hardening for Solaris and placed it in our Internet DMZ.

Before placement into the DMZ we needed to make all necessary policy changes. These changes included presenting our design to our weekly change control meeting. This meeting is used by the organization to track changes to the infrastructure, as well as keeping the technology team abreast of projects that could impact their team.

After AppGate was configured, we started work on NFuse. NFuse required a little more configuration since it needed to be part of our Microsoft domain; further, the Internet Information Server (Microsoft Web Server) needed to be configured according to our organizational standards. Once NFuse was configured, we needed to start "proving" our design could work.

There were some difficulties in trying to integrate AppGate with Citrix NFuse and Metaframe. Most of the issues dealt with the design of SSH port forwards and Citrix NFuse's application delivery method. In order to understand the issues, I need to explain the design of NFuse and how it "web enables" applications on Citrix Metaframe. I will then explain how AppGate SSH port forwarding works. NFuse works in the following manner:

1) NFuse runs on Windows NT/2000 with IIS installed. IIS allows the web server to deliver the contents to the Internet browser.

2) Citrix provides a downloadable ICA client using ActiveX, which can be imbedded in the web page. If the browser does not have the ICA client, it will be downloaded and installed automatically.

3) When a client requests the default page, IIS renders a HTML page with username and password. The client enters their Microsoft domain credentials.

4) The credentials are verified to a Microsoft domain controller. If the credentials are accepted, NFuse renders a list of applications based on Microsoft domain group membership. (Citrix Metaframe publishes applications based on Microsoft domain groups). Also NFuse puts an encrypted cookie with the client's domain credentials in the Internet browser.

5) The application list is displayed using HTML. When the client clicks on the hyperlinked application, NFuse uses XML to communicate with the Citrix Master Browser. The Master Browser role ensures that the Citrix server that houses the application is up, and also provides load balancing services for the Citrix Metaframe farm. The Master Browser then communicates back to NFuse with the IP address of the Citrix server that has that application available.

6) NFuse receives this information, and then creates a dynamic ICA file with the IP address of the application server. NFuse also inserts the client's domain credentials (encrypted) into the ICA file. This is needed or the client would have to enter their domain credentials every time they choose an application using NFuse.

7) The Internet browser downloads the dynamic ICA file by using HTTP, and employs the ActiveX ICA client to communicate directly to the server using the ICA protocol on port 1494. The ActiveX client is run in the browser, displaying the application. ICA is natively encrypted.

The AppGate SSH client, after authentication, binds ports to its localhost (127.0.0.1) based on the user's role membership. When an application, such as an Internet browser, points to its own localhost (example http://localhost:12345/index.html) with the specific port forward; the SSH client intercepts the data. Once the data is intercepted, the SSH client encrypts the data and sends it to the server. Once the server receives the data, the server decrypts it and forwards it to the destination (defined by using AppGate Java Console) server using its native protocol.

Now that I have explained how both NFuse and SSH port forwards work, I can explain the technical team's dilemma – how do we get NFuse to send a dynamic ICA file to an Internet browser with an IP address of 127.0.0.1 instead of the private IP address of the Citrix server? To answer this question we needed to get Citrix Support involved. After Citrix reviewed the problem, they determined that the code of NFuse needed to be changed for NFuse to use localhost. Custom code needed to be written for the point just before NFuse sends the dynamic ICA file to the client's Internet browser. The programming basically takes the private address, and if it matches, turns it into 127.0.0.1 with a unique port number assigned to the localhost. Figure 6 is the actual code changes made to the launch.asp file on NFuse.

Figure 6 – NFuse Code changes to the launch.asp file

```
icafile = CStr(icafile)
icafile = Replace (icafile, "10.10.10.1", "127.0.0.1:2300")
icafile = Replace (icafile, "10.10.10.2", "127.0.0.1:2301")
icafile = Replace (icafile, "10.10.10.3", "127.0.0.1:2302")
icafile = Replace (icafile, "10.10.10.4", "127.0.0.1:2303")
icafile = Replace (icafile, "10.10.10.5", "127.0.0.1:2304")
icafile = Replace (icafile, "10.10.10.6", "127.0.0.1:2305")
icafile = Replace (icafile, "10.10.10.7", "127.0.0.1:2306")
icafile = Replace (icafile, "10.10.10.8", "127.0.0.1:2307")
icafile = Replace (icafile, "10.10.10.9", "127.0.0.1:2308")
icafile = Replace (icafile, "10.10.10.10", "127.0.0.1:2309")
icafile = Replace (icafile, "10.10.10.11", "127.0.0.1:2310")
```

This resolved our port forwarding issues using AppGate, and distinguished between each Citrix Metaframe server in our farm. Although there is some administration involved in maintaining the code, our team felt that it could be added into our procedures when installing a new Metaframe server. When that problem was resolved, we then had the ability to make a full connection using SSH port forwards from NFuse and Metaframe.

Another issue that our technical team faced was that since the SSH Java client runs in an Internet browser, if the window changed to a different site or closed, the SSH sessions would be lost. Moreover, our partners, after authentication, would need to manually enter the website of NFuse to get the applications. Having the Java applet, after authentication, create a new browser window and forward it to the NFuse web page solved this. We also put a note on the webpage not to close the previous window or the connection would be closed. This was to be documented in our procedures and is communicated to new users during their initial training.

