

Issues in Securing Remote Access Servers

Background.....	2
Terms used.....	2
Remote access server.....	2
Gateway.....	2
Service provider	2
Well-recognized SSL certificate.....	3
Topologies for connecting to remote access servers.....	3
Direct connect.....	3
Gateway connect	3
Securing a gateway connection with SSL.....	5
Two-stage SSL.....	5
Split handshake	6
Back-end tunnel.....	7
Pseudo-SSL.....	8
Proprietary methods.....	9
Failures of proprietary methods	10
Problems.....	11
Which server is authenticated?.....	11
Is anyone listening in?.....	12
Does the gateway work as advertised?	12
Why do these problems exist?	12
Conclusion	14
Lessons from the past.....	14
Clipper chip	14
Parallels to Clipper.....	14
Turning “We won’t look” into “We can’t look”.....	14
Endnotes	15

Sericon Technology Proprietary and Confidential – No part of this document may be reproduced, copied or distributed in any fashion without the express written permission of Sericon Technology Inc.



Background

In recent years there has been an increase in the number of applications and devices that enable remote access to data over the Internet. As more people have always-on broadband connections at home, these applications are no longer restricted to corporate environments. People want these capabilities not only to access business data, but also to access their own data: to share files such as digital images with their friends and family, to access their own large digital media stores from remote locations, and to access their personal files when they are away from their computers.

Using these powerful data-sharing applications responsibly means ensuring that the data they serve remains secure and private. This is more than simply guarding access to these resources with user names and passwords: it means also providing a mechanism so that no one can view private data as it is transmitted from the remote access server hosting it to its final destination.

This document examines the different ways these remote access servers are typically configured and how they transmit data to its destination. It focuses on the methods for securing this transmission against eavesdroppers.

Terms used

Remote access server

A *remote access server* is any application or device that enables sharing files or services over the Internet, for example: Web servers, file-sharing systems (P2P or otherwise), remote computer control applications, Internet hardware devices, and network attached storage (NAS) devices that also allow you to make your data available across the Internet. In the future, this list could include mobile devices such as cell phones and PDAs if they are equipped with a mass storage device and software to allow them to share its contents.

Clients typically use an ordinary Web browser to interact with a remote access server. This makes it easy to see which security methods are in place, as typical Web-security methods can be used. Where a simple HTTP browser is not used and the client interacts with a Java applet or an ActiveX component, the security aspects are sometimes less transparent.

Gateway

A *gateway* is a service that runs somewhere on the Internet and helps a browser connect to a remote access server. All data transmitted between the browser and the remote access server flows through the gateway. Typically, a user logs into a gateway and then requests a connection to a remote access server. Gateways may provide the following functionality:

- Connectivity – the remote access server may not have to open a port to allow inbound communication
- Security – the gateway may be essential in helping the remote access server operate securely in a cost-effective manner
- Discovery – the gateway may help browsers find the location of a remote access server

While gateways are not required to run a remote access server, they can accomplish one or more objectives in an easy or cost-effective way.

Service provider

A *service provider* is the company responsible for developing a remote access server. If a gateway is a method of connectivity, then the service provider is responsible for it, too.

Well-recognized SSL certificate

A *well-recognized SSL certificate* is a certificate recognized by a high percentage of browsers deployed across the Internet. This is in contrast to a certificate signed by a private CA or self signed. By deploying only well-recognized SSL certificates that are viewable by consumers, the spirit of trust in SSL is maintained: people are discouraged from blindly accepting certificates into their browsers without understanding the consequences. This should be avoided because it legitimizes unsafe practices and is ultimately dangerous.

Topologies for connecting to remote access servers

A remote access server can run on a home or corporate computer. The main requirement for running such an application is always-on connectivity. To understand how to secure remote access servers, we must understand the different ways that browsers can connect to them.

Direct connect

The simplest way to connect to a remote access server is directly. This is how Web servers normally work, and it has several requirements:

- The remote access server must be locatable on the Internet. This means that a naming mechanism must exist, which is usually DNS for Web servers (or something similar), or another specialized system for software such as a P2P application.
- The remote access server must be connectible. This means that it provides an open port on the Internet on which the server listens. If the computer is directly connected to the Internet, this is easy; if the computer is attached to a router or running behind a firewall, it is more difficult. In that case, the user must open a port on the firewall and forward it to connect the externally opened port to the port on which the server listens. Configuring this is complex (even if helped by technologies such as UPnP), and can be very intimidating to many users. Note too that some people may consider it dangerous to open a firewall port.

