

## Re: HELP PLEASE The request failed with HTTP status 401: Access Denied.

**Source:**

<http://www.derkeiler.com/Newsgroups/microsoft.public.dotnet.framework.aspnet.security/2002-11/2018.html>

---

**From:** nu-k-ar ([nospam@plz.com](mailto:nospam@plz.com))

**Date:** 11/27/02

From: "nu-k-ar" <[nospam@plz.com](mailto:nospam@plz.com)>  
Date: Wed, 27 Nov 2002 09:28:17 +0100

This article may contain URLs that were valid when originally published, but now link to sites or pages that no longer exist. To maintain the flow of the article, we've left these URLs in the text, but disabled the links.

---

-----  
Web Security: Part 2: Introducing the Web Application Manager, Client Authentication Options, and Process Isolation  
Keith Brown

This article assumes you're familiar with ISAPI, ASP, CGI  
Level of Difficulty 1 2 3  
Keith Brown works at DevelopMentor researching, writing, teaching, and promoting an awareness of security among programmers. Keith authored Programming Windows Security, to be published by Addison-Wesley in June 2000. He also coauthored Essential COM, also published by Addison-Wesley (December, 1998).

This article, the second of two parts, continues coverage of Web security for Windows. It introduces the Web Application Manager in IIS that allows Web processes to be isolated, decreasing the security risk associated with running in a logon session.

The article then picks up where Part One left off—it discusses authentication methods such as basic authentication, digest authentication, integrated Windows authentication, and anonymous logons, and the benefits and drawbacks of each.

---

The first installment of this article covered the basics of SSL authentication and certificates, and showed you how to obtain and install a Web server certificate and how to require SSL for certain files or directories (see <http://msdn.microsoft.com/msdnmag/issues/0600/websecure/websecure.asp>). This installment will introduce the Web Application Manager (WAM) and explain the importance of Web application process isolation. It will then describe the various options for client authentication and explain the type of security context that results for each option. Understanding the subtle differences in the resulting client security context is a key factor in determining how to architect an Internet Information Services (IIS) gateway into a traditional COM+ middle-tier suite of components.

Back in the old days of 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 ISAPI and ASP Web apps therefore ran in the System logon session, which is dangerous.

To understand the danger, consider the following C/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 ISAPI DLL running inside of INETINFO.EXE, and that the user actually typed 300 characters into the text box. Since GetStringFromTextBox doesn't pay any attention to how much memory buf points to, it will 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 will 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 (this becomes quite easy if she can reproduce the bug on her own machine) and point the return address into the buffer.

In the most elegant attack I've seen documented ([http://www.cultdeadcow/cDc\\_files/cDc-351](http://www.cultdeadcow/cDc_files/cDc-351)), 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 using a language that is safer 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 Trusted Computing Base (TCB) of the machine and the game is over. It's funny; I remember when people were merely 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, hopefully 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 of 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 Microsoft® Transaction Server (MTS) catalog as a configured component. This component is the WAM. Here COM's claim of local-remote transparency rings true; using a COM interface allows the Web server to interact with in-process as well as out-of-process WAMs polymorphically.

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: low (IIS process), medium (pooled), and high (isolated).

The low and the high options are exactly the same as what IIS 4.0 provided. In the first case, the application code runs in INETINFO.EXE. In the last case, the application code runs in its own dedicated process. What's interesting is the new medium option, 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 different than hijacking one outside of the TCB, assuming that the administrator has erected effective perimeter defenses around the TCB.

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 5.0 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)

Since 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-hosting applications that IIS creates by default when it is installed with the operating system: IIS In-Process Applications and IIS Out-Of-Process Pooled Applications.

The first application is designated as a library app, 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 IIS Out-Of-Process Pooled Applications, 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 1 WAM Apps Responding to HTTP Requests

Figure 1 shows the results if all of 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:

- IIS In-Process Applications
- IIS Out-Of-Process Pooled Applications
- IIS-{Default Web Site//Root/App5}
- IIS-{Default Web Site//Root/App6}

As Figure 1 illustrates, 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 is 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, remember that IIS doesn't attempt to synchronize your changes with the security database (of course, you can do this programmatically using ADSI). If you change this to a user name of Foo\Bob with a password of shazam, you had 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).

Before you make a change like this, 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

While Figure 1 is interesting, it doesn't show the full picture. Every WAM thread that performs work in response to an HTTP request from a client does so while impersonating an account. 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 ASP application (which is really just a stylized form of ISAPI) does so while impersonating somebody.

There are various ways that IIS can determine the identity of the client, and thus obtain a logon session for the WAM thread to impersonate. They are, in order of preference: anonymous (no authentication), certificate-based authentication, integrated Windows authentication, digest authentication, and basic authentication.

Each resource can have its own configuration for authentication, and you can select one or more of these options via the metabase. Last month I showed the metabase attributes for enabling or requiring client certificates. For the other options, the AuthFlags attribute (a bitfield) controls which authentication options are allowed. There are also three subflags (AuthAnonymous, AuthNTLM, and AuthBasic), but they are rather out of date. There is no corresponding AuthDigest subflag, and AuthNTLM should really be renamed AuthNegotiate because it now means SPNEGO (Simple and Protected Negotiation). Here are the bit values you can use with the AuthFlags 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 don't typically rely on the operating system to perform authentication. Instead it's done at the application level, frequently 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, and so on) can be accessed successfully 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 instead of trying a more expensive authentication option.

