brown10.qxd 7/5/00 11:38 AM Page 435

Chapter 10

IIS

As you well know, not everyone on the Internet runs Windows. There are many other platforms that can be reached across the Internet that have absolutely no idea how to send or receive DCOM packets. Because of the vast commercial enterprises farming the Internet using HTTP, and the vast numbers of consumers who want to jump on the fabled information superhighway, an incredible amount of research, development, and most importantly, standardization has gone into hardware and software for making HTTP sing on just about any platform you can imagine. Most pagers these days can access the Web. There are refrigerators in the works that will automatically order groceries over the Internet when your supplies are running low. If you want to tap this well, you have to buy into HTTP and Web server technology, and if you want to secure your transactions, you have to learn ways of authenticating in the face of firewalls and an incredible variety of client platforms.

brown10.qxd 7/5/00 11:38 AM Page 436

This chapter is all about putting a Web front end on your COM+ distributed application. You'll find that there is virtually no code in this chapter, because the job is mainly an administrative task (I assume you already know how to program ISAPI, ASP, or CGI apps). If this chapter does its job, by the time you're done reading it you'll feel more comfortable working with Internet Information Server (IIS) because you'll have a more clear understanding of the security context that your application will be running in given any particular configuration. (There are so many different configuration options that it's easy to become confused, and this often leads to a general sense of unease.) The chapter begins by explaining how authentication works on the Web using public key cryptography and certificates, then works through the various client authentication options, and ends by providing some tips to help you use IIS as a gateway into a COM+ application.

Authentication on the Web

When you go online and plan a trip, you typically send sensitive information across the wire, such as information about where you're going to be and special needs that you might have; often you'll even send your credit card number across the wire to purchase tickets or reserve a room or a rental car. It's amazing to me that so many people are willing to do this without understanding how their conversations are secured. Most consumers who care about security look for the little key in Netscape's Communicator or the little padlock in Microsoft's Internet Explorer (IE) and feel comfortable that these conversations are being encrypted so that a bad guy cannot see this sensitive information. But what's the point of encrypting a message if you don't know who you're sending the message to? How do you know that it's really Amazon.com on the other end of the wire and not a bad guy? I'm pretty confident that Jeff Bezos doesn't have lunch with each and every Amazon customer and exchange a secret passphrase that can later be used to generate session keys. What we need is some form of authentication that scales to the global Internet.

Chapter 7 described in reasonable detail a couple of authentication protocols that can be used to prove the identity of one principal to another electronically: NTLM and Kerberos. Can we apply either of these technologies to this particular problem?

brown10.qxd 7/5/00 11:38 AM Page 437

Consider NTLM. NTLM is all about proving the identity of the client to the server, which is usually satisfactory in a controlled environment such as a single business. The client is never cryptographically assured of the server's identity, but it's also much harder for internal servers to be spoofed than it is on the Internet at large. On the Web, where compromise of a router is a much more likely scenario, the focus reverses and protecting the consumer from spoofed servers is of critical importance. E-commerce sites typically still need to have some form of client authentication, but often this is as simple as correlating a credit card number with a billing address. Client authentication is pushed off to the credit card authority. Even if we were to reverse NTLM so that the client challenges the server, this requires the server to have a shared secret with each client (or that client's authority), which is pretty much equivalent to having the CEO of Amazon.com whisper a shared passphrase in your ear at lunch.

What about Kerberos? Kerberos has specific provisions for mutual authentication. The client can in fact verify the identity of the server during the authentication handshake, and any information that the client encrypts with the resulting session key and sends to the server is only useful to the one true server with which the client originally authenticated, because that server also knows the session key and can decrypt the incoming message.

When it comes to scalability, Kerberos is certainly a move in the right direction; by issuing tickets that have an expiration time on the order of ten hours, we reduce the load on the authority significantly, as opposed to a protocol such as NTLM, where the authority must be contacted for each authentication request.

So why doesn't Amazon.com use Kerberos to prove its identity to its clients? Well, when used in the conventional way, Kerberos requires the client to prove his or her identity to the server. Having the server prove its identity to the client is a nifty optional feature of Kerberos, but this doesn't help Amazon.com, who would now be forced to cryptographically authenticate each and every client, which once again puts Amazon.com in the business of whispering secrets to clients on lunch breaks.

However, what if we were to use Kerberos in reverse? Perhaps the server could present a ticket to the client to prove the server's identity, as opposed to

the other way around. Figure 10.1 shows how this might work. The client makes a request to the server, and the server sends a ticket plus an authenticator to the client to verify its identity. The ticket contains the server's name, an expiration date, and a key that the client and server can use to secure their conversations. This information was encrypted by the authority (not the server), so as long as the client trusts that authority, she'll trust that the ticket was not forged. If the client can decrypt this information (checking the authenticator to make sure the server really sent this message and that it wasn't stolen or replayed), she can happily use the key in the ticket to send confidential messages to the server.

Look at all the good ideas we've been able to leverage from Kerberos: Kerberos tickets *do* contain a key that can be used to establish an encrypted session, and they *do* contain an expiration date that helps make the authentication protocol scale better than something like NTLM, and they *do* have the notion of ownership (the server's name is in the ticket and anyone who purports to be the server must prove knowledge of the secret key associated with the ticket). The glaring problem is that the ticket also has a *fixed target* (in our case, the target is the client). If Amazon.com wants to prove its identity to Alice, it must obtain a ticket targeted at Alice. If Amazon.com wants to prove its identity to Mary, it must obtain a ticket targeted at Mary. This means that *everyone*

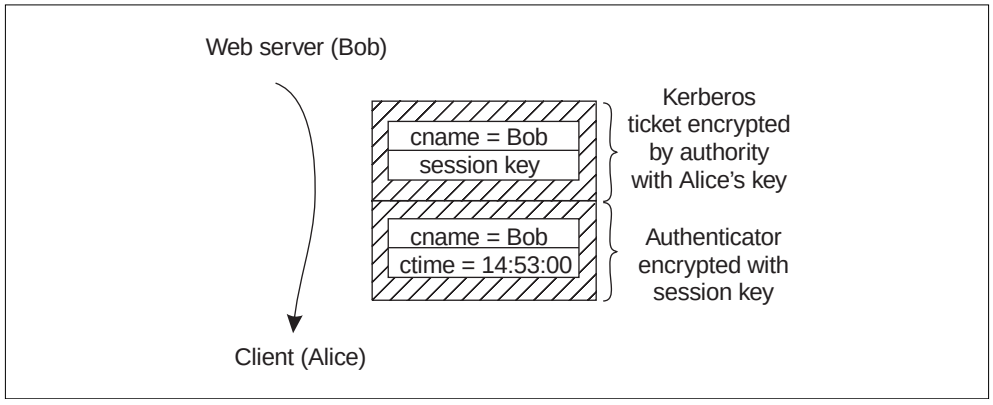


Figure 10.1 Kerberos in reverse?

brown10.qxd 7/5/00 11:38 AM Page 439

must register with a Kerberos authority, even clients. A client cannot simply be an anonymous Internet user and have any chance of verifying a server's identity using this scheme.

Why does this limitation exist? Because the Kerberos KDC relies entirely on conventional cryptography to prove the origin of its tickets. Using the reverse-Kerberos scheme that I've concocted, when Alice receives a ticket from Amazon.com, that ticket is encrypted with a secret key shared by Alice and the authority. The only reason Alice trusts the contents of the ticket is that she can decrypt it successfully, and thus she knows it came from her authority. It's not feasible at all to have Amazon.com (or its Kerberos authority) register as a principal with every client's authority on the Internet in order to weave a chain of secret keys across the Web. It just won't scale (once again, we've resorted to whispering secrets).

If, on the other hand, there were some way for an authority to encrypt a ticket with a special private key in such a way that *anyone* could decrypt it using a *different* key that was not a secret (but instead was a well-known value), we'd be just about half-way to a solution. When Alice receives this ticket from Amazon.com, if she can decrypt it with the well-known key for the authority, she will trust that the contents were really produced by that authority.

This doesn't completely solve the problem, though. If anyone in the world can decrypt the ticket with the authority's well-known key, this also means that anyone can see the secret key inside that Alice will use to encrypt data that she sends to Amazon.com. These sorts of tickets *cannot hold secrets*. So instead of a secret, the authority can put a well-known key for Amazon.com in the ticket. Similar to the key the authority used to encrypt the ticket, anyone can use this well-known key to encrypt a message, but only Amazon.com knows the corresponding private key to decrypt that message. In this scenario, the ticket doesn't need to be encrypted at all because it holds no secrets. Instead, the authority could simply *sign* the ticket using the special private key I mentioned earlier. Anyone in the world who trusts that authority could then verify this signature using the authority's well-known key.

This seemingly bizarre idea for a cryptosystem in which two paired keys are used (one public and one private) was invented in the mid-1970s by Whitfield

brown10.qxd 7/5/00 11:38 AM Page 440

Diffie,[1] and it's known as **public key cryptography.** The tickets we're talking about now are no longer Kerberos tickets, but rather **digital certificates.**

Public Key Cryptography

Without going into the mathematics involved in making it work,[2] the idea behind public key cryptography is quite simple. Instead of having a single secret key that can be used for encryption and decryption, the key is split into two parts: a **public key** and a **private key.** Only a single entity knows the private key, but the public key is just that—public. Most public key cryptosystems work in the following way: If you encrypt some **plaintext** with key A, you can only decrypt the resulting **ciphertext** with key B (see Figure 10.2). Because two different keys must be used for encryption and decryption, public key algorithms are also known as **asymmetric algorithms,** whereas conventional cryptosystems that use a single key are known as **symmetric algorithms.**

Here's an example of a public key algorithm. In DSA (the Digital Signature Algorithm[3]), key A is a private key and key B is the corresponding public key. This means that only one person can encrypt the plaintext into ciphertext,

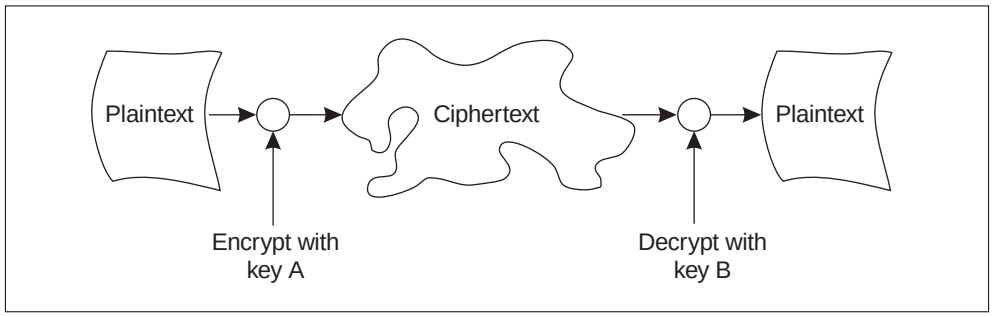


Figure 10.2 Public key cryptography

[1] As far as us civilians know, at least. It's entirely possible that this technology had been discovered years before by a major government; this is one sort of discovery that a government would want to keep to itself.

[2] See Schneier (1996) for an approachable introduction to the math.

[3] Part of the DSS (Digital Signature Standard) specification.

brown10.qxd 7/5/00 11:38 AM Page 441

but many people can decrypt it. Clearly this method cannot be used to send secrets, because anyone with the public key can decrypt the ciphertext, but this sort of mechanism is exactly what is required for signatures. By calculating a one-way hash of some plaintext and encrypting that hash with a private key, anyone who knows the corresponding public key can verify the signature. After verifying a digital signature of this type, you know that the plaintext wasn't tampered with since it was originally signed, and you know that the only entity that could have created the signature was the one who knows the associated private key.

By far the most well-known digital signature algorithm is RSA, named after its inventors, Rivest, Shamir, and Adleman. This algorithm can be used to create digital signatures as with DSA, but it can also be used to send secrets. In this mode, we reverse the way the keys are used so that anyone who knows the public key can encrypt a block of plaintext, but only the holder of the private key can decrypt the resulting ciphertext. What's convenient about RSA is that it works both ways: The same algorithm can be used to encrypt secrets as well as create signatures by reversing the way the keys are used (see Figure 10.3).[4]

One thing that stands out about asymmetric algorithms is that although they are great for producing and verifying signatures for which only a hash value needs to be encrypted or decrypted (a hash value is typically between 128 and 256 bits of data), they are really poor performers for encrypting bulk data. Symmetric algorithms are hundreds of times faster at bulk encryption. In practice, if Alice wants to send an encrypted message to Bob leveraging his public key, she can do something as simple as generating a random *conventional* key and sending it to Bob encrypted with his public key. She can then send Bob as much data as she likes, and encrypt it using a symmetric algorithm. Thus in practice, public keys are used for two different purposes: generating digital signatures and exchanging symmetric keys (also known as session keys).