Securing a directly connected remote access server is fairly straightforward. While SSL is considered the tool of choice for this, it is usually expensive and difficult to install: several steps are required to create, sign, and install an SSL certificate, which provides the solution. This is beyond the capabilities of many casual users. SSL certificates are also notoriously expensive, with a well-recognized certificate typically costing at least \$100 per year.

Depending on the application and the number and type of users who connect to a particular remote access server, a private self-signed SSL certificate (which can be free) may be an option. However, permanently installing such a certificate into a browser's certificate store is very dangerous and only people who understand all the security implications should do this. Before installing a certificate, users should consider who has access to the private key, how key distribution is managed, how a certificate is authenticated, and how widely this certificate is deployed. If users do not understand these issues, then they may introduce a security vulnerability into any computer that accepts this certificate.

Gateway connect

A popular way to connect to a remote access server is through a gateway service that runs on the Internet. This service is generally controlled by the service provider, and it can concurrently service multiple remote access servers in diverse locations. This method is popular because it may not require modifications to the firewall that protects the computer on which the remote access server runs. This means that consumers who use a product that runs this way do not need to worry about opening a port on a firewall or port-forwarding, which are very difficult for many people to understand and perform. For corporate users, a gateway connection means that nothing needs to be opened on a corporate firewall, which many IT managers prefer.

Despite these benefits, the gateway has a disadvantage: all traffic between a browser and the remote access server must go through it. This is a disadvantage for the following reasons:

- Most important, the service provider must ensure that it is always available – otherwise access to remote access servers is cut off.
- It is expensive to maintain the servers that host the gateway program and provide enough bandwidth to and from these servers to handle the demand. This cost is often passed on to the customer as a monthly charge for this service.
- A gateway also affects the speed at which data is delivered from the remote access server to the browser, as all data is forced to pass through the gateway.

A connection through a gateway can work in one of the following ways:

- If the remote access server opens a port and allows incoming connections, the functionality is almost identical to a direct connect method. The potential advantages are improving security, discovery, and allowing advanced authentication of a browser. These advantages are probably not worth the cost of running the gateway, and since this mode is very similar to a direct connect method, it is not discussed further here.
- If the remote access server performs remote port-forwarding to the gateway (similar to how remote ports may be forwarded in the SSH Protocol¹) then there are several advantages. In particular, forwarding the port to the gateway means that it is not necessary to open a port on the firewall that protects the computer running the remote access server. This is very important for casual computer users who may not know how to properly configure their firewall or router: if these are done improperly, they risk computer problems.

There are several ways to protect a gateway-connected remote access server with SSL. These methods are discussed in the next section of this document.

Table 1 – Summary of methods for connecting remote access servers

	Direct Connect	Gateway Connect
Firewall	Must open a port	No changes necessary
Speed	Fastest	Data is slowed down because it must pass through the gateway
Cost	No additional cost	High cost to service providers to maintain the gateway – often passed on to the users
Availability	Available as long as the remote access server is operational	If the gateway is unavailable, the server is unreachable
Security	Very high with SSL if used properly	Depends on the method used
Discovery	DNS or similar	Through the gateway

Securing a gateway connection with SSL

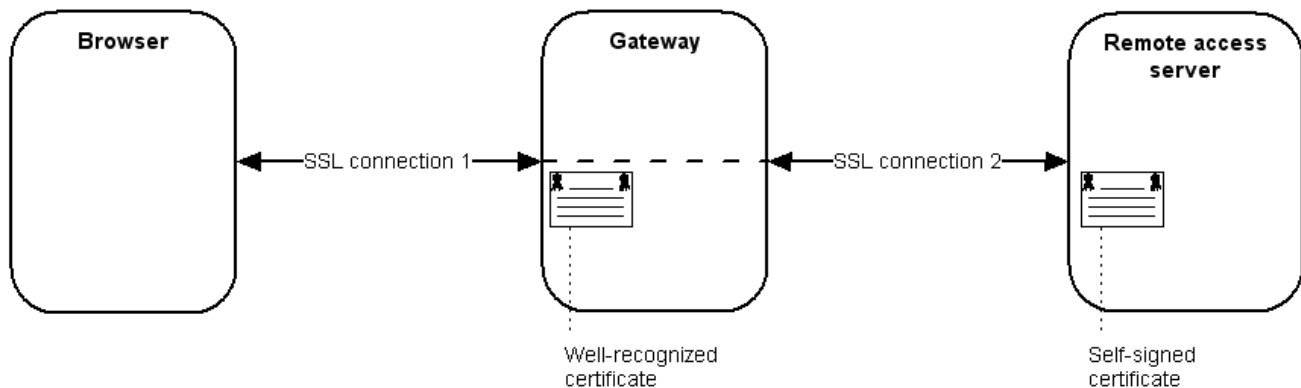
There are several ways to secure the communication between a browser and remote access server through a gateway. The likeliest way is with SSL, which is known to be very secure and is familiar to users. SSL also provides users with information on its usage, so that when an SSL link is in effect, users can easily see and verify information about how their data is secured during transmission. It is important to understand the differences and tradeoffs between the different implementations of SSL when considering how to architect a complete solution, as some methods provide only the illusion of security rather than real security.