This means that IIS will impersonate the anonymous Internet user account and open the file while impersonating. If, on the other hand, 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, or file) in the IIS metabase has the following (inheritable) attributes:

**AnonymousUserName** This is 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** This is 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 gives you a logon without checking the password at all. This is a nifty feature, but beware: it's only supported for local accounts. If this feature alarms you, remember that INETINFO.EXE runs in the System logon session and is therefore part of the TCB. This is an example of a member of the TCB doing what 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 since 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 setting (which is typically the case), IIS does its best to cache a single logon session and impersonate that session for all of these resources, even across isolated application boundaries.

#### Certificate-based Client Authentication

Last month 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 the certificate is valid. This involves comparing the current date with the valid from and to dates in the certificate, followed by verification of 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 "Enable certificate trust list" box. This tells IIS not to trust any authorities except those 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 to trust.

As IIS validates certificates, it must check for revocation, which can be time-consuming. By default, IIS skips this step, but you can control it 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 zero forces IIS to check for revocation, which may 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 attribute.

At this point, another metabase attribute becomes useful: AccessSSLMapCert, which is really just another bit that can be set or cleared in the AccessSSL attribute that I mentioned in my last article. If this attribute is set to True, IIS will attempt to map the client's certificate onto a user account within Windows via one of two mechanisms: either IIS will use its own internal table of mappings from certificates to user accounts and passwords, or it will ask a 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 SSLUseDS-Mapper 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). 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.

If you want to grant access (using the one-to-one or many-to-one options), 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 doesn't provide for many-to-one mappings, which must be set up in 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. This 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, 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. To do this, 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.

Certificates can be the most secure way to authenticate clients

over the Internet when used properly, but there are other options available if issuing client certificates doesn't make sense for your particular application.

#### Basic Authentication

Basic authentication is the native authentication mechanism built into HTTP/1.0. The client sends an HTTP request to the server, and the server sends back a failure, demanding that the client prove her identity by sending a 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. That assumes, of course, you trust the server with your cleartext password.

If you attempt to turn on this authentication mechanism in the IIS snap-in, you'll get a very nasty warning that tells you not to do this unless you plan to use SSL to secure the connection. This is great advice. You should require SSL for any resource 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: interactive logon (the default), batch logon, and 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, since 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 (especially for domain accounts) than establishing a batch or network logon. 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; in my opinion you should avoid doing this, as I'll explain shortly.

One thing you should be aware of when using Basic authentication is that Web applications (either CGI, ISAPI, or ASP) can get access to the client's cleartext password via a server variable named AUTH\_PASSWORD. I know of no way to disable this, which is too bad considering that 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 of 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 5.0 and Internet Explorer 5.0, but it will likely have limited use as certificate-based authentication becomes 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 doesn't 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

reversible encryption as opposed to one-way encryption.

Probably the most dangerous thing 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 Digest authentication spec RFC 2069 (see <http://www.ietf.org/rfc/rfc2069.txt>) state their position as follows:

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 their 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 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 (see <http://www.ietf.org/internet-drafts/draft-ietf-cat-iakerb-03.txt>) to work around this problem, but it 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 authentication 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 the client's local interactive logon session will often be used directly.

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 CGI applications. All of these applications are ultimately derived from the basic CGI model that provides a set of variables describing the client and server environment and the request, along with input and output streams for reading and writing the request and response headers and bodies. The interface that exposes these variables and streams to the application differs significantly between the three application types, but the basic information provided is similar. Figure 2 shows some security-related

variables.

I wrote a simple script that echoes these security-related variables, and I ran it several times while changing the authentication options on the server. Figure 3 shows my results when running over HTTP. Note that I've omitted all the certificate-related fields because they were all empty.

Figure 4 shows the results of the same script invoked over HTTPS. The chart explains 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 the enterprise certificate authority in my test system quux.com.

#### 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 an account. The particular account being impersonated will depend on the mechanisms that I've just described. One interesting difference between IIS 4.0 and IIS 5.0 is that on the earlier platform calls to `GetObjectContext` failed with `CONTEXT_E_NOCONTEXT`, which made it impossible for the ISAPI DLL to call `IObjectContext::CreateInstance`. This was required in Windows NT® 4.0 to flow the impersonation token to the configured component being created (to perform role-based access checks based on the client as opposed to `IWAM_MACHINE` or `SYSTEM`). This requirement was removed in IIS 5.0, and the client's security context appears to flow correctly, at least according to my own experiments.

#### ASP Applications

While 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 I've described previously.

As I pointed out for ISAPI applications, in IIS 4.0, 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 of the calls to be denied (typical if your WAM is running under `IWAM_MACHINE`), or all calls to be allowed since `SYSTEM` is not subject to role-based access checks. (This happens if your WAM is running in-process in `INETINFO.EXE`. It's not a problem at all in IIS 5.0.)

So for IIS 4.0, it's critical that scripts use `Server.CreateObject` to instantiate objects, although 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 information in that certificate from ASP by using the Request object's `ClientCertificate` method. Here's a simple ASP script that echoes into a table all the attributes in the client's certificate.

```
<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>
```

#### CGI Applications

ISAPI and ASP applications run in the logon session of the client or anonymous user (or at least their threads run in that logon session via impersonation). CGI applications run the same way, except 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 into the System logon session. This attribute, known as `CreateProcessAsUser`, is set to `True` by default, which is where you should leave it. Like most metabase attributes, it is inheritable.

Setting this attribute to `False` causes CGI applications