[4] Although it's a bad idea to use the same key pair to do both; usually one key pair is used for signatures and another is used for encryption (Schneier 1996).

brown10.qxd 7/5/00 11:38 AM Page 442

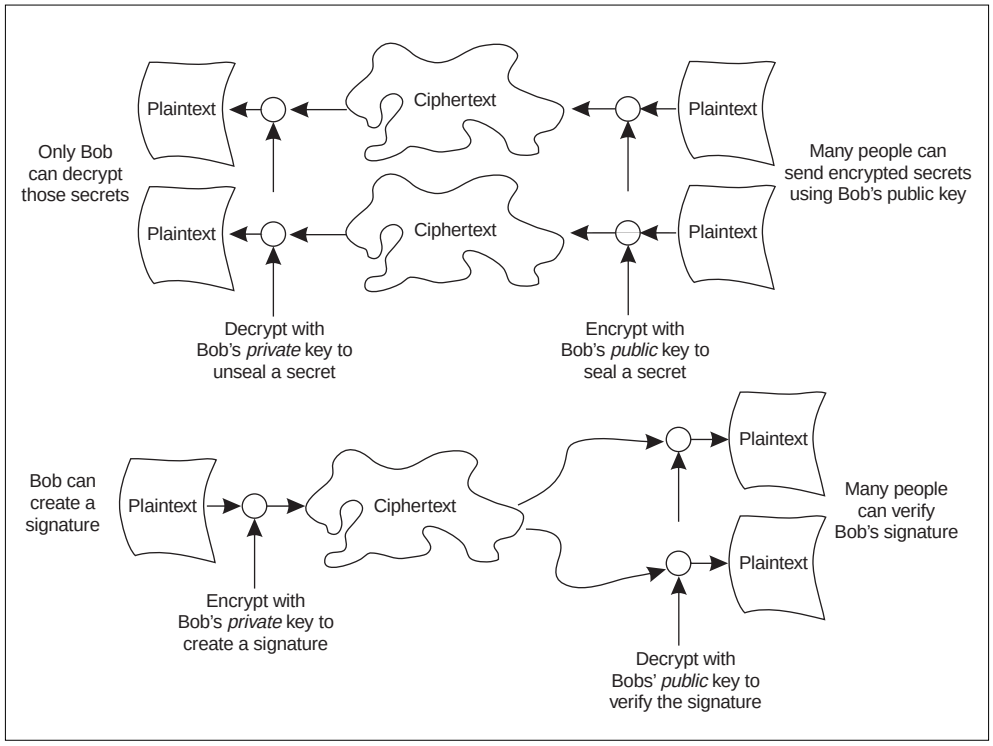


Figure 10.3 RSA encryption and signature generation

Certificates

I remember that when I first learned about public key cryptography, I initially thought it was the silver bullet that would solve all key exchange problems. What I quickly realized, however, was that when used properly it can lead to a more scalable cryptosystem, but that key exchange is still hard.

With a conventional cryptosystem, all keys are secret keys. When the KDC constructs a Kerberos ticket and embeds a session key inside, the contents of that ticket must be carefully encrypted so that a bad guy cannot discover the embedded session key. This also means that if a bad guy were to tamper with the ticket, in an attempt to change the session key to a value he or she knows, the server receiving the ticket would detect this because the ticket would not

decrypt properly. (The results would be a garbled mess, and Kerberos implementations watch for this sort of funny business.)

However, when you send your *public* key across the wire, it's tempting to think that it doesn't need any protection. Granted, it's not a secret, so you don't need to hide it as Kerberos hides its session keys inside tickets, but imagine what would happen if a bad guy were to tamper with the key in transit. If Bob sent his public key to Alice, and Fred intercepted that message and replaced Bob's public key with his own before sending the message on to Alice, any secrets that Alice subsequently sent to Bob using the compromised key would be readable by Fred. Granted, if Bob receives any of these messages *directly,* if he's paying any attention at all he'll see that they decrypt to complete gibberish (only Fred can successfully decrypt messages encrypted with his own public key), but if Fred has hijacked a router between Alice and Bob, it's all over. Fred can simply intercept each of Alice's encrypted messages, decrypt them, read them, modify them to his liking, and then encrypt them using Bob's real public key. Neither Alice nor Bob will have a clue that Fred is in the middle. If Bob also asks Alice for her public signature key, Fred can substitute his own key; now Fred will be able to sign messages to Bob, and Bob will be tricked into thinking that Alice was the signer (Figure 10.4 shows the scam). The crux of the problem is that in order for Alice or Bob to be able to safely use public keys

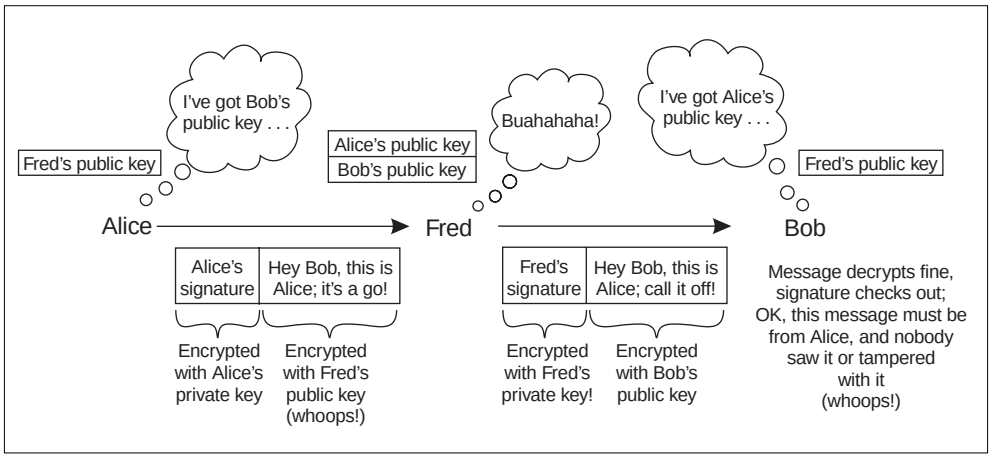


Figure 10.4 The "Fred in the middle" attack

brown10.qxd 7/5/00 11:38 AM Page 444

that they receive electronically, they must have some way to verify the identity of the person who owns the corresponding private key.

Can't Bob just sign the key he sends to Alice, so that Alice can verify that it really came from Bob? This is like asking which came first, the chicken or the egg. Alice can't verify any of Bob's signatures until she obtains his public signature key, and how will she ever verify *that* key? The point that I'm trying to drive home here is that secure key exchange is just plain hard, even with public keys (which are not secrets). Two popular solutions to the problem are as follows:

1. Exchange initial public keys with your friends (even face to face if necessary) so that Fred *can't* get in the middle. Then treat those friends as trusted authorities. This is the model used by PGP (Pretty Good Privacy).
2. Use a hierarchy of trusted authorities. This is the model used by X.509.

PGP and Cumulative Trust

Here's the idea behind PGP in a nutshell: Alice and Bob are friends, so they exchange public keys simply by sending them via email, but then they meet at lunch (or call each other on the phone) and authenticate those keys. The way this is done is quite simple. For instance, for Alice to verify Bob's public key, she calculates a hash of the key she received in the mail, and Bob takes a hash of the key he sent. Alice then reads the hash value aloud (either over the phone or over a ham sandwich). Now that they trust the validity of each other's keys, in the future, Alice and Bob can sign or seal packets that they send to one another. If Bob wants to introduce Alice to Mary, he can send Alice a signed message containing Mary's public key, and Alice, because she trusts Bob, adds this key to her "keyring" (presumably Bob met Mary face to face or obtained her key electronically from someone he trusts and with whom he had previously exchanged keys). This "web of trust" expands into a community of users who trust one another's public keys.[5]

[5] This is tremendously simplified, of course. See Zimmerman (1995) for more detail.

Each public key that Alice receives electronically from Bob comes wrapped in a tight little package called a *certificate.* The certificate contains a public key along with (typically) an email address identifying the owner of that public key, plus an expiration date; the contents are signed with Bob's private key. Bob in this case is the certifying authority. Because Alice trusts Bob, when she validates his signature she develops trust in Mary's public key.

X.509 and Hierarchical Trust

The X.509 model of trust asserts that there is a rigid hierarchy of authorities (your buddy can't simply act as an authority). In the degenerate case there is just one authority whose public key is well known. If Alice needs to obtain Bob's public key, she simply asks him for it electronically, and Bob sends Alice a certificate (see Figure 10.5) that contains a public key and an X.509 distinguished name, along with (among other things) an expiration date and the name of the authority that issued the certificate. The contents of the certificate are signed with the private key of the issuing authority, and since this authority is well known, Alice can verify Bob's certificate by simply using the well-known public key of the authority.

In the real world, there are several authorities whose public keys (contained in self-signed certificates) actually ship with Web browsers such as Communicator and IE. Many individual companies also maintain their own

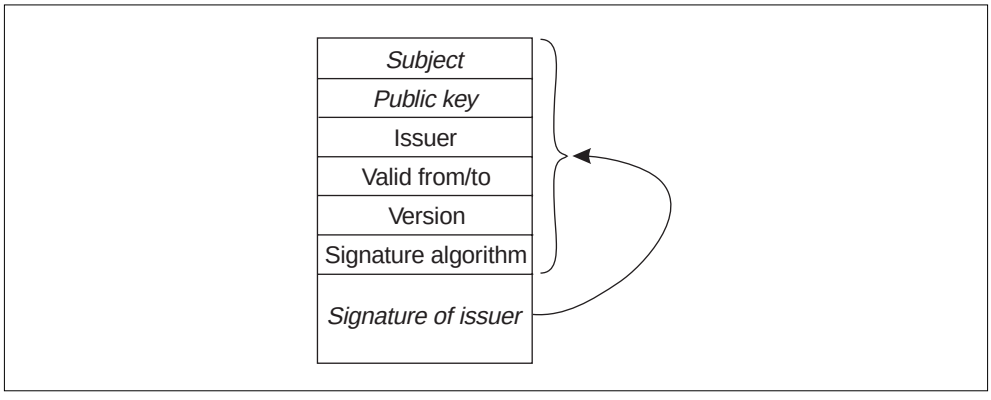


Figure 10.5 An X.509 certificate

certificate authorities (CAs) so that they can issue certificates internally; this works well as long as the certificates are only used within that particular company. In order to broaden the scope of trust for these internal certificates, it's possible for the company's certificate authority to be validated by one of the well-known authorities, forming a tree of trust (see Figure 10.6).

In Figure 10.6, the company Foo may choose to accept Bob's certificate that was issued by Bar, because Bar has been certified by an authority that Foo trusts. This manifests itself by a chain of certificates, starting with Bob's certificate, which is signed by Bar, and Bar's certificate, which is signed by Quux. Quux is a root authority and thus signs its own certificate; this is the sort of well-known certificate that gets distributed with software such as a Web browser. As long as Foo trusts one of the certificates in the chain (in this case, Quux), Foo develops trust in Bob's certificate. This is a simplified explanation; for more information on developing a **public key infrastructure** (**PKI**) such as this, see Adams (1999), Feghhi (1999), and Ford (1997).

Web servers use X.509 certificates to prove their authenticity to clients. Each client can obtain the Web server's certificate, ask it for proof of its identity, and traverse up the hierarchy of trust until a trusted certifying authority is found.

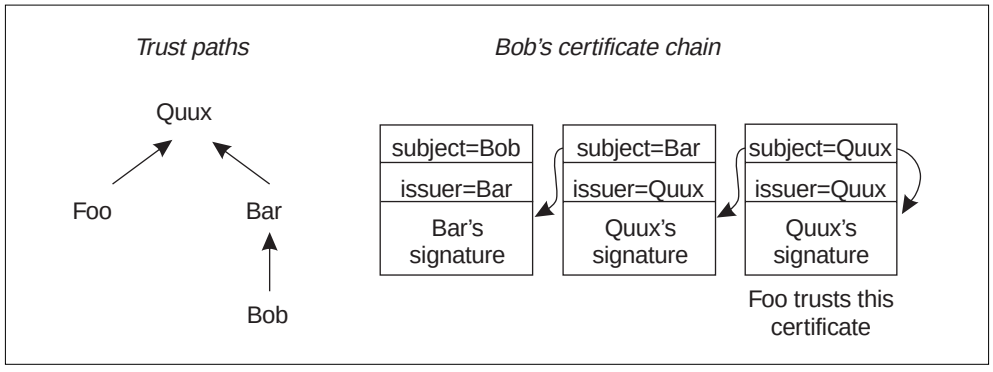


Figure 10.6 A hierarchy of trust and a resulting certificate chain

brown10.qxd 7/5/00 11:38 AM Page 447

Interlude: Some Acronyms and Terms

This section contains a rather terse description of some of the acronyms and terms that you'll commonly hear on the Web security playground. I wanted to provide this to give you some idea of the history of authentication on the Web; by putting these acronyms into context, I can focus on the dominant protocol.

SSL, PCT, and TLS, Oh My!

In 1994, Netscape Communications developed and popularized a network authentication protocol known as the Secure Sockets Layer (SSL 2.0). In 1995, Microsoft countered with a protocol known as Private Communication Technology (PCT 1.0), which was an improvement on SSL 2.0. Around the same time Netscape released an independent suite of improvements via the SSL 3.0 protocol, which dominates the Web as of this writing. SSL 3.0 was submitted to the IETF as an Internet draft ("Secure Sockets Layer 3.0 Specification") in 1996, and an IETF working group was formed to come up with a recommendation. In January 1999, RFC 2246 was issued by the IETF, documenting the result of this group's efforts: the Transport Layer Security protocol (TLS 1.0), which is virtually indistinguishable from SSL 3.0.[6]

After the world embraced SSL 3.0 and TLS, Microsoft gave up on PCT with a sigh (the following quote was taken from the November 1999 MSDN build):

> Developers are not encouraged to use PCT because it is Microsoft proprietary and has been completely superseded by SSL 3.0 and TLS.