Two-stage SSL

The simplest way to provide SSL in a gateway environment is to offer a two-stage solution. When the browser connects to the gateway with intent to communicate with the remote access server, an SSL link is set up between the browser and the gateway. This link is secured by the certificate that exists on the gateway computer. Note that the gateway potentially services many of these connections concurrently, and a well-recognized certificate should be used.

Once the initial link exists, the gateway can set up another, independent SSL link between it and the remote access server. The SSL certificate securing this link need not be well recognized, because the service provider controls both ends of the link and may choose to trust any certificate authority, which means that it can sign its own certificate.

Figure 1 - Data flow for two-stage SSL



It is easy to see that this SSL solution is secure only if the gateway computer is controlled by a trusted entity, for example, if a corporation runs its own gateway to allow access to a pool of remote access servers under its control. In this case, a two-stage SSL approach can work (depending on the type of data transmitted and its security considerations). Otherwise, since the data between the remote access server and the browser is secured by two independent SSL links, it is unencrypted while it passes through the gateway. This means that anyone who has access to the gateway computer can also access the supposedly private data that passes through it.

Split handshake

The split handshake approach to securing a gateway connection takes advantage of an SSL feature known as the *handshake*². The handshake is a set of steps that the client and server perform to secretly agree on how to encrypt the data that will flow through the SSL link. A simplified version of the SSL handshake works as follows:

1. The client connects to the server and indicates that it wants to communicate over SSL. It also generates a random number and sends this to the server. Note that this stage happens over a normal TCP/IP link, and can be seen by anyone who can intercept the data.
2. The server then generates its own random number and transmits it back to the client along with its SSL certificate. Once again, this transmission can be intercepted because it is not encrypted.
3. The client then generates a random string called the *Pre Master Secret*. It uses the server's public key (extracted from the SSL certificate) to encrypt the Pre Master Secret, and sends this to the server.
4. The server can determine the Pre Master Secret because it holds the private key that corresponds to its SSL certificate. Note that only the holder of this private key can perform this function.
5. The client and server independently use a *Key Derivation Function* to generate the *Master Secret* based on the Pre Master Secret and the two generated random numbers.
6. The client and server exchange some additional information to ensure that everything worked correctly.
7. The client and server can begin to communicate over an encrypted channel using the Master Secret as an input to the encrypting cipher.

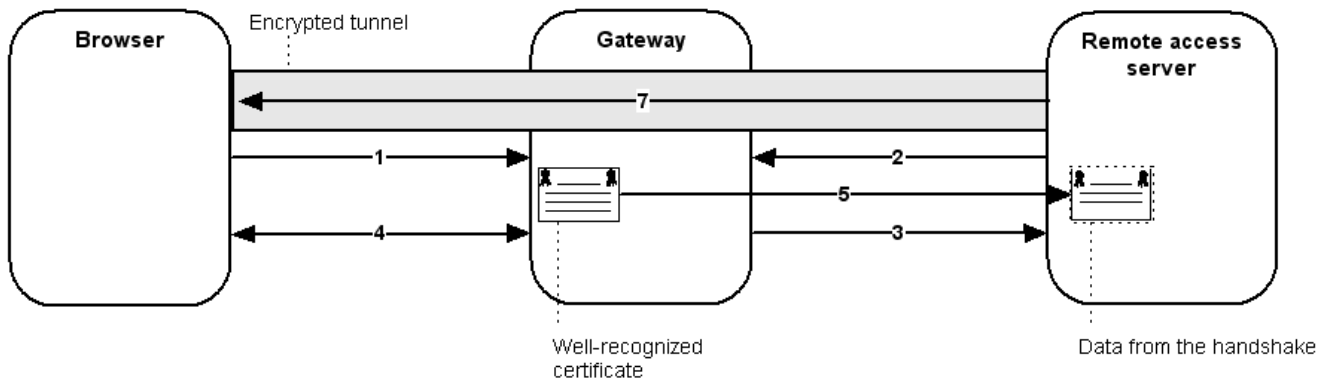
Note that two important things happen during the SSL handshake:

- Information is exchanged so that the client and server can communicate securely.
- The client can positively identify the server with which it communicates based on the information in the SSL certificate. While an optional part of the handshake enables the server to authenticate the client, this is unnecessary for this discussion.