The technology team tested all connectivity from AppGate to Citrix NFuse and Metaframe. The basic infrastructure using the Java-based SSH client worked through the Internet. Next, we needed to test the encryption using a packet sniffer. The team verified that the packets entering AppGate were encrypted. Next, we tested the firewall configuration by placing a laptop in the DMZ where AppGate was located. By placing a laptop in the DMZ, we could mimic what services a rouge machine in the DMZ could access in our trusted network. Finally AppGate was tested to verify it could only reach services on the trusted network that it was allowed to by the firewall. After finishing the testing of the design, the next step was to present our findings to the management team for approval of the pilot.

In order to set up a pilot, we needed approval from the management team to move forward. The project team presented the design and testing plans to the management team for approval for capital to start the pilot. The pilot would be based on the recommended hardware and OS platforms for both AppGate and NFuse. After careful deliberation of the management team, they decided we could move forward with the purchase of the equipment and software to make the project a success.

After the approval of capital from the management team, the technology team needed to get quotes on hardware and software for completing the project. Once the quotes were in, we decided on a vendor to purchase the equipment and went forward with ordering. Shipping of the equipment went fairly fast and the technology team was very busy setting up the hardware and software. We ran into one specific issue with video cards for our Sun hardware. We only had one video card and the rest were backordered. This made installation difficult

because we needed to swap the video card for each system that needed to be built. Once the backordered video cards came, installation went much faster.

Once the hardware was set up, we needed to install and configure the OS and applications on both the UNIX (Appgate and RSA ACE) and Windows (NFuse) platforms. Two teams performed the installation: one team worked on the UNIX installation and hardening while another team installed and hardened the windows platform. After the OS installation and setup, hardening procedures using SANS security step by step guide were implemented on both OS platforms. Finally, configuration of RSA ACE server, AppGate, and NFuse was completed. The application installation and configuration moved fairly fast because the configuration was based on our proof of concept. With the systems built and tested we needed our partners to start using the system to complete our pilot.

After

The pilot was to be a month long with a limited pool of partners. The customer partners were chosen at random and given training on how to use the system. Our training department completed this, and our partners were also given written instructions. After the pilot was completed, a survey was to be sent out for feedback that we could use to improve the system before the go-live date.

After the pilot was completed, just before the go-live, the management team felt they needed to have an audit on the design from an outside consultant team. The outside consultants were hired to verify the design and make sure the project team did not miss any glaring security holes. The consultants interviewed members of the technology team and reviewed all documentation to the project. After the audit, the consultants presented the management team with their findings which approved the design.

Stability of the system went really well, with minimal downtime during the pilot. Pilot partners were required to login and use the system a couple of times a day. The month long pilot ended with the surveys sent back for analysis. Most of the pilot partners were happy with the system; however, a common theme among the complaints was the number of logins the partners had to endure. The system required front end authentication using RSA Tokens, which forwarded the users to NFuse. NFuse required Microsoft domain authentication to display the NFuse icons for the applications delivered though Citrix Metaframe. Finally, there was a separate authentication request to get into the clinicians' application. We could not figure out a way (within reasonable cost) to reduce the number of logins. Another issue we ran into was firewall configuration blocking the SSH port. To get around this, we worked with the technical staff at our partner sites to give the appropriate lock down to use AppGate through the partner's firewall.

We presented the survey results to our management team with knowledge that the login problem could be worked on in the future. The management team accepted that issue, and we began the rollout and training of the partners for using the system.

Future Improvements

Future improvements for the system include reducing the number of logins to get to the clinician applications, as well as providing the functionality using handheld devices. Another possible improvement is running AppGate SSH on the SSL port (443) to allow easier firewall traversal at partner sites with restrictive firewall policies.

Summary

The infrastructure was built and tested, and the rollout of the system to our partners was completed on schedule. The team felt we designed and implemented a very secure system, meeting the requirements set forth by management. This design could easily support any windows application via the Internet using Citrix NFuse and Metaframe, as well as any future web based applications by using AppGate port forwards directly to any web server. One of the biggest issues was delivering applications to partners without installing and maintaining a client on their PCs. AppGate provided us with a zero-footprint client, as well as strong authentication and encryption solutions. Further, it was easy to administer and support, provided greater security to our organization, and allowed us to give authenticated users a greater number of application services. NFuse solved our application delivery method by providing load balanced applications running on Citrix Metaframe in a web browser. The combination of the two technologies created a system that is scalable, secure, and stable.

References

Greenberg, Steve “What is thin client computing” URL: http://www.thinclient.net/technology/history.htm#Thin%20Client/Server (December 31, 2002)

RSA Security, Inc. “RSA SecurID – The golden standard of two factor authentication” URL: http://www.rsasecurity.com/products/securid/tokens.html (December 31, 2002)

Citrix Systems, Inc. “Citrix NFuse Classic” URL: http://www.citrix.com/products/nfuse_classic.asp (December 31, 2002)

RSA Security, Inc. “RSA Laboratories’ Frequently Asked Questions about Today’s Cryptography 4.1 – What is SSH” URL: http://www.rsasecurity.com/rsalabs/faq/5-1-5.html (December 31, 2002)

Appgate Network Security “AppGate Functionality Overview” URL: http://www.appgate.com/products/appgate.htm (December 31, 2002)

Figures 1-4 created by Gerald Matt (December 28, 2002)