No matter how much I'd like to use the term *TLS* throughout the rest of this chapter (because it's been standardized), the world still refers to the protocol as SSL; thus, because there is so little difference between TLS and SSL, I'll

[6] I searched long and hard to find the differences between SSL and TLS and finally came up with an Internet draft that specified the proposed modifications to SSL made by the authors of the TLS RFC. This draft has expired, of course, but in case you're interested, you can search the Web for draft-ietf-tls-ssl-mods-00.txt. The most important change mentioned in this document that made it into TLS is a revision to the MAC calculation algorithm, but other minor changes were also proposed, such as the addition of more detailed error codes and a change in the way clients who have no certificates respond to a server's request for a client certificate.

brown10.qxd 7/5/00 11:38 AM Page 448

cave in and refer to the protocol as *SSL* as well. The name change just confuses people.

SCHANNEL

You may have heard the term SCHANNEL (which stands for "secure channel"); this is the name of the SSP in Windows that implements all four of the authentication protocols discussed earlier: SSL 2.0, PCT 1.0, SSL 3.0, and TLS 1.0. SCHANNEL is a Windows-specific term that is often bandied about in MSDN documentation as an umbrella term for all these authentication protocols.

HTTPS

The Internet Assigned Numbers Authority (IANA) reserved port 443 for HTTP over SSL (although all the different flavors of SSL, including PCT and TLS, also use this port); HTTPS is the name of the URL scheme used with this port. Thus, http://www.develop.com implies the use of vanilla HTTP to port 80, and https://www.develop.com implies the use of HTTP over SSL to port 443.

Secure Sockets Layer

At the heart of SSL (a.k.a. TLS) is the *record protocol* that provides message framing, typing, and fragmentation, as well as compression, encryption, and MAC generation and verification.[7] SSL assumes that a connection-oriented transport is in use (TCP being the canonical example), and unless the message fragments are received in the correct order from the underlying communication protocol, the receiver won't be able to decrypt the stream (each fragment isn't guaranteed to be independently decryptable).

To encrypt or generate/verify MACs, obviously both endpoints need to share a secret key, otherwise known as a session key. SSL uses a higher-level protocol known as the *handshake protocol* on top of the record protocol to exchange this key and authenticate the client and server to one another. Authentication is technically optional, and there are three modes in which SSL can be used:[8]

[7] The concept of a MAC (message authentication code) was discussed in Chapter 7.

[8] Note that it's possible to negotiate a cipher suite with a NULL bulk data encryption algorithm, which allows authentication without encryption. Although Web browsers don't use this option, it's interesting to consider for other, more intelligent user agents.

1. Mutual authentication
2. Server-only authentication (the client knows who the server is)
3. No authentication (deprecated)

The third option is silly if you think about it. Exchanging sensitive data over an encrypted but unauthenticated link is like two spies sitting alone in a dark corner of an obscure restaurant, whispering secrets to one another without either of them having a clue who the other one is. The vast majority of commercial HTTPS traffic over the Web today uses option 2, wherein the client is anonymous (as far as SSL is concerned), but the server is authenticated.

SSL uses a four-way handshake in all three cases (see Figure 10.7); this discussion will focus on the elements of this handshake necessary for authen-

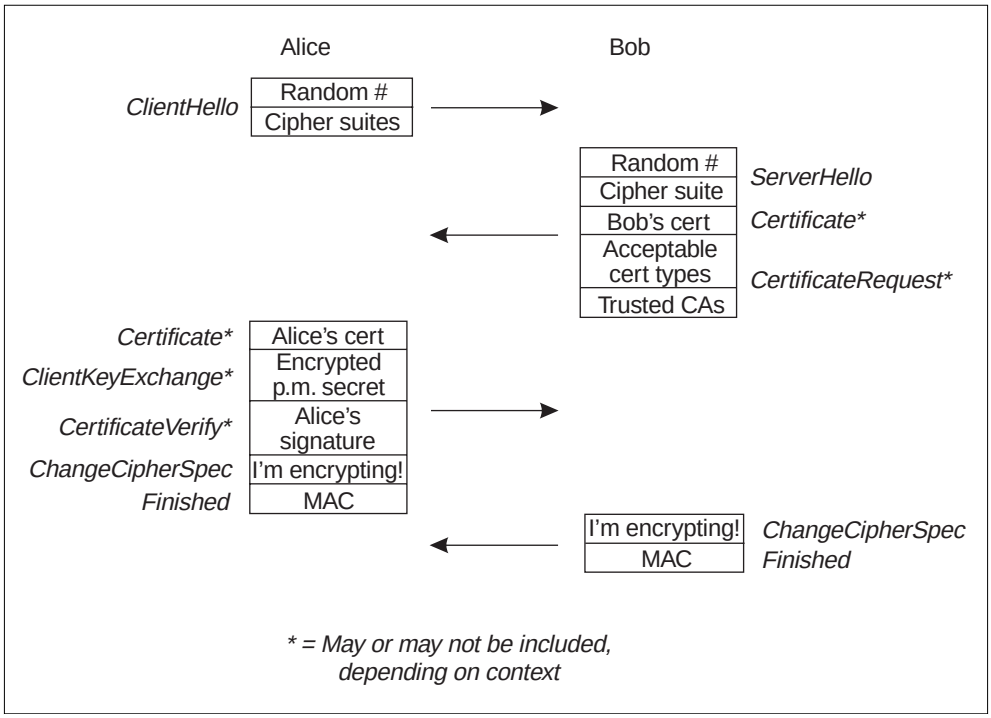


Figure 10.7 Establishing a new SSL session

brown10.qxd 7/5/00 11:38 AM Page 450

tication and key exchange.[9] The client (Alice) first sends a Client Hello message to the server (Bob) to indicate that she wants to establish a new SSL session. This message contains a random number generated by Alice, as well as an ordered set of preferred cipher suites (each cipher suite indicates a key exchange algorithm, a bulk encryption algorithm, and a MAC algorithm).

Bob looks at the incoming request, selects a cipher suite from Alice's proposed list (assuming one is acceptable), and sends a Server Hello message back to Alice. This message includes a random number generated independently by Bob, along with the cipher suite that Bob chose from Alice's list. If Bob chose a cipher suite whose key exchange algorithm requires him to prove his identity (only the deprecated option 3 does not), he'll send his X.509 certificate as well.[10] Depending on the key exchange algorithm, Bob might also need to include extra information to allow key exchange or satisfy U.S. export restrictions,[11] but to keep things simple, let's assume that the certificate Bob sends to Alice can also be used directly for key exchange. Finally, as long as Bob has provided his certificate, he is allowed to include a request for Alice's certificate, which forces Alice to prove her identity.

When Alice receives this information from Bob, she can verify Bob's certificate. This includes checking the signature of the root authority with her copy of the authority's well-known public key (certificate verification is revisited in the upcoming section Certificate Revocation). At this point, Alice knows that she has a public key for Bob, but she really doesn't have any proof that it's actually

[9] See RFC 2246 if you want more detail.

[10] Technically, this can be a list of certificates if Bob's certificate wasn't signed by an authority that Alice trusts. In practice, this means that Bob sends back a chain of certificates up to but not necessarily including the self-signed certificate for a well-known root authority such as VeriSign or Thawte, with which all Web browsers are already equipped. As of this writing, most of the Web sites we know and love (www.amazon.com, for example) have certificates that are signed directly by a well-known root authority such as VeriSign. This is actually one of the minor differences between SSL and TLS; in SSL, the server was always required to send the entire certificate chain, including the self-signed certificate for the root authority.

[11] This might include parameters for the Diffie-Hellman key exchange algorithm or an independent RSA public key no longer than 512 bits to satisfy U.S. export restrictions for key exchange. This allows strong authentication with weak encryption; the U.S. military is happy to allow Alice to know the name of the tall, dark, and foreign person she's having cybersex with, as long as they get to watch!

brown10.qxd 7/5/00 11:38 AM Page 451

Bob on the other end of the wire. No session key has been exchanged, but Alice is now going to remedy that.

Alice sends the following information to Bob: her certificate (if it was requested),[12] and another random number (known as the premaster secret) that has been encrypted with Bob's public key.[13] If Bob requested a certificate from Alice, she will also include her signature on all the data she's sent to or received from the record layer so far during the handshake.

Up until now, the SSL record layer has been streaming data using the NULL bulk data encryption algorithm; nothing has been encrypted. Alice now streams out a Change Cipher Spec message, which basically says, "Until I say otherwise, I'm going to instruct my record layer to use the cipher suite we just negotiated." This means that Alice needs to calculate keys for bulk data encryption and MAC generation/verification, both of which are ultimately just functions of the premaster secret and the two random numbers generated independently by the client and server during the first two messages. Finally, Alice streams out a Finished message, which is encrypted by the record layer. This message includes a MAC of all the data she's sent to or received from the record layer so far during the handshake.

When Bob receives Alice's transmission, he verifies her certificate (assuming that he asked her for one), obtains the premaster secret by decrypting it using his private key, and (once again, assuming he requested her certificate) verifies Alice's signature on the handshake messages he's seen so far using the certificate she sent. (He's now developed trust that it really is Alice on the other end of the wire.)

Bob now receives Alice's Change Cipher Spec message, calculates the keys for bulk data encryption and MAC generation/verification, and instructs his record layer to start using the new cipher suite to decrypt the incoming transmission. This allows Bob to read the Finished message, which is at the tail of

[12] When Alice sends her certificate to Bob, she also may send a chain of certificates, although in this case SSL provides a way for Bob to indicate to Alice the root certificate authorities that he trusts so that she doesn't have to guess.

[13] If Diffie-Hellman key exchange was negotiated, this information will instead be parameters for a Diffie-Hellman key exchange (assuming those parameters weren't already specified in the server's certificate).

brown10.qxd 7/5/00 11:38 AM Page 452

Alice's transmission. After verifying the MAC in the Finished message, Bob develops trust that the entire handshake and cipher suite negotiation wasn't tampered with, because this MAC protects the entire set of handshake messages exchanged so far.

Finally, Bob sends a Change Cipher Spec message back to Alice (instructing his record layer to use the new cipher suite to encrypt outgoing messages), followed by a Finished message (which once again includes a MAC of all messages exchanged so far).

Alice receives the Change Cipher Spec message, instructs her record layer to decrypt incoming messages using the negotiated cipher suite, and reads the Finished message from Bob. If the message decrypts successfully and she can verify the MAC Bob sends to her, she develops trust in Bob's identity (only Bob could have decrypted the premaster secret she sent, which was required for him to be able to generate the MAC).

At this point, Alice and Bob have negotiated a cipher suite, exchanged shared keys for bulk data encryption and message authentication, and Alice knows that she's talking to Bob. Bob may also (if he asked for a client certificate) know Alice's identity.

Certificate Revocation

One of the biggest potential traps of using certificates is the tendency for people to assume that a certificate is valid simply because its signature can be verified and it's not yet expired. In a password-based authentication system, if Bob's password is compromised, he can simply change his password. In NTLM the authority immediately enforces this change for all new authentication requests[14] because the authority is involved in every single network authentication exchange. In Kerberos, this change is usually enforced in something less than ten hours (after Bob's outstanding TGTs expire). So what happens in a certificate-based system if Bob's *private key* is compromised? Well, Bob notifies his authority and obtains a new certificate, and the authority "revokes" the old certificate.

[14] Barring replication latency between domain controllers.

brown10.qxd 7/5/00 11:38 AM Page 453

But what does this mean for Alice, who contacts a server that she presumes is run by Bob but really has been hijacked by Fred (who has an illegitimate copy of Bob's old private key)? Fred can simply send Bob's old certificate (which won't expire for a year or so) back to Alice and can prove ownership of the certificate because he now holds the associated private key. Unless Alice specifically checks for *revocation,* she'll never know that she's really talking to Fred. Once again, public key cryptography is not a silver bullet. Alice still needs to contact an authority to verify that Bob's certificate hasn't been revoked.

Bob's authority publishes a **certificate revocation list** (**CRL**) that Alice can obtain occasionally, perhaps once a day or once a week; she can use her current copy of the list to validate Bob's certificate. As an example, IE 5 has a security option entitled "Check for server certificate revocation" that forces the browser to download a CRL (or retrieve a freshly cached one) and verify that each server-side certificate hasn't been revoked. If you think about it, one reason that certificates have expiration dates is to keep CRLs from growing infinitely long.

From Theory to Practice: Obtaining and Installing a Web Server Certificate

Now that I've described how SSL authentication works, you'll find that it's mainly an administrative task to enable it in IIS. Recall that for SSL to be effective, the server must have a certificate so that at least one of the parties can be authenticated. If you look at the properties for a Web site in IIS, you'll notice on the Web Site tab that the entry for SSL Port is disabled by default. IIS will not support SSL without a server-side certificate.