This helps us understand what happens during a split handshake:

1. The browser connects to the gateway with the intent of connecting to a remote access server.
2. The remote access server periodically checks in with the gateway to see if there are any requests for connections to it.
3. The gateway informs the remote access server that there is a pending connection, and allows the remote access server to forward a port to it.
4. The gateway replies to the browser, and begins the SSL handshake using its well recognized certificate.
5. After the Pre Master Secret is received and decrypted by the Gateway (see Step #4 above), the Gateway transfers all of the intermediate SSL handshake data (e.g. the Pre Master Secret and the generated numbers from the earlier part of the handshake) to the remote access server. In other words, the handshake is "split".
6. The remote access server and browser independently calculate the Master Secret.
7. Now that a Master Secret exists, an encrypted tunnel exists between the browser and the remote access server through the gateway. Note that the purpose of the gateway at this point is simply to transmit encrypted TCP packets between the two ends.

Figure 2 - Data flow for split handshake



We should note the following important issues about this system:

- The remote access server can complete the SSL handshake without ever gaining access to the gateway's Private Key. This is very important because there are many SSL sessions going through the gateway, and if one remote access server has access to this key, it can decrypt the data of another remote access server's session.
- The gateway sees all traffic between the browser and the remote access server, and since it has access to the data used to compute the Master Secret, it **can** decrypt all data sent between the browser and the remote access server. Companies that utilize split handshake will claim that they cannot look at customer data when it passes through the gateway because the data is encrypted. However, this is misleading, since the gateway has access to all the information necessary for decryption.

Back-end tunnel

A back-end tunnel acts like a split handshake except for one important difference: in a back-end tunnel, the remote access server holds its own SSL certificate so that the gateway may act as a true gateway (simply transporting TCP packets) without having to be involved in the SSL handshake at all. In short, it allows the remote access server to create a truly secure tunnel through the gateway to the browser.

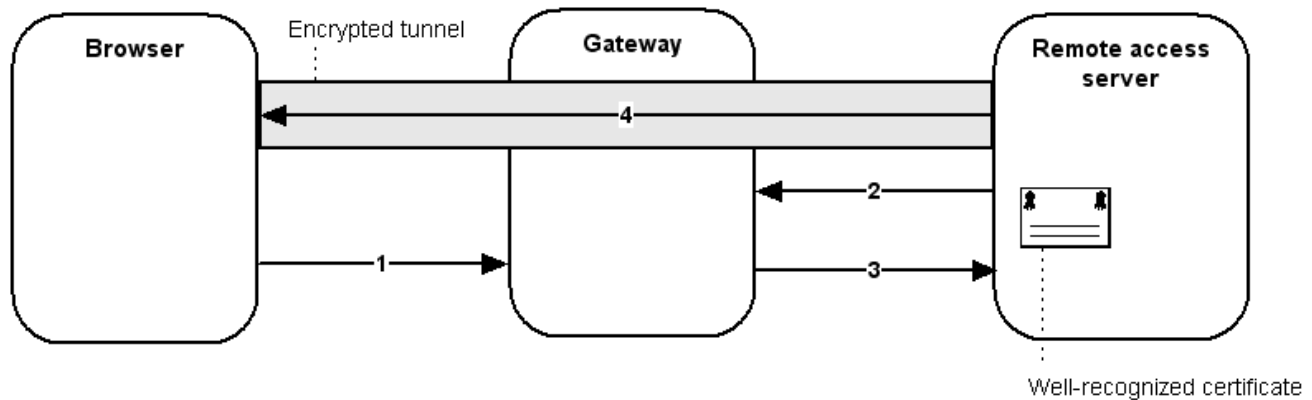
It is critical that each remote access server has a unique certificate generated specifically for it alone. This is in contrast to an unsafe system where a service provider may generate a single certificate and install it (along with the corresponding private key) on all remote access servers. It is unsafe because the private key is widely available and any "secure" communication between the gateway and the remote access server is decipherable by anyone with this key.