Obtaining and installing a Web server certificate for an established company is very straightforward.[15] IIS provides a wizard that makes it easy. IIS allows you to have multiple Web sites exposed from a single machine, and each of these Web sites can have a certificate (or not); the point here is that you must set them up independently. Choose the Web site for which you want to obtain

[15] The following description assumes IIS 5.0; IIS 4.0 had a tool called Key Manager that was used to manage the acquisition and installation of server-side certificates. The procedure is very similar in any case.

brown10.qxd 7/5/00 11:38 AM Page 454

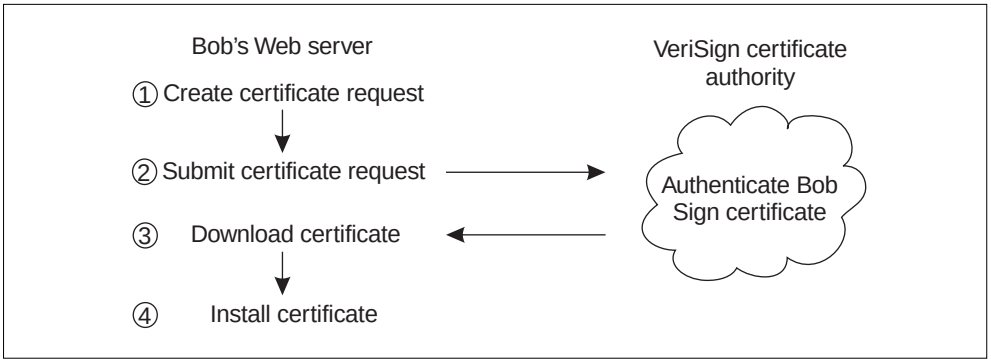


Figure 10.8 Requesting and installing a certificate

a certificate, bring up its property sheet, and choose the Directory Security tab. Press the Server Certificate button to invoke the wizard.

The wizard allows you to create certificate requests, install new certificates, remove an existing certificate, and so forth. Each of these activities requires answering a few straightforward questions. Unless you're planning on using your own enterprise certificate authority[16] to issue a certificate directly, obtaining and installing a certificate will be a four-step process (see Figure 10.8). The first step is to create the request. To do this, you need to choose the strength for the key in the certificate (you should use 1024 bits or greater), along with some strings that will end up in the certificate exactly as you type them in the wizard:

- **Name** Some friendly name that you can use to distinguish this certificate from any others you might obtain.[17]
- **Organization Name** The name of your company.
- **Organizational Unit** Typically the name of your department.
- **Common Name** This is the name that will be used to authenticate the URL being used to access the Web server, so it should match the

[16] When installing Certificate Services on a Windows 2000 domain controller, you can choose to have it integrate with Active Directory, becoming an "enterprise" certificate authority.

[17] This isn't important for authentication, but will be included as an extra property in the certificate.

DNS name that you expect clients to use (for instance, the common name for DevelopMentor's Web site is "www.develop.com").

- **Country/Region** Self-explanatory.
- **State/Province** You cannot use an abbreviation here (specify "Texas" instead of "TX").
- **City/Locality** No surprises here.

After this, the wizard asks you to choose the name of a text file where it will dump the request. By the time this file is created, the wizard will have made calls into the CryptoAPI to generate a public/private key pair for the certificate, storing the private key on the local machine. This public key will be included along with the other information you've entered in the text request file, which you can send to your certificate authority (for instance, VeriSign or Thawte[18]). The resulting file contains a base-64 ASCII rendering of all the information in the request:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIDBjCCAm8CAQAwcTERMA8GA1UEAxMIcXV1eC5jb20xDzANBgNVBAsTBkJyYWlu
czEWMBQGA1UEChMNRGV2ZWxvcE1lbnRvcjERMA8GA1UEBxMIVG9ycmFuY2UxEzAR
BgNVBAgTCkNhbGlmb3JuaWExCzAJBgNVBAYTAlVTMIGfMA0GCSqGSIb3DQEBAQUA
A4GNADCBiQKBgQDFUxFtzr170yxptKuGI1590Sta5z2dVElLfjAn+q4T1uZE3DiH
HXNRHW1eS9W2aeMZhRnYRi5U8eOdG3RUO4YXy4B1sqfy5I0qjjySA89ghVd/6JcA
K1nhGJL9FPJ6XKVUNLez7NpSCFlYs5foyTqyxDkHzTnQwRwkkwQ9dlbnfwIDAQAB
oIIBUzAaBgorBgEEAYI3DQIDMQwWCjUuMC4yMTk1LjIwNQYKKwYBBAGCNwIBDjEn
MCUwDgYDVR0PAQH/BAQDAgTwMBMGA1UdJQQMMAoGCCsGAQUFBwMBMIH9BgorBgEE
AYI3DQICMYHuMIHrAgEBHloATQBpAGMAcgBvAHMAbwBmAHQAIABSAFMAQQAgAFMA
QwBoAGEAbgBuAGUAbAAgAEMAcgB5AHAAdABvAGcAcgBhAHAAaABpAGMAIABQAHIA
bwB2AGkAZABlAHIDgYkAXxNuAz6gcBaZUdef8WQ2PAroKMW8sprcKv7QD2encz6/
Wct9DZ5CkGynLGy0f+Lff7ViSDJqxYWaJ68ddqgXyAqIilF63kivPTiC6yxLaNX6
5v3cnKFx4UrUrGXZtub7M7/NuxSipOW0Vv7yCHganypxDyRzp6IhulEnL4APEH4A
AAAAAAAAADANBgkqhkiG9w0BAQUFAAOBgQBljJb1ZhWOwOLfzfHbC3yxGkXDy9w3
NA7uhQOvgntnqmSmdHP9nsM3DnxwaHb3EVxMKbAuLsSRDAE1KGqeamvQ3uFjuuL0
5q4nKhX25LyGFDSc6h1OHcv+0ugZ/9klsiViSeEGpMwllUf057o7q1Vls4HN22vM
wkcejcttDjo3Kw==
-----END NEW CERTIFICATE REQUEST-----
```

[18] Thawte was actually acquired by VeriSign in December 1999.

brown10.qxd 7/5/00 11:38 AM Page 456

The second step is to send the request to the authority. You'll typically paste this into the authority's Web-based certificate application form, along with contact information, proof of domain ownership, and proof of your right to do business under the specified organization name. Third-party certificate authorities will charge you a fee for this service. (Fees vary depending on the level of service and security; as of this writing you can expect to pay approximately $100 to $1000 for a certificate that expires in a year.)

The third step is when the authority issues (or denies) your request. Assuming the authority can determine that you are who you say you are (by checking addresses, making phone calls, etc.), you'll receive email (typically within a few business days) indicating that your certificate request has been granted and that you can download your certificate, which will again take the form of a file (the contents will look very similar to the certificate request).

The fourth and final step is to install the downloaded certificate. If you now revisit the certificate wizard, it'll allow you to process the response; just give it the path to the file you downloaded from the authority and you should be off and running.

Note that during this request/response phase, the private key remains on the computer where you generated the request. Be aware that if you delete the "pending" request using the wizard (before you install the certificate), the private key will be erased and you'll have to start all over again. If you've already paid money to a third-party authority to sign your public key, this can be painful, so watch out.

Once you've installed the certificate, you can export the certificate *along with its corresponding private key* to a file that you can drop on a floppy and put in an offline vault in case the Web server crashes and the private key becomes unrecoverable. To do this, bring up the property page for the Web site, go to the Directory Security tab, and press the View Certificate button. Go to the Details tab on the resulting dialog, and choose Copy to File. If you choose to export the private key, you'll be asked for a password that will be used to encrypt the file. If you were really concerned about security, you could store the password in a *separate* vault, but to be honest, you should be more worried about someone simply compromising the Web server and stealing the private key from there. (Unfortunately, for a Web server, there's not much sense in

keeping the private key offline, because it is needed to establish each new HTTPS session.)

After installing a certificate, you'll notice that the SSL Port field on the Web Site tab of the Web site's property sheet is now enabled, and defaults to the standard port number for HTTPS, 443. You should now be able to access your Web site from a browser using the https: scheme.

What Is Server Gated Cryptography?

You may be wondering what the Server Gated Cryptography checkbox in the IIS cerificate wizard is all about. Prior to January 2000, United States export laws strictly prohibited the export of software that used strong encryption (128-bit SSL bulk data encryption keys fall under this category). Browsers built by Microsoft and Netscape were therefore subject to these laws and normally do not use 128-bit encryption unless the client specifically downloads an upgrade. Microsoft therefore proposed an extension to SSL called **Server Gated Cryptography** (**SGC**). SGC allows a server certificate to include a special annotation that indicates that the server has been approved to use strong encryption (certain types of businesses including banks and online merchants were excluded from the older export laws). If a browser that supports SGC detects this, it will negotiate the use of strong (128-bit) bulk data encryption with that server. As of this writing, VeriSign issues SGC certificates (this is part of their Global Site Services program), but they are considerably more expensive than a normal certificate.

However, now that U.S. export laws have been relaxed, SGC is not nearly as critical to global trade over the Internet. Clients can now download strong encryption packs for their browsers that work with or without SGC support on the server side. There are still some limitations, but they generally only apply to embargoed countries (see http://www.microsoft.com/exporting for more details).

Requiring HTTPS via the IIS Metabase

In IIS, once you've installed a server certificate, any virtual directory may be accessed via HTTP or HTTPS. If you want to *require* HTTPS for a particular resource, you can use the metabase to do so. If you're not already familiar with the metabase, it's simply a hierarchical data structure that mimics the layout of

brown10.qxd 7/5/00 11:38 AM Page 458

your Web site. The metabase tree is what you're looking at when you manage the Web site using the IIS MMC snap-in. The metabase uses an attribute inheritance scheme somewhat similar to the DACL inheritance scheme used in Windows 2000: When you change an attribute on a node, it propagates to all children of that node, except for those children that have provided their own definition of the attribute.

The attribute that controls whether HTTPS is required is *AccessSSL*. Here's a script that turns on this attribute for a virtual directory known as "Secure" on the default Web site:[19]

```
set vd = getObject("IIS://localhost/W3SVC/1/Root/secure")
vd.accessSSL = true
vd.setInfo
```

In the metabase, once a child node has defined an attribute, the flow of inheritance is interrupted at that node (exactly the same way SE_DACL_PROTECTED works in a security descriptor, as described in Chapter 6). If you want to remove the attribute from an object and unblock the flow of inheritance for that attribute, you must use the PutEx method:

```
set vd = getObject("IIS://localhost/W3SVC/1/Root/secure")
vd.putEx 1, "AccessSSL", ""
vd.setInfo
```

If you enable AccessSSL on a virtual directory, as long as no children provide their own definitions, they will automatically inherit the new setting, and all the resources subordinate to that directory will require the client to use HTTPS. (If the client attempts to access any of these resources via vanilla HTTP, she'll get an error instructing her to switch to HTTPS.) As with most metabase settings, you can control this setting on a per-file basis if you need that level of granularity. To access this property via the user interface, bring up the property sheet for the resource in question and choose Directory Security for a Web site or virtual directory object, or File Security for an individual file, and then press

[19] If you're not familiar with administering IIS programmatically, take a look at the IIS reference (just surf to http://localhost/iishelp on your Web server to get the help index). Also check out the `Inetpub\AdminScripts` directory for some example scripts.

brown10.qxd 7/5/00 11:38 AM Page 459

the Edit button in the section labeled Secure Communications. Most of the metabase keys this chapter describes have pretty obvious user interface representations, but some aren't accessible via the user interface at all (I'll point these out as I get to them).

The following list provides some related settings that might interest you, along with notes as to how they affect the behavior of IIS.

- **AccessSSL** This attribute was discussed previously; it requires use of HTTPS to access the resource.
- **AccessSSL128** This setting only makes sense if you've also turned on AccessSSL; it further limits cipher-suite negotiation to 128-bit or better bulk encryption keys. Most user agent software produced in the United States uses a relatively weak key by default to satisfy older U.S. export laws, so if you plan on using this feature, make sure all your clients are using software that can negotiate 128-bit encryption keys.
- **AccessSSLNegotiateCert** Causes the server to request a client certificate during SSL negotiation. If the client doesn't present a certificate (which is legal according to SSL), IIS will not fail the connection request. Thus some clients will present certificates, whereas others will not. This assumes that the `AccessSSLRequireCert` attribute is set to `False`.
- **AccessSSLRequireCert** This option only makes sense if you've also turned on `AccessSSLNegotiateCert`; this indicates that clients must provide a certificate in order to access the resource.
- **AccessSSLMapCert** Enables automatic mapping of client certificates to Windows security accounts. This is discussed in more detail in the section entitled Client Authentication.
- **AccessSSLFlags** Technically, all the previous options are stored in this one single attribute; the individual attributes simply provide an easier way to set these bits. The main reason I'm mentioning this is because all these settings are inherited as a unit. See the IIS documentation if you want the precise bit mappings.

brown10.qxd 7/5/00 11:38 AM Page 460

Managing Web Applications

Back in the old days with IIS 3.0, the Web server simply ran in a single process known as `INETINFO.EXE`, which lurked in the System logon session. All server-side Web applications therefore ran in the System logon session, which is really dangerous. Let me give an example why.

Interlude: The Buffer Overflow Attack

Consider the following C++ code:

```
void foo() {
  char buf[256];
  GetStringFromTextBox(buf);
}
```

This ultra-simplified code reads a string from a text box on a user-submitted form. Imagine that this code were housed in an Internet Server API (ISAPI) DLL running inside `INETINFO.EXE`, and that the user actually typed 300 characters into the text box. Because `GetStringFromTextBox` doesn't pay any attention to how much memory `buf` points to, it will happily overflow the buffer. Is this bad? It's incredibly bad. Because `buf` is on the stack, any overflow will quickly overwrite the return address on the stack, and when `foo` returns, it won't return to the caller, it'll return to whatever address was overwritten on the stack.

Crackers find great fun in sending unexpectedly long strings to applications as input and just waiting for one of them to explode. Once they discover a bug like this, it's just a matter of shortening the string until the program doesn't crash anymore. Now they know the exact length of the buffer and have a darn good idea where to write their own return address onto the victim's stack. By crafting an input string that contains binary executable machine instructions, all the cracker has to do is figure out the address of the buffer on the stack[20] and point the return address into the buffer. The most elegant attack I've seen

[20] This becomes quite easy if she can get a Dr. Watson crash report (in other words, if she can reproduce the bug on her own machine).

brown10.qxd 7/5/00 11:38 AM Page 461

documented[21] is one in which the cracker sends a small program as input that loads `WININET.DLL` and calls a few functions that download an executable program of the attacker's choosing onto the victim's hard drive. The program then launches the downloaded executable and quietly shuts down the victimized process in an attempt to hide the fact that something is horribly awry. (Note that simply using a safer language than C or C++ can help alleviate this problem.)

Imagine if this attack were carried out on an ISAPI application running in the System logon session. The attacker has now compromised the TCB of the machine and it's game over. It's funny, I remember when people were worried about ISAPI DLLs *crashing* the Web server; now that you've seen that a buggy ISAPI DLL can allow an attacker to *hijack* the Web server's process, I hope that the imperative for moving this code out of the System logon session is clear.

Introducing the Web Application Manager

IIS 4.0 addressed this problem in an elegant way. By separating the core Web server functionality present in `INETINFO.EXE` from the code needed to invoke ISAPI applications, it is possible to run ISAPI applications in separate processes, sandboxing them in lesser-privileged logon sessions. This separation was implemented by having the core Web server talk to the ISAPI environment via an (undocumented) COM interface. IIS 4.0 offered two choices for each Web "application" (each virtual directory is considered a separate application): The application could run in-process inside `INETINFO.EXE`, or it could run in its own separate process. In either case, a logical COM object representing the environment for each application was registered and added to the MTS catalog as a configured component. This component was named the **Web Application Manager** (or **WAM** for short). This is one case where COM's claim of "local-remote transparency" does in fact ring true; using a COM interface allows the Web server to interact with in-process as well as out-of-process WAMs polymorphically.

[21] It's hard to say if this essay will still be on the Web by the time you are reading this book, but if it is, it's great reading: http://www.cultdeadcow.com/cDc_files/cDc-351. Don't visit this site if you're easily offended by street talk. If you find that this site is unavailable, visit my Web site and I'll mirror it there.

brown10.qxd 7/5/00 11:38 AM Page 462

Unfortunately, IIS 4.0 didn't go quite far enough, and since each "isolated" Web application chews up another process, the default setting for each new Web application was to run in-process. IIS 5.0 makes out-of-process Web applications more attractive by providing *three* options for isolation:

1. Low (IIS Process)
2. Medium (Pooled)
3. High (Isolated)

Options 1 and 3 are exactly the same as what IIS 4.0 provided; in the first case, the application code runs in `INETINFO.EXE`, and in the last case, the application code runs in its own dedicated process. What's interesting is the new option 2, which places all pooled applications into a *single* process that runs outside `INETINFO.EXE` (and in a separate logon session, as you'd expect). This new option is the default in IIS 5.0, and is a great way to mitigate threats like the buffer overflow attack. (Don't get me wrong—the attack is still possible, but hijacking a process in the TCB is *very* different from hijacking one outside the TCB, assuming that the administrator has erected effective perimeter defenses around the TCB.)[22]

The process isolation level is represented by a metabase attribute named `AppIsolated`, an inheritable attribute that can be set on a per-application basis (the current IIS 5 docs indicate that this value can be set on subdirectories inside an application, but this has no effect). This attribute can have one of three values:

`0`: Low (IIS Process)

`1`: High (Isolated)

`2`: Medium (Pooled)

Because each WAM runs in the context of a configured COM+ application, you can actually use the Component Services snap-in to look at the two WAM-

[22] For an extreme example of carelessness, if everyone is granted all access to critical registry keys and system files, then the TCB can be compromised by simply replacing components of the operating system itself. Sandboxing a server in a lesser-privileged logon session won't help unless the administrator secures the perimeter of the TCB.

brown10.qxd 7/5/00 11:38 AM Page 463

hosting applications that IIS creates by default when it is installed with the operating system:

IIS In-Process Applications

IIS Out-Of-Process Pooled Applications

The first application is designated as a library application, so that when `INETINFO.EXE` creates an instance of the WAM for an in-process application, the WAM will load into that same process. The second application is designated as a server application, so that when `INETINFO.EXE` creates an instance of the WAM for a pooled application, that object will be served up from a single COM+ surrogate process.

If you look at the properties for the second application listed earlier, you'll notice that it is designated to run as a distinguished principal: IWAM_*MACHINE*, where *MACHINE* is the NetBIOS name of the machine at the time IIS was installed. This is a local account that is created by the system at IIS install time. Every Web application that you designate to run at the third level of isolation (Isolated) causes IIS to create another distinct COM+ application configured in a similar way.

To demonstrate the differences between these levels of process isolation, imagine creating six virtual directories, App1 through App6, and configuring them as follows:

App1: Low (IIS Process)

App2: Low (IIS Process)

App3: Medium (Pooled)

App4: Medium (Pooled)

App5: High (Isolated)

App6: High (Isolated)

Figure 10.9 shows the results if all these applications are in use simultaneously. Traversing over to the Component Services snap-in, here is the complete list of COM+ applications that are now hosting WAMs:

brown10.qxd 7/5/00 11:38 AM Page 464

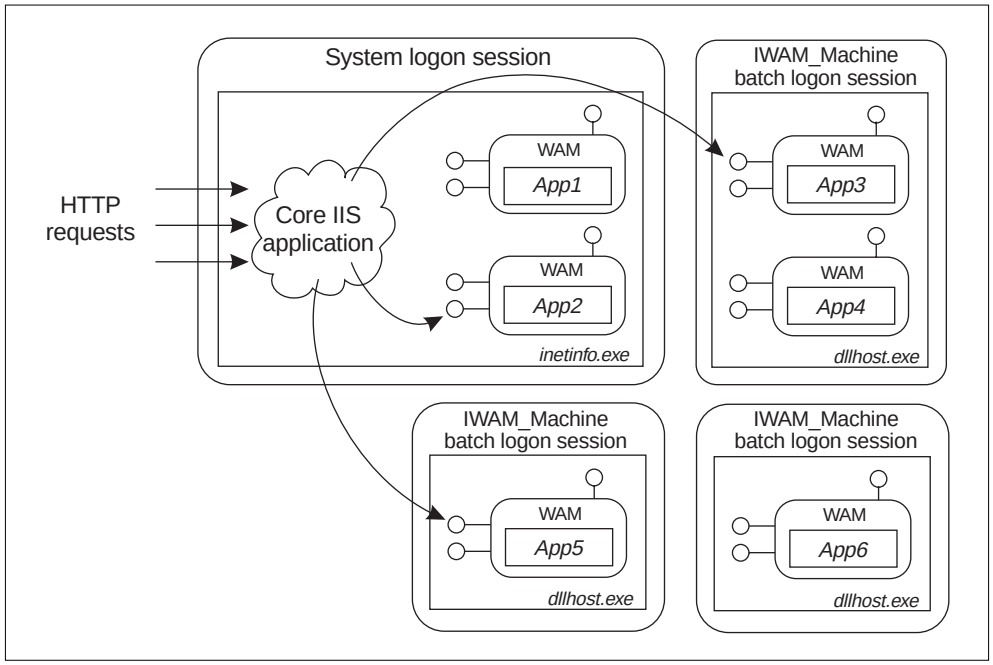


Figure 10.9 Runtime profile of Web applications

IIS In-Process Applications

IIS Out-Of-Process Pooled Applications

IIS-{Default Web Site//Root/App5}

IIS-{Default Web Site//Root/App6}

As Figure 10.9 makes clear, App5 and App6 are configured to run in separate server applications, also under the auspices of IWAM_MACHINE.

IIS actually doesn't hardcode the account name and password for IWAM_MACHINE; you can get and set these values via the metabase. If you change these values, IIS will use the new values from then on for the identity of all new COM+ applications it creates. Here's some code that displays the user name and password for the IWAM_MACHINE account (you must do this programmatically; these attributes aren't exposed via the IIS snap-in):

```
set w = getObject("IIS://localhost/W3SVC")
msgBox w.wamUserName & chr(13) & w.wamUserPass
```

If you make changes, be aware that IIS doesn't attempt to synchronize your changes with the security database (you can, of course, do this programmatically; see the appendix). If you change this to a user name of "Foo\Bob" with a password of "shazam," you'd better make sure that the Foo domain does in fact have an account called Bob, and that its password is indeed "shazam." Also note that I used the older-style authority\principal form for the name, because COM+ currently chokes on server identities in UPN form (bob@foo.com), as mentioned in Chapter 9.

Before you make a change like this, recall from Chapter 9 that it's usually safest to use a local account for a server's identity unless you absolutely need something different. Anyone with administrative access to the metabase can read this user name and password, so leaving this as a local account severely restricts the damage that can be done if the machine is compromised and the password disclosed.

Client Authentication

Although Figure 10.9 is interesting, it doesn't provide the full picture. Every WAM thread that performs work in response to an HTTP request from a client does so while impersonating. Which account that thread actually impersonates depends on a number of factors, but it's critical to remember that each WAM thread that makes calls to the file system, an ISAPI application, or an Active Server Pages (ASP) application (which is really just a stylized form of ISAPI) does so while impersonating *somebody*.

The various ways that IIS can determine the identity of the client, and thus obtain a logon session for the WAM thread to impersonate, are listed here in order of preference:

1. Anonymous (no authentication)
2. Certificate-Based Authentication
3. Integrated Windows Authentication

4. Digest Authentication
5. Basic Authentication

Each resource can have its own configuration for authentication, and you can select one or more of these options via the metabase. The metabase attributes for enabling or requiring client certificates were discussed previously. As for the other options, the `AuthFlags` attribute (a bitfield) controls which authentication options are allowed.[23] Here are the bit values you can use with this attribute:

`0x00000001`: Anonymous

`0x00000002`: Basic

`0x00000004`: Integrated Windows Authentication (SPNEGO)

`0x00000010`: Digest

Anonymous (No Client Authentication)

You may have noticed from the previous list that IIS prefers not to authenticate its clients. Authenticating each client consumes CPU resources and increases network traffic. Many large commercial Web sites typically don't rely on the operating system to perform authentication; rather, authentication is done at the application level (perhaps by asking for a credit card number and verifying the billing address, or even by asking for an application-defined password).

IIS has the notion of an "anonymous Internet user" account, which defaults to IUSR_MACHINE, another local account (similar to IWAM_MACHINE) that's created when IIS is installed. For a given resource, if the Anonymous option is enabled, IIS will check to see if the requested resource (HTML file, ASP script, GIF file, etc.) can be successfully accessed by this account. If the DACL on the file grants access to the anonymous Internet user account, IIS will execute the client's request under the auspices of that account, as opposed to trying any of

[23] There are also three subflags (`AuthAnonymous`, `AuthNTLM`, and `AuthBasic`), but they are rather out of date. There is no corresponding `AuthDigest` setting, and `AuthNTLM` should really be renamed `AuthNegotiate` because it now means SPNEGO.

brown10.qxd 7/5/00 11:38 AM Page 467

the more expensive authentication options.[24] This means that IIS will impersonate the anonymous Internet user account and open the file while impersonating. If instead the DACL denies access to the anonymous Internet user, IIS will move on to the next authentication option in the list (access will be denied if no other option is enabled for this resource).

Note that I've been careful to label this account in abstract terms; this is because it doesn't always have to be IUSR_MACHINE. Rather, each individual Web resource (Web site, virtual directory, file system directory, file) in the IIS metabase has the following (inheritable) attributes:

- **`AnonymousUserName`** The name of a valid user account that IIS will use to establish a logon session for anonymous Internet users when they request this particular resource
- **`AnonymousUserPass`** The password for the account just described
- **`AnonymousPasswordSync`** A boolean attribute with magical properties that I'll describe shortly

By default, the `W3SVC` node in the metabase (the root of all Web nodes) sets `AnonymousUserName` to `IUSR_MACHINE`, and `AnonymousPasswordSync` to `True`. This latter property is quite magical; when set, it causes `INETINFO.EXE` to obtain a logon session without providing a password by invoking a special subauthentication DLL in the LSA that basically says "Sure, I'll give you a logon" without checking the password at all. This is a nifty feature, but be aware that it's only supported for local accounts.[25] If this feature alarms you, remember that `INETINFO.EXE` runs in the System logon session and is therefore part of the TCB. This is just an example of a member of the TCB doing whatever it pleases on the local machine.

Based on my own experimentation with IIS 5.0, if you use the password synchronization option, the resulting logon session for the anonymous user will be a network logon session with no network credentials, which makes sense

[24] One exception to this rule is as follows: If the server requests a certificate and the client presents one, and if a mapping to a Windows user account can be found (I'll cover this shortly), IIS will use the mapped account as opposed to simply accessing the resource as the anonymous Internet user.

[25] Although with experimentation, I was able to get it to work with domain accounts as well. If you try to configure a domain account with this magic setting via the Internet Services snap-in, you'll

brown10.qxd 7/5/00 11:38 AM Page 468

because IIS obtained the logon session without providing a password. On the other hand, if you don't use this feature and you specify a password explicitly, the logon session will be an interactive logon session with network credentials. (Of course, if you're using a local account, these network credentials won't buy you much unless you've created a matching user name and password on some other machine.)

Another thing to be aware of is that when two or more resources share the same anonymous account settings (which is typically the case), IIS does its best to cache a single logon session and impersonate that one logon session for all those resources, even across isolated application boundaries (Figure 10.10 shows an example). If you've read Chapter 8, this should make you think, "I'm probably sharing a logical LAN Manager client port with other Web applica-

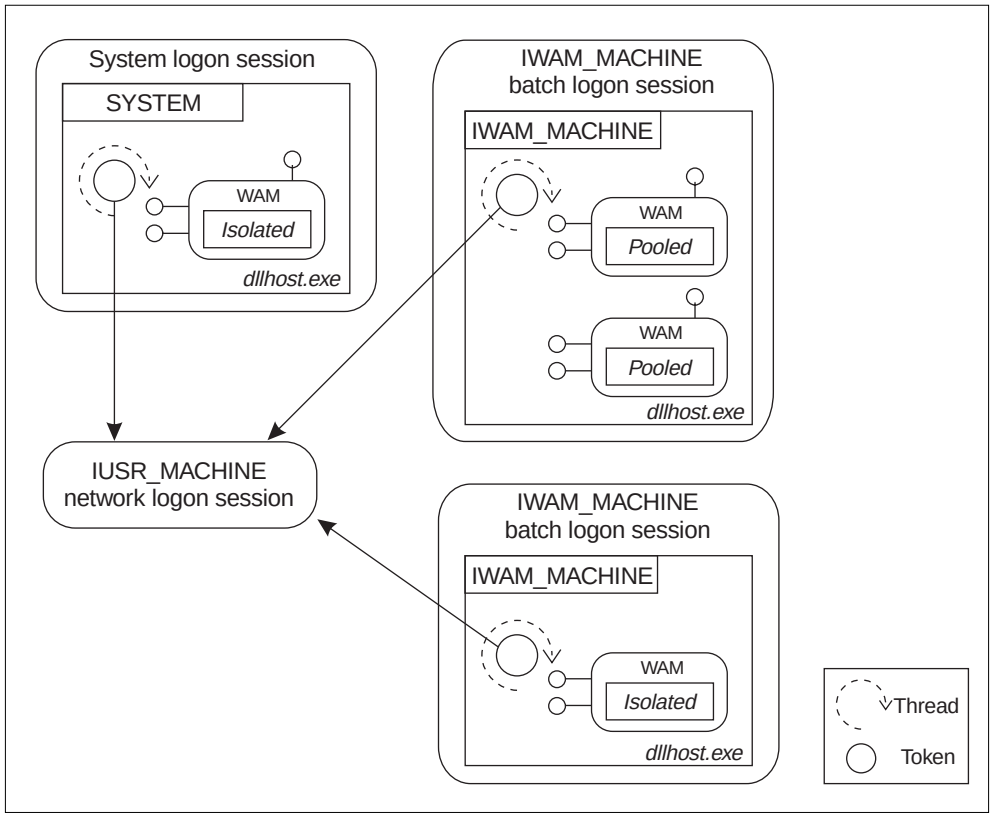


Figure 10.10 A typical set of logon sessions for anonymous Internet users

brown10.qxd 7/5/00 11:38 AM Page 469

tions." If you try to use alternate credentials to access a remote file server by calling `NetUseAdd`, you might succeed, but you'll be changing the credentials used for that server for *all other Web applications* that are also using the anonymous logon. It's also possible that someone has already beat you to the punch, in which case you'll run into the dreaded conflict of credentials that was discussed in Chapter 8. Watch your step here. This is one case in which having your application set to an isolation level of High can be a good thing because you can always call `RevertToSelf` to get to your own application's private batch logon session (for IWAM_MACHINE), at which point you can call `NetUseAdd` without tromping on any other applications.

Certificate-Based Client Authentication

Earlier in the chapter I described the SSL protocol and explained that as long as the server has a certificate, the server is allowed to request a certificate from the client. I also described the metabase attributes that allow you to control whether IIS should request or even *require* a client certificate.

Before IIS will accept a client certificate, it must verify that that certificate is valid. This involves comparing the current date with the valid from/to dates in the certificate, followed by verifying the signatures in the certificate chain. Finally, IIS must determine whether any of the certificates in the chain are trusted. By default, each Web site trusts all certificates that haven't expired, so it's a good idea to create a certificate trust list (CTL) for your Web site that limits this trust to certificates for a selected set of trusted authorities. To do this, bring up the property sheet for the Web site, choose the Directory Security tab, and then press the Edit button that takes you to the certificate-related settings. Check the box that says "Enable certificate trust list." This tells IIS not to trust any authorities except the ones that you put in the CTL. Press New to invoke a wizard that will help you build the CTL; it's pretty straightforward once you've figured out which certificate authorities you need to trust.

As IIS validates certificates, it needs to check for revocation, which can be time-consuming. By default IIS skips this step, but you can control this via the `CertCheckMode` metabase attribute on each Web site (this is not available via the IIS snap-in; it must be set programmatically). Setting this value to a number greater than 0 forces IIS to check for revocation (which may

brown10.qxd 7/5/00 11:38 AM Page 470

involve downloading a CRL from a certificate authority, or simply looking in a cached CRL). If you are serious about client authentication, you should enable this check.

At this point, another metabase attribute becomes useful: `AccessSSLMapCert`, which is really just another bit that can be set or cleared in the `AccessSSLFlags` attribute that was mentioned early in the chapter. If this attribute is set to `True`, IIS will attempt to map the client's certificate onto a Windows user account via one of two mechanisms: Either IIS will use its own internal table of mappings from certificates to user accounts/passwords, or it will ask its domain controller to perform the mapping based on settings in the directory service. You can control which method is used via the metabase attribute `SSLUseDSMapper`. (This is a global setting on the `W3SVC` node in the metabase, accessible via the Web Service Master Properties in the user interface; as of this writing, you cannot control this on a per–Web site basis.) By default `SSLUseDSMapper` is set to `False`, which indicates that IIS should perform its own internal mapping.

An administrator can configure the internal IIS certificate-to-account mapping via the IIS snap-in. Each Web site has its own set of account mappings, which can be edited by going to the property sheet for the Web site, choosing the Directory Security tab, and pressing the Edit button that takes you to the certificate-related settings. After checking "Enable client certificate mapping," which is just the `AccessSSLMapCert` metabase setting, press Edit to configure the certificate-to-user-account mappings.

There are two ways to map certificates to user accounts in IIS: one certificate to one account, or many certificates to one account. The first method requires you to obtain a certificate from the user (typically packaged in a base-64-encoded ASCII file), whereas the second method allows you to simply specify a set of criteria for the various components of the distinguished name for the subject and issuer in the certificate (for instance, you can detect certificates issued by VeriSign in which the subject's organization is ACME Corporation and map these certificates onto a particular user account). With the many-to-one option, you can either grant *or deny* access to a matched certificate. Denying access is easy—you just need to indicate that this is your intention via a radio button.

brown10.qxd 7/5/00 11:38 AM Page 471

If you want to grant access (using the one-to-one or many-to-one option), you'll need to provide a user account and password, which IIS will tuck away in the metabase. If you want to configure this mapping programmatically via script, you'll want to check out the `IIsCertMapper` interface; this interface currently allows you to set up one-to-one mappings, but does not provide for many-to-one mappings, which must be set up via the IIS snap-in.

If IIS authenticates a client via one of these internal certificate mappings, the result will be an interactive logon session with the network credentials of the account (which makes sense, because the logon session was created locally with a password). If, instead, IIS delegates authentication to the directory service, it will establish a network logon session that will (naturally) not have network credentials. If a match cannot be found, or if the client did not submit a certificate,[26] IIS will move on to the next authentication option in the list, or will deny access if no further options are available.

If Alice was issued a certificate from an enterprise security authority in her domain (one that's integrated with the directory service), her certificate will have enough information in it for the directory service to determine which account is Alice's. For other clients, it's possible to set up certificate mappings manually in the directory service: Bring up the Active Directory Users and Computers snap-in, turn on the Advanced Features option via the MMC View menu, then right-click on any user account and choose Name Mappings to do this.

Certificates are the most natural and, when used properly, can be the most secure way to authenticate clients over the Internet, but there are also other options available if issuing client certificates doesn't make sense for your application.

Basic Authentication

Basic Authentication is the native authentication mechanism that is built into HTTP. The client sends an HTTP request to the server, and the server sends back a failure, demanding that the client prove his or her identity by sending a

[26] This of course assumes that you haven't turned on the `AccessSSLRequireCert` attribute, which causes IIS to immediately fail the connection during the SSL handshake if the client doesn't provide the requested certificate.

brown10.qxd 7/5/00 11:38 AM Page 472

user name plus a base-64-encoded ASCII password. Base 64 is not an encryption algorithm; rather, it's an expansion algorithm that allows Internet-unfriendly characters to be represented by friendly ones, and is therefore completely reversible without a key. Thus Basic Authentication only makes sense over an SSL encrypted link with strong server-side authentication. As long as you know who you're sending the password to, and as long as you are assured that nobody but that server will see the password, Basic Authentication works just fine, assuming you trust the server with your cleartext password.

If you attempt to turn this authentication mechanism on via the IIS snap-in, you'll get a very nasty warning that says, "Don't do this unless you plan on using SSL to secure the connection." This is great advice; you should *require* SSL for any resources where you plan to allow Basic Authentication. If you can't afford a public key infrastructure, this combination will serve you well.

When IIS receives the client's user name and password, it will check the `LogonMethod` attribute in the metabase to determine which type of logon session to establish. There are three options:

`0`: Interactive logon (the default)

`1`: Batch logon

`2`: Network logon

The first two result in a logon session with network credentials; the network logon naturally does not have network credentials. Which one should you choose? Well, a batch-style logon will be pretty tough for most clients to establish, because no principals are granted this logon right by default. An interactive logon is more wide open in this respect, depending on the type of machine your server runs on (domain controllers severely limit the right to an interactive logon). However, establishing an interactive logon can be considerably more expensive than establishing a batch or network logon (as I discussed in Chapter 4). Generally, everyone is granted a network logon on any machine, including domain controllers. (Whether it's good security practice to run a public Web server on a domain controller is another question entirely; you should avoid doing this in my opinion, as I'll discuss shortly.)

One thing you should be aware of when using Basic Authentication is that Web applications (CGI, ISAPI, ASP) will have access to the client's cleartext

brown10.qxd 7/5/00 11:38 AM Page 473

password via a server variable named AUTH_PASSWORD. I know of no way of disabling this, which is too bad, considering IIS has already used the password to authenticate the client; letting this information leak through to scripts is dangerous. (Just because you trust the TCB of a server with your password doesn't mean you want to trust all its scripts with your password as well.) However, even if this little flaw didn't exist, trusting the TCB of the Web server with your credentials may be more than a security-conscious client is willing to do. In that case, consider using a client-side certificate, which is ultimately a client's safest bet.

Digest Authentication

Digest Authentication is a relative newcomer (it was introduced with HTTP 1.1 and was first implemented by Microsoft in IIS 5 and IE 5), but it will likely have limited use as certificate-based authentication becomes more and more popular. Digest Authentication is somewhat similar to NTLM: It's a simple challenge/response protocol that allows a client to prove knowledge of a password without transmitting the password across the wire. It does not provide mutual authentication, and it does not provide a way to exchange a session key for data encryption or MAC generation. It also requires password storage for users to be weakened significantly. (The passwords must be stored in such a way that the domain controller can decrypt them to obtain a plaintext password; this is known as using *reversible encryption*.)

Probably the most dangerous thing of all is that the only way this mechanism can be used is if the Web server resides on the same machine as the domain controller (to have access to those cleartext passwords). Exposing a public Web server from a domain controller opens some serious security risks; you'd better make absolutely sure that all the applications on your server are sandboxed or bug-free, because a compromise of the TCB of a domain controller is a compromise of the domain itself.

I mention this protocol only for completeness. If you care about securing the transactions on your Web server, you'll get a server certificate and use SSL to encrypt the session. The authors of the spec (RFC 2069) state their position as follows:

brown10.qxd 7/5/00 11:38 AM Page 474

> Digest Authentication does not provide a strong authentication mechanism. That is not its intent. It is intended solely to replace a much weaker and even more dangerous authentication mechanism: Basic Authentication. An important design constraint is that the new authentication scheme be free of patent and export restrictions.
>
> Most needs for secure HTTP transactions cannot be met by Digest Authentication. For those needs SSL or SHTTP are more appropriate protocols. In particular, Digest Authentication cannot be used for any transaction requiring encrypted content. Nevertheless, many functions remain for which Digest Authentication is both useful and appropriate.

Integrated Windows Authentication

The Integrated Windows Authentication option tells IIS to engage the user agent in a native Kerberos or NTLM handshake piggybacked on HTTP. These extensions are not supported by non-Microsoft browsers, which significantly limits the option's appeal for use with clients on the Internet at large. Another problem with using Kerberos or NTLM is that neither protocol gets along very well with firewalls. For instance, the Kerberos KDC listens on port 88 for TCP or UDP requests; can you imagine any administrator in his or her right mind opening this port to allow random crackers on the Internet to have direct conversations with the most trusted entity in the domain? I think not. Microsoft and Cisco have proposed extensions to Kerberos to work around this problem (see the Internet draft "Initial Authentication and Pass Through Authentication Using Kerberos V5 and the GSS-API"), but this still won't solve the browser problem; most browsers expect to use certificates to authenticate their clients.

Within the perimeter of the firewall in the confines of a Windows-only enterprise, however, this is a very convenient option. Alice doesn't need to be issued a certificate to participate. She just has to agree to use Internet Explorer, and the system will use the network credentials that were established when she logged in via Winlogon to authenticate her to IIS servers on the company intranet.

The result of this form of authentication is (naturally) a network logon session on the server. (This assumes the client is not running her browser on the same machine as the Web server; in that case often the client's local interactive logon session will be used directly.)

brown10.qxd 7/5/00 11:38 AM Page 475

If authentication fails using the client's default credentials (either because of problems with authentication or because the client has not been granted access to the resource), Internet Explorer will pop up a dialog asking for alternate credentials, allowing the client to retry.

Server Applications

Three classes of server applications are in common use with IIS today: raw ISAPI applications, ASP script-based applications, and Common Gateway Interface (CGI) applications. All these applications are ultimately derived from the basic CGI model that provides a set of variables describing the client and server environment as well as the request, along with input and output streams for reading and writing the request and response bodies. The interface that exposes these variables and streams to the application differs significantly among the three types of applications, but the basic information provided is similar. Table 10.1 lists some interesting variables that are security related.

Table 10.1 Security variables for server applications

Variable	Definition
AUTH_TYPE	Indicates the authentication mechanism that IIS used to obtain the logon session for the client.
REMOTE_USER, AUTH_USER, LOGON_USER	Although the IIS documentation discusses each of these separately, giving them slightly different semantics, in practice they're generally the same, and refer to the client principal that was authenticated in authority\principal form.
AUTH_PASSWORD	For Basic authentication, this is the cleartext password that the client specified (better trust those ASP scripts!). For Digest Authentication, this is the set of values computed by the digest (see RFC 2069 for details). For all others, this is blank.
SERVER_PORT_SECURE	0 indicates HTTPS is not in use; 1 indicates HTTPS is in use.
HTTPS	"off" indicates HTTPS is not in use; "on" indicates HTTPS is in use.

Table 10.1 Security variables for server applications (continued)

Variable	Definition
CERT_FLAGS	This is a bitfield with the following masks: 0x0001: A client certificate was received. 0x0002: The issuer is not a trusted authority.
CERT_SUBJECT	The common name of the subject listed in the client certificate.
CERT_ISSUER	The distinguished name of the issuer listed in the client certificate.
CERT_SERIALNUMBER	The client certificate serial number.
CERT_COOKIE	The hash value of the client certificate.
CERT_SERVER_ISSUER, HTTPS_SERVER_ISSUER	The distinguished name of the issuer listed in the server certificate.
CERT_SERVER_SUBJECT, HTTPS_SERVER_SUBJECT	The distinguished name of the subject listed in the server certificate.
CERT_KEYSIZE, HTTPS_KEYSIZE	The strength, in bits, of the negotiated bulk encryption key.
CERT_SECRETKEYSIZE, HTTPS_SECRETKEYSIZE	The strength, in bits, of the server's private key exchange key.

I wrote a simple script that echoes back these security-related variables, and ran it several times while changing the authentication options on the server. Table 10.2 shows my results when running over HTTP (note that I've omitted all the certificate-related fields because naturally they were all empty). Here are the values that I didn't have room for in the table:

```
<dpw1>
    username="quux\alice", realm="preston", qop="auth",
    algorithm="MD5", uri="/b/x.asp",
    nonce="2150219db204639164346810000031fabb800044f103d689
    0a9e076c63f6", nc=00000001,
    cnonce="d2694a8e9e65dd4607378e132f284eb4",
    response="e03cce66fa29f14dccd16bb8835c924e"
```

brown10.qxd 7/5/00 11:38 AM Page 477

Table 10.2 Resulting variables over HTTP

Variable	Anonymous	Basic	Digest	Integrated
AUTH_TYPE		Basic	Digest	Negotiate
REMOTE_USER		quux\alice	quux\alice	QUUX\alice
LOGON_USER		quux\alice	quux\alice	QUUX\alice
AUTH_USER		quux\alice	quux\alice	QUUX\alice
AUTH_PASSWORD		woosel	<dpw1>	
SERVER_PORT_SECURE	0	0	0	0
HTTPS	off	off	off	off

brown10.qxd 7/5/00 11:38 AM Page 478

Table 10.3 Results of using variables over HTTPS

Variable	Anonymous	Basic	Digest	Integrated	Client Certificates
AUTH_TYPE		Basic	Digest	Negotiate	SSL/PCT
REMOTE_USER		quux\alice	quux\alice	QUUX\alice	QUUX\alice
LOGON_USER		quux\alice	quux\alice	QUUX\alice	QUUX\alice
AUTH_USER		quux\alice	quux\alice	QUUX\alice	QUUX\alice
AUTH_PASSWORD		woosel	<dpw2>		
SERVER_PORT_SECURE	1	1	1	1	1
HTTPS	on	on	on	on	on
CERT_FLAGS					1
CERT_SUBJECT					<subject>
CERT_ISSUER					<issuer>
CERT_SERIALNUMBER					<serial>

brown10.qxd 7/5/00 11:38 AM Page 479

Variable	Anonymous	Basic	Digest	Integrated	Client Certificates
CERT_COOKIE					<cookie>
CERT_SERVER_ISSUER	<sissuer>	<sissuer>	<sissuer>	<sissuer>	<sissuer>
HTTPS_SERVER_ISSUER	<sissuer>	<sissuer>	<sissuer>	<sissuer>	<sissuer>
CERT_SERVER_SUBJECT	<ssubject>	<ssubject>	<ssubject>	<ssubject>	<ssubject>
HTTPS_SERVER_SUBJECT	<ssubject>	<ssubject>	<ssubject>	<ssubject>	<ssubject>
CERT_KEYSIZE	56	56	56	56	56
HTTPS_KEYSIZE	56	56	56	56	56
CERT_SECRETKEYSIZE	512	512	512	512	512
HTTPS_SECRETKEYSIZE	512	512	512	512	512

brown10.qxd 7/5/00 11:38 AM Page 480

The results of the same script invoked via HTTPS are shown in Table 10.3. This table makes it very clear which elements relate to the server-side certificate as opposed to the client-side certificate. (Both client and server certificates in this case were generated by quux.com, the enterprise certificate authority in my test system.) Here are the values that I didn't have room for in the table:

```
<dpw2>
    username="quux\alice", realm="preston", qop="auth",
    algorithm="MD5", uri="/b/x.asp",
    nonce="fb210da32ca30592643468100000cc6d97809692660cb7d6
    b596f5e731c1", nc=00000001,
    cnonce="8e7206169ad8e237f3fb08d525cc2378",
    response="f5faa59c9eeeb7ded74ea8ade38b77cf"
<serial>
    29-92-0f-64-00-00-00-00-00-0b
<cookie>
    6f245f3a7b3e0d19c84e03786bbf6e41
<subject>
    CN=alice
<issuer>
    E=no@mail.com, C=US, S=CA, L=Torrance, O=quux.com,
    OU=lab, CN=Quux Authority
<ssubject>
    C=US, S=CA, L=Torrance, O=quux.com, OU=lab,
    CN=wendoline
<sissuer>
    E=no@mail.com, C=US, S=CA, L=Torrance, O=quux.com,
    OU=lab, CN=Quux Authority
```

Notes on ISAPI Applications

ISAPI applications are hosted by the appropriate WAM depending on how the Web application is configured in the metabase with respect to process isolation. Each thread that enters the ISAPI DLL will be impersonating; the particular account being impersonated will depend on the mechanisms described earlier. One interesting difference between IIS 4 and IIS 5 is that on the earlier platform, calls to `GetObjectContext` failed with `CONTEXT_E_NOCONTEXT`. This made it impossible for the ISAPI DLL to call `IObjectContext::CreateInstance`, which was required in Windows NT 4 to flow the impersonation

token to the configured component being created (in order to perform role-based access checks based on the *client* as opposed to IWAM_MACHINE or SYSTEM). This requirement was removed in IIS 5 (Brown 1999d), and the client's security context appears to flow correctly, at least according to my own experiments.

Notes on ASP Applications

Although ASP is implemented as an ISAPI DLL, it provides its own thread pool and transfers incoming requests from the WAM thread onto its own thread before making calls into your scripts. Never fear, ASP also transfers the token on the WAM thread to its own thread before making the call, so threads entering ASP scripts will *always* be impersonating, using the mechanisms described earlier.

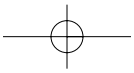
As mentioned previously for ISAPI applications, in IIS 4 it's critical that you call `IObjectContext::CreateInstance` when creating configured components; otherwise, the thread's security context will not be propagated to the new object. This can either cause all calls to be denied (typical if your WAM is running under IWAM_MACHINE) or cause all calls to be allowed, because SYSTEM is not subject to role-based access checks (this happens if your WAM is running in-process in `INETINFO.EXE`). This is not a problem at all in IIS 5.

So for IIS 4, it's critical that scripts use `Server.CreateObject` to instantiate objects (using <object> tags is also safe). Forgetting to do this can lead to unpredictable and unsafe behavior.

Another interesting security-related tidbit is that if a client submits a certificate, you can access all the various information in that certificate from ASP by using the `Request` object's `ClientCertificate` method. Here's a simple ASP script that echoes back all the attributes in the client's certificate in a table:

```
<table border=1 cellpadding=3>
<thead><td>Key</td><td>Value</td></thead>
<% for each key in Request.ClientCertificate %>
<tr><td><%= key %></td>
<td><%= Request.ClientCertificate(key) %></td></tr>
<% next %>
</table>
```

brown10.qxd 7/5/00 11:38 AM Page 482



Notes on CGI Applications

Just as ISAPI and ASP applications run in the logon session of the client (or the anonymous Internet user)—or at least their threads run in that logon session via impersonation—so too do CGI applications, except that the entire process is directed into the client's logon session, as opposed to just a thread in that process. This is a great sandboxing measure for CGI applications, but there is a metabase attribute that can be used to force a particular CGI application (or a whole host of them; this attribute is inheritable, as are most metabase attributes) into the System logon session. This attribute is known as `CreateProcessAsUser`, and it's set to `True` by default, which is where you should leave it unless you really know what you're doing.

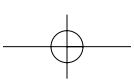
Setting this attribute to `False` causes CGI applications to run in the System logon session, in the noninteractive window station. (Not that the latter is in any way a sandbox; when running as `SYSTEM` you can pretty much move to any window station you desire.)

IIS as a Gateway into COM+

The primary reason for incorporating a Web server in a distributed three-tier application is to broaden the reach of the application to a multitude of platforms. (Imagine trying to use DCOM to reach clients on all the various flavors of UNIX, or on a handheld like the Palm OS or a simple appliance like a pager.) A Web-based front end is the de facto architecture for reaching across the Internet at large.

However, the natural architecture for a classic three-tier Windows-based distributed application is to use a Web server simply as a gateway into a more structured environment built with COM+ components. From a security standpoint, this type of architecture makes a lot of sense and can help you avoid an overabundance of application-level security checks. Once again, the less security-related code in your application, the better off you'll be.

Because IIS provides a plethora of authentication services that you can choose declaratively, all you have to do is figure out a way to smoothly move the client's security context from the WAM into your COM+ components and let COM+ role-based access checks do the heavy lifting. ASP goes to great pains



brown10.qxd 7/5/00 11:38 AM Page 483

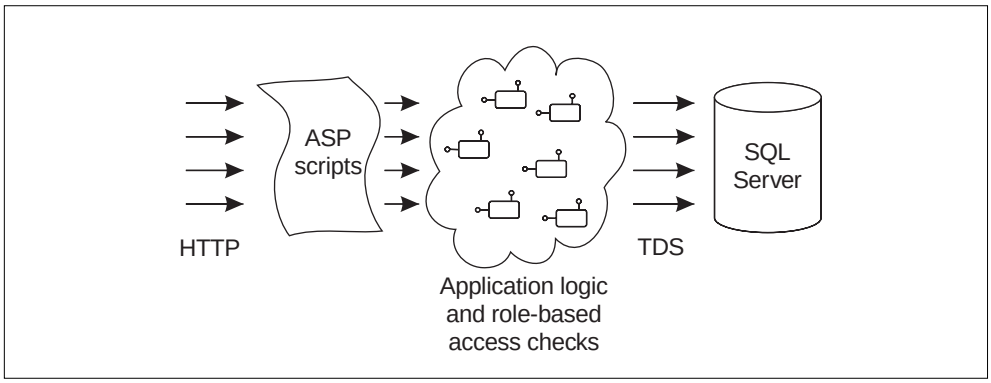


Figure 10.11 Natural three-tier architecture with a Web front end

to do its part, by transferring the client's token from the WAM thread to the ASP thread. COM+ also gives you a boost by using dynamic cloaking in all COM+ applications by default. Because the WAM is a configured component, it runs in a process where dynamic cloaking is enabled, and thus each outgoing COM call you make from an ISAPI application or ASP script will go out using the client's security context. (Recall that dynamic cloaking implies that outgoing COM calls are sensitive to the current thread token; without cloaking turned on, COM ignores the thread token for outgoing calls.) Things didn't work nearly as smoothly in Windows NT 4, as detailed in Brown (1999d).[27] Figure 10.11 shows natural three-tier architecture in which IIS acts as the gateway into a collection of COM+ components in the middle tier.

If you plan on collocating the Web server and your COM+ applications on the same machine (as in Figure 10.12), the security architecture will be very natural and obvious. (There might even be a farm of these machines, each with a Web server and a set of equivalent COM+ applications, but that's not a problem.) The key is having the Web application and the components it uses located on the same machine. Because dynamic cloaking is turned on in the WAM by default, when you make calls from your scripts into your local COM+

[27] If you've ever wondered what the MTS Trusted Impersonators group in Windows NT 4 was for, you'll enjoy reading this article.

brown10.qxd 7/5/00 11:38 AM Page 484

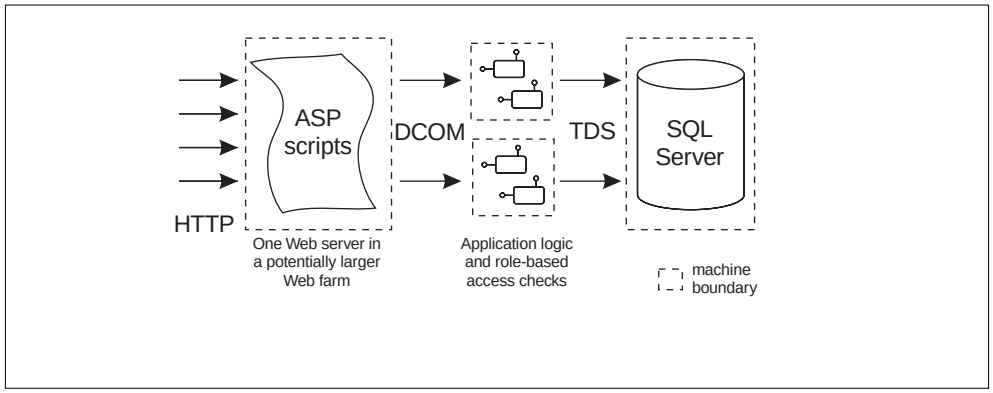


Figure 10.12 Collocating role-based access checks on the Web server

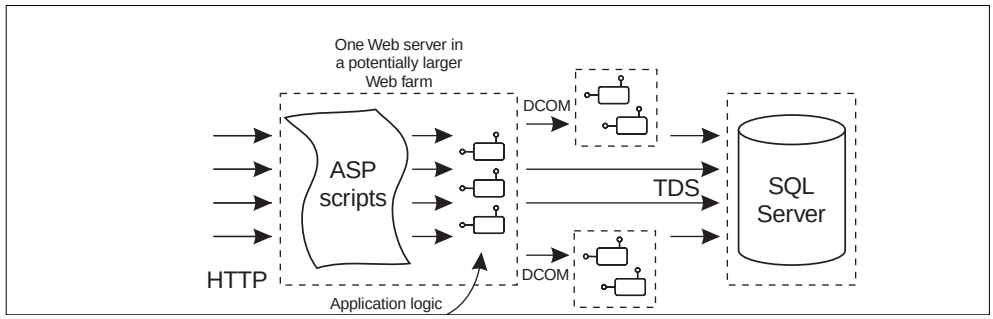


Figure 10.13 Separating role-based access checks from the Web server

components, role-based access checks will occur naturally against the client's security context.

On the other hand, if you plan on putting the Web server on one machine and your COM+ components on a separate machine (see Figure 10.13), you have to make sure that the client's logon session on the Web server will have network credentials in order to survive the hop from the Web server machine to the other machine where the role-based access checks will occur. If the client's logon session doesn't have network credentials, you'll end up making the call

brown10.qxd 7/5/00 11:38 AM Page 485

using a NULL session; if your client was in fact authenticated, you've basically just dropped that information on the floor.

Table 10.4 shows when the logon session for the client will have network credentials and when it won't, depending on the type of authentication in use (this table assumes the client is on a different machine than the Web server). As you can see, the options are not very pretty. Practically speaking, if the client is authenticated (and therefore not using the anonymous Internet user account), unless you're using Basic Authentication you likely won't have network creden-

Table 10.4 Authentication and network credentials

Scenario	Network Credentials?	Which Credentials?
Anonymous; password sync on	No	
Anonymous; password sync off	Yes	Anonymous Internet user account in effect at that node in the metabase, typically IUSR_MACHINE
Certificate based; directory service mapping on	No	
Certificate based; directory service mapping off	Yes	User account specified in IIS internal mapping
Basic Authentication; interactive logon method	Yes	Client-specified user name and password
Basic Authentication; batch logon method	Yes	Client-specified user name and password
Basic Authentication; network logon method	No	
Digest Authentication	No	
Integrated Windows Authentication	No	

brown10.qxd 7/5/00 11:38 AM Page 486

tials, and you won't be able to pass the access control responsibilities on to COM+ components on a different machine in the middle tier.

If you're adamant about using this multihop architecture, you might consider using a COM+ library package, configured with role-based security, as a layer between you and the remote objects that your script would normally talk to directly. These library components can mimic the remote object's interfaces (in fact, this code can be generated automatically based on a type library if you're smart about it), and can simply delegate calls to the remote object.

The beauty of this model is twofold. First, you don't want to be making calls from ASP scripts directly to remote objects, because most scripting languages double the number of round-trips required to make each method call (think `IDispatch::GetIDsOfNames` followed by `IDispatch::Invoke`). If you generate the code for the library component layer in a vtable-friendly language such as Visual Basic, Java, or C++, the `IDispatch` round-trips will be negligible because they'll happen locally between your script and the library components. When the library components actually make calls across the wire they can use vtable interfaces and execute the call in a single round-trip.

Second (back to security), your role-based checks will now work as desired because the COM+ interceptors in the library application will see the client's security context. Note that this trick requires Windows 2000, because MTS library packages have no notion of role-based security.

Miscellaneous Topics

This section contains some miscellaneous tips and traps to watch out for.

Using Client IP Addresses to Control Access

Because HTTP is designed to run over TCP/IP, and TCP/IP headers include the source IP address and port, it's possible to grant or deny access purely on the basis of the client's host address. Although this is an interesting feature that is commonly used to block unsophisticated pranksters from accessing Web resources on a server, unless this policy is backed up with IPSec (which authenticates host addresses cryptographically using Kerberos), simply relying on a client's advertised host address to determine access restrictions is very insecure. Spoofing a network address in a TCP request is relatively easy in the big

brown10.qxd 7/5/00 11:38 AM Page 487

scheme of things. If you decide to use this feature and want to set this up via a script, you'll use the methods defined under the `IIsIPSecurity` interface.

Mapping Virtual Directories to UNC Paths

Mapping virtual directories to UNC paths is an interesting feature that IIS supports. Instead of having a virtual directory map to a file system on a local device, you can direct it to a remote host by specifying a UNC path (\\machine\share). Remember, though, IIS is *always impersonating* when it accesses files for a client, and depending on the authentication mechanism you've chosen, the logon session being impersonated will likely not have network credentials. Thus, there are three metabase attributes for every virtual directory that come into play when a UNC path is in use:

- **`UNCUserName`** The user name that should be used in lieu of the actual client's identity to access the resource. According to my own experiments, this must be in authority\principal form, rather than UPN (principal@authority) form.
- **`UNCPassword`** The corresponding password for the account.
- **`UNCAuthenticationPassthrough`** Instead of setting a user name and password, you can set this attribute to True (the default setting is False) to indicate that you want to attempt to delegate the client's credentials if possible to make the additional network hop to the remote file system.

  If all parties involved support Kerberos, and the computer that hosts the Web server has been designated "trusted for delegation," then if the client's account has not been marked "sensitive and cannot be delegated" the Lan Manager client will ask for a forwarded TGT for the Web server to use to delegate the client's credentials and make the call. This of course assumes that you've enabled Kerberos authentication for the virtual directory (by selecting Integrated Windows Authentication). There are so many constellations that must align correctly for this to work that it's questionable whether it's worth it to even enable this option, but as you'll see, it's probably no worse than using a hardcoded user account.

When you hardcode a user account and password, IIS simply verifies that the client can be authenticated (period) according to the `AuthFlags` attribute for the target resource. Thus, if you have allowed anonymous access, everyone gets past this hurdle. Once the authentication requirement is satisfied, IIS now ignores the real client's identity and instead establishes an interactive logon session for the user specified via `UNCUserName`. (This will of course fail if the specified user has not been granted the interactive logon right on the machine hosting the Web server.) IIS impersonates this logon session (as it would normally impersonate the client's logon session, or IUSR_MACHINE) and executes the request. This means that if you are targeting an ASP script via UNC redirection, that ASP script will never see the actual client authenticated by IIS; rather, it will see the user principal specified via `UNCUserName`. This can lead to security holes unless you're paying attention, so watch your back.

Using RevertToSelf to Reenter the TCB

Imagine that you had a Web application that ran at a process isolation level of Low (in other words, inside `INETINFO.EXE`). Although this can be dangerous, it can also allow you to do very powerful things. However, because all threads that call into your application from the WAM will be impersonating *someone,* you need to rejoin the TCB temporarily if you need to execute privileged code (for instance, if you want to call `LogonUser`).

The following piece of code shows how this can be done (I've packaged this in a nonconfigured, in-process COM component so that you can call it directly from an ASP script):

```
HRESULT Foo::DoSomethingPowerful() {
  HANDLE htok;
  OpenThreadToken(GetCurrentThread(),
                  TOKEN_IMPERSONATE,
                  TRUE, &htok);
  RevertToSelf();
  // we're now executing in the TCB!
  SetThreadToken(0, htok);
  CloseHandle(htok);
  return S_OK;
}
```

brown10.qxd 7/5/00 11:38 AM Page 489

This code temporarily removes the thread token, causing the WAM thread to execute in the security context of the process, in this case `INETINFO.EXE`, and the System logon session. Note that the code carefully places the token back on the thread before returning to the WAM; restoring the environment this way protects the application from any assumptions that ASP or the WAM might make, and is purely a precautionary measure.

Calling this code from a Web application configured at Medium or High isolation also allows you to run in the security context of the host process, but in this case it won't be the System logon session; rather, it'll be (typically) IWAM_MACHINE, which is not very privileged at all, at least by default, and thus is much less interesting.

Debugging Authentication Settings

If at any time you want to find out what security context your ASP script is running in, you can download a little component that I built called the token dumper (from http://www.develop.com/books/pws). This COM component exposes a single method that scripts can invoke:

```
HRESULT TokenDump([in]          long  grfOptions,
                  [out, retval] BSTR* pbstrResult);
```

The options allow you to limit the output in some ways (see the documentation that comes with the tool for details); just pass `-1` if you want to get *all* the output. This function will dump the token contents into a pretty-printed HTML stream, including the details of the thread *and* process tokens. I've found this tool invaluable in debugging secure Web applications. Here's how to use it in an ASP script:

```
<%= createObject("tokdumpsrv.tokdump").tokenDump(-1) %>
```

Where to Get More Information

IIS documentation ships with the product itself; just go to a machine that has IIS installed and surf to http://localhost/iishelp. As of this writing, the IIS 5 documentation is somewhat out of date, but one hopes that the next service pack will fix this. The Platform SDK also includes this documentation as well; I'd

brown10.qxd 7/5/00 11:38 AM Page 490

expect this to be updated sooner than the IIS product documentation, but only time will tell.

Summary

- Public key cryptography reduces the need for shared secrets.
- Public key cryptography does *not* reduce the need for trust.
- Signatures created with a private key can be verified by anyone, because the public key is not a secret.
- Certificates provide a way of authenticating a public key.
- At its essence, a certificate is a name and a public key bound together by the signature of a trusted third party.
- A certificate revocation list (CRL) is a published list of certificates that have been revoked for one reason or another; when authenticating, it's important to check the certificate against a current CRL before assuming it's valid. Both IE and IIS provide this option.
- Often a hierarchy of trust will be necessary for two parties in different organizations to develop trust in each other's certificates, which results in a chain of certificates, each signed by a more widely trusted authority.
- An administrator can install a certificate trust list (CTL) for each Web site to indicate the certificates (typically for root authorities) that he or she is willing to trust in a client certificate chain.
- SSL 3.0 is the de facto authentication protocol used with HTTP. TLS 1.0 is essentially just SSL 3.1 with the IETF's stamp of approval, and PCT is a dead technology.
- SCHANNEL is the name of the security service provider that implements the aforementioned authentication protocols.
- SSL runs in three modes: mutual authentication, server-only authentication, and no authentication (which is deprecated).
- HTTPS is simply HTTP over SSL on a designated default port (443). All communication over HTTPS is encrypted using a conventional session key exchanged during SSL authentication.

brown10.qxd 7/5/00 11:38 AM Page 491

- The IIS metabase is a hierarchical store of attributes that parallels the structure of the Web sites for a particular machine. Attributes in the metabase are normally inheritable to make administration easier.
- The Web Application Manager (WAM) was introduced to move Web applications into safer environments than the System logon session. These sandboxes not only protect the Web server from crashing but also significantly reduce the threat imposed by buggy Web applications and the crackers who exploit them.
- IIS provides a plethora of client-side authentication options, each of which has pluses and minuses depending on the environment in which it is used.
- IIS is often used as an HTTP gateway into a world of COM+ components. It's designed to make it easy to pass through the client's identity so that COM+ interceptors can provide role-based security checks.
- Whenever you place a network hop between the Web server and the COM+ components that perform role-based access checks, life becomes much less automatic, and you need to start thinking about how to get network credentials for that hop or how to move those access checks into the Web server process itself.

brown10.qxd 7/5/00 11:38 AM Page 492