A back-end tunnel works as follows:

1. The browser connects to the gateway with the intent of connecting to a remote access server.
2. The remote access server, as it periodically does, checks in with the gateway to see if there are any requests for connections to it.
3. The gateway informs the remote access server that there is a pending connection, and allows the remote access server to forward a port to it.
4. The gateway begins relaying TCP packets between the browser and the remote access server. This first thing that will happen is the SSL handshake will take place between the browser and remote access server. Note that other than relaying these packets between the two ends, the gateway has no other function in the handshake.

5. Once the handshake is complete, an encrypted tunnel exists from browser to remote access server through the gateway.

Figure 3 - Data flow for back-end tunnel



There is a significant difference between this method and the split handshake: the gateway does not have access to any intermediate handshake data, as it does in a split handshake. While all data, even the handshake, flows through the gateway, the gateway **cannot** decrypt any of it because the SSL certificate comes from the remote access server.

Pseudo-SSL

In a normal SSL situation, when a client and server want to communicate over SSL, they perform the SSL handshake. This sets up an encrypted channel over which they communicate. However, normal communication does not immediately begin: instead, the client and server use this encrypted channel to create yet another secret key. This secret key is different from the previous secret keys of the handshake because it is for a *symmetric cipher*. A symmetric cipher is a single key used for both encryption **and** decryption of data. This is in contrast to the *asymmetric ciphers* encountered previously which use a private/public key pair. A symmetric cipher is ultimately used in SSL because it is faster and far more efficient than an asymmetric cipher.

The pseudo-SSL method creates secure communication by immediately creating a channel based on a symmetric cipher (just like regular SSL) without going through the SSL handshake. The challenge of this is in key distribution, i.e., getting the same key to both ends of the channel without it being intercepted by someone listening in on the transaction.

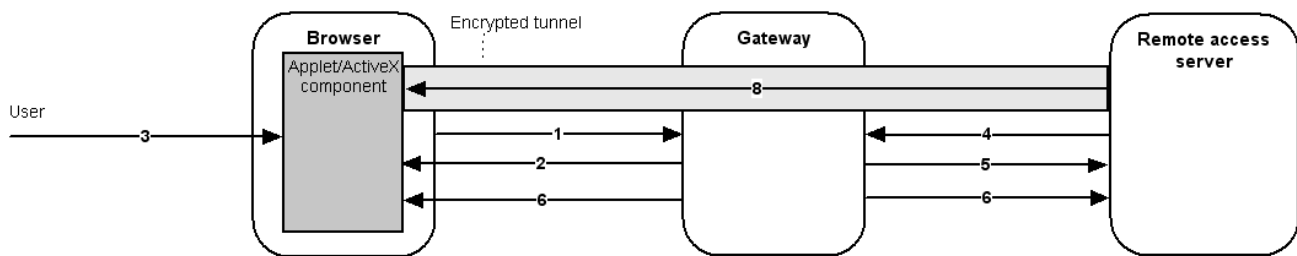
The pseudo-SSL method cleverly solves this problem: when a remote access server is installed and set up, it prompts the customer for a password. This password is never transmitted anywhere, and it is used only to calculate a symmetric key. If this password can independently calculate the same symmetric key on both sides of the channel, it solves the key distribution problem. Unfortunately, there is no mechanism within any standard browser to do this, so a browser plug-in, such as a Java applet or an ActiveX component, is required.

The entire system works as follows:

1. The browser connects to the gateway with the intent of connecting to a remote access server.
2. A Java applet or an ActiveX component is loaded into the browser.
3. The user types in the password into this plug-in component. This password is the same one as the one known to the remote access server.
4. The remote access server periodically checks in with the gateway to see if there are any requests for connections to it.

5. The gateway informs the remote access server that there is a pending connection, and allows the remote access server to forward a port to it.
6. The gateway generates a large random number, and transmits it to both the component and the remote access server.
7. The component and the remote access server independently use the random number and the password to create a symmetric key.
8. The gateway begins relaying TCP packets between the browser and the remote access server. The symmetric key encrypts the relayed data.

Figure 4 - Data flow for pseudo-SSL



This system looks very secure, but is it? There are several problematic issues:

- The method for independently calculating a symmetric key is not generally disclosed by companies using pseudo-SSL. This means that it probably has not been extensively reviewed by experts. Without such review, it is possible that the methods in use are not at all secure, and it is only a matter of time until someone figures out a security gap and exploits it.
- Because of the Java applet or ActiveX component, it is very difficult for the end user to know exactly what methods are used to secure his communication. Despite the claims in the product literature, there is no way to independently verify any promises of real security.
- This method tries very hard to do exactly what SSL does properly. However, a benefit of SSL is the end user's ability to independently verify it, by viewing the certificate; pseudo-SSL does not share this advantage. This suggests that the designers would have been better off by choosing a proper SSL implementation.

Proprietary methods

There are other methods that can create a secure link between a browser and remote access server. For instance, a special system for encryption can be invented to secure this link. In this case, or if there is a proprietary system where the service provider refuses to reveal how the system works, the system should be approached with great skepticism and caution. SSL is a highly trusted system because:

- Extensive technical documentation is available explaining how it works.
- Many experts in the field have tested it, looking for weaknesses, without finding any.
- Users can independently verify which security methods are in place since SSL support is built into the browser.

The history of cryptography is littered with systems that have been developed under the veil of secrecy, declared "100% secure" by their developers, and then broken in short periods of time.

One must understand an entire security process before believing that it is indeed secure. All too often, companies are eager to declare that they have end-to-end encryption and security because they rely on SSL. The magical incantation of "SSL" is not in itself meaningful: it must be evaluated in the context of an entire

system that includes the system architecture, the parts of the system that must be trusted, and how secret keys are created and distributed. Only then can we confidently declare that a system is secure.

It is also important not to fall into the trap of “security through association.” A service provider may provide a list of companies that use its product in the following belief: since **its customers** believe the system is secure, then it must be. True security happens only when there is complete disclosure of the methods used, and experts are able to evaluate and verify them.

Failures of proprietary methods

History has shown that full disclosure is the best way to ensure a secure system. This is evident from the multitude of “secure” systems that were developed in private, kept secret, and then broken because either a secret was leaked, or people discovered ways of defeating the “security.” Examples of “secure” systems that have been broken because they did not heed these warnings include:

- The Content Scrambling System (CSS) was designed to prevent DVDs from being copied, thus protecting movie studios assets. It was quickly broken by hackers who reverse engineered a DVD playback program to discover a poorly guarded secret, which allowed them to find the system’s “master key”. This allowed them to create their own programs to copy DVDs freely. Today, the movie studios are in a fight to protect their assets against piracy because the security system that they designed for protection was so flawed.³
- The Wired Equivalent Privacy (WEP) system used to secure 802.11b wireless networks is based on a system that has been proven insecure. An attacker who can watch traffic transmitted between two points on a wireless network can eventually figure out the secret keys used to secure the transmission and recover all transmitted data. If the designers of WEP had enlisted more experts in cryptography and gone through more reviews of their system, they would not have a system that is now considered insecure.⁴
- Researchers at The Johns Hopkins University were able to view the source code for the Diebold AccuVote-TS DRE voting system after it was accidentally leaked onto the Internet. They discovered a system that had multiple vulnerabilities so severe that anyone with a basic understanding of how they worked could tamper with the machines and manipulate voting results. It was very clear to the researchers that the security systems for these machines were very poorly designed and inadequately reviewed, if at all. The potential ramifications of exploiting such a security failure are enormous, as these machines were used in the presidential election of 2004.⁵

These cases highlight the problems that can occur when dealing with a security system that does not follow the common wisdom. Companies relying on these systems will inevitably lose more than they gain.

Problems

Of the five different methods for securing gateways that are described here, only the back-end tunnel, which uses SSL completely from end-to-end and in the manner in which it was designed, should be considered secure. The other four methods (two-stage SSL, split handshake, pseudo-SSL, and any proprietary methods) should be considered “lower-security methods” since each method has at least one exploitable flaw. This section describes flaws and how they can be exploited.

Which server is authenticated?

When a browser connects to an SSL-enabled server, the browser has information about the security aspects of the connection. For instance, when SSL is in effect, most browsers display an icon (often a little padlock that closes – see Figure 5) to show the user that the connection is secure. If the user clicks this icon, then information is displayed, usually from the certificate securing the connection (see Figure 6). The user can then see information about the server with which the browser is communicating. In systems that use the lower-security methods, the certificate is presented about the gateway server and not the remote access server. This can confuse the user who wants an end-to-end encrypted link between the browser and the remote access server. However, the lower-security methods do not enable the user to verify this.

Figure 5 – Padlock icon in Internet Explorer indicating that SSL is in use

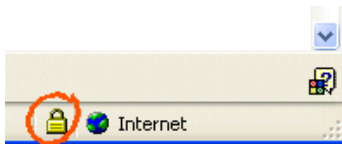
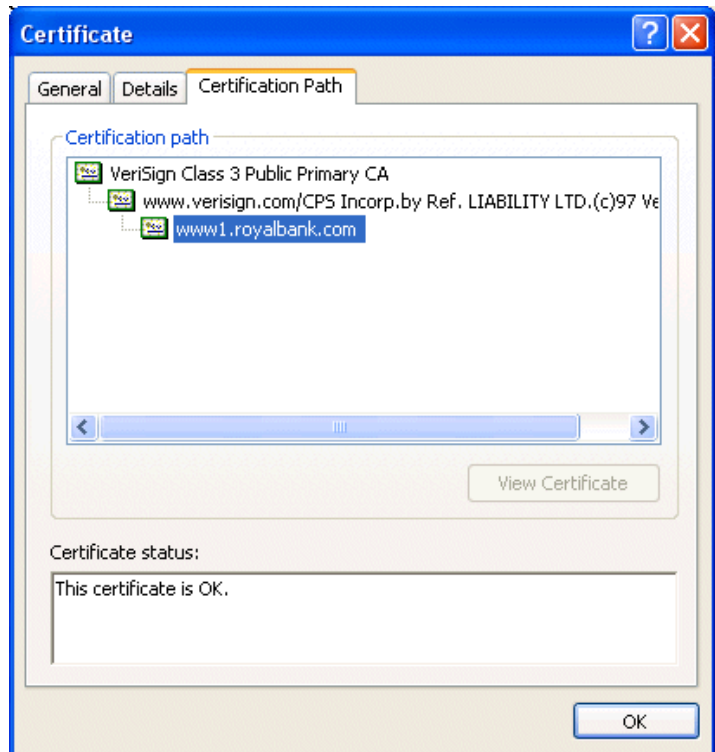


Figure 6 - A bank's certificate chain in Internet Explorer. Note that you can also view other certificate attributes.



Is anyone listening in?

When people access their data on a remote access server, there is an implicit assumption that this data is private and invulnerable to eavesdropping. In particular, there should be no way to launch a Man-In-The-Middle attack against such a system, and SSL is a defense that was designed to resist such an attack.

In a gateway-connected environment, there is a built-in entity between the two points communicating – the service provider running the gateway. Therefore, it is very important to ensure that the service provider cannot launch a Man-In-The-Middle attack. However, the lower-security methods cannot ensure this. In fact, it is quite easy for anyone with access to a gateway to launch such an attack, and the only defense against it is the service provider's own internal security system.

Does the gateway work as advertised?

This may be the most disturbing issue of all: whatever the products claim to do, all the lower-security methods **look the same** to the person sitting at the browser end of a connection. There is no independent way to verify which method is currently in place. This means the user must trust the service provider to honestly disclose the method in use. The service provider must also be trusted to properly secure the gateway from both external **and** internal threats. The latter is particularly difficult, since many employees of the service provider ultimately have access to the computer on which the gateway runs, and they can modify it so that private data is fully readable. Note that if this occurs, it is impossible for users to detect it.

Service providers will undoubtedly claim that their systems have security precautions, and that their employees are trustworthy. However, there have been recent cases of employees misusing their company's private customer information. For instance, Jason Smathers pleaded guilty on February 7, 2005 of violating the CAN-SPAM act which intends to limit the amount of SPAM e-mail that people receive.⁶ Mr. Smathers was a computer engineer at AOL who used his insider's knowledge of that system, as well as his privileged access to it, to misappropriate 92 million screen names of AOL subscribers and sell them to a company that sent SPAM e-mails. AOL, along with its customers, was clearly damaged by this action, even though it most certainly had safeguards built into the system to try to prevent this from happening. Preventing an insider with access to privileged systems from exploiting them is very difficult, and we can certainly expect more such cases in the future. Any service provider claiming that its SSL-based security system is strong because the company and its employees are trustworthy is badly mistaken. The proper use of SSL **should not** require end users to extend any additional trust to the service provider.

When dealing with the Soviet Union, Ronald Reagan used to say "Trust, but verify". This precept is also relevant when dealing with SSL. A service provider with an SSL-based solution for remote access servers ultimately expects end users to trust it. However, the service provider should not expect users to trust its solution without enabling them to verify that it deserves their trust.

Why do these problems exist?

Given that there is a way to improve the security of these systems, why do some companies still offer solutions based on these lower-security methods?

- **Cost-effectiveness** – The correct way to make these systems more secure is to ensure that each remote access server has its own well recognized SSL certificate. If this were the case, then anyone at the browser end of the transaction would immediately see that there is true end-to-end encryption. However, such an SSL certificate typically costs over \$100 per year, which makes the incremental value of providing this service not cost-effective for the end user.
- **An uninformed marketplace** – Unfortunately, computer security is very complex, and very few people have sufficient knowledge to separate the sense from the nonsense. Many marketers of anything related to security know they only need to mention "128-bit Security" and most people will assume it must be secure without looking any further. While the vulnerabilities outlined in this document are real,

they are very subtle: most computer users are unable to understand them, let alone find them, without more information. As people become more educated in the complexities of security, this will change.

- **Who really loses?** – It may seem cynical, but if a security vulnerability is exploited, it is the customer who ultimately loses private data, not the service provider who failed to secure the system properly. If a security flaw is exploited, it is almost impossible for users to detect, so the service provider may not even acknowledge it.
- **Hubris** – Almost all security development companies feel that their system's design is correct beyond question and thus invulnerable---until it is compromised. Unless we constantly reevaluate the systems that we build, find their flaws, and resolve to fix them properly, we significantly increase the chances that these systems will indeed be compromised.

Conclusion

Lessons from the past

Clipper chip

In April, 1993, the Clinton administration announced the Clipper chip, which was an initiative to provide encrypted communication.⁷ A similar chip, called Capstone, used the same algorithms to encrypt data. The United States government attempted to mandate the use of these chips for any application in which encryption was warranted. For instance, if a company wanted to market a telephone that could encrypt the audio so that it would be immune to wiretaps, then Clipper would have to provide the encryption. However, there were several problems with Clipper:

- The underlying algorithm called Skipjack was developed by the NSA and was classified on the grounds of national security. This meant that there was no independent validation that the algorithms were as strong as the NSA claimed, or even that they were secure at all.
- The biggest problem with Clipper, though, was the existence of a “master key” that would allow anyone who possessed it to decrypt anything that Clipper encrypted. This key was to be held by two “escrow agents” of the government to be used in assisting law enforcement. While eventually there were published procedures for how and when these keys were to be used, there was always great concern for how to safeguard the privacy rights of individuals from the threat of the irresponsible use of this power.

Because of these issues, there was a huge backlash against Clipper from both individual citizens as well as online rights advocates, such as Computer Professionals for Social Responsibility (CPSR) and the Electronic Frontier Foundation (EFF). Ultimately, the Clipper chip initiative failed miserably as people began to realize that any device that contained a Clipper chip would be unmarketable.

It's easy to look back and determine why Clipper failed. When it comes to privacy, people want to hear: “The system is secure – we **CAN'T** look at your data.” However, with Clipper, the best its developers could honestly say was “The system is secure – we **WON'T** look at your data.” This is not just a silly game of semantics: it is a fundamental difference against which people rebelled en masse when the facts were laid out in front of them. We must learn from this lesson.

Parallels to Clipper

There are several parallels between the way that many service providers secure their remote access server products and the Clipper chip. While both claim to offer their users real privacy and security, in fact, they can only do this with a caveat: the government could listen to any Clipper-secured transaction just as the service provider can listen to any transaction that flows through its lower-security gateway. Whether a service provider is doing this at any given time is immaterial. The fact is that the service provider can do it, and just like in the Clipper case, the customer is unable to determine if it is happening and is powerless to stop it.

Turning “We won't look” into “We can't look”

Products enabling easy access to data over the Internet are entering the marketplace. This is based on people's desire for this functionality and their ability to use these products because of the widespread availability of always-on broadband connections. Because the shared data may be private, companies providing these products must ensure that they meet the highest standards of privacy and security. In time, companies will find that they must provide stronger security as the marketplace becomes more knowledgeable. When this happens, the companies now claiming “We won't look” will need to adopt technologies that will let them say “We can't look”.

Endnotes

-
- ¹ D. Barrett, R. Silverman. *SSH, The Secure Shell: The Definitive Guide*. O'Reilly, February, 2001.
- ² E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley Professional, October, 2000.
- ³ D. Hamilton. *Banned Code Lives in Poetry and Song*, The Wall Street Journal, April 12, 2001.
- ⁴ A. Stubblefield, J. Ioannidis, and A. Rubin. "Using the Fluhrer, Mantin, and Shamir Attack to Break WEP", *Proc. ISOC Symposium on Network and Distributed System Security*, San Diego, California (February, 2002).
- ⁵ T. Kohno, A. Stubblefield, A. Rubin, and D. Wallach. "Analysis of an Electronic Voting System", *IEEE Symposium on Security and Privacy* (May, 2004).
- ⁶ "Ex-AOL Worker Pleads Guilty in Spam Case". *InformationWeek*. February, 7, 2005.
- ⁷ The White House Office of the Press Secretary. April 16, 1993.
(http://www.epic.org/crypto/clipper/white_house_statement_4_93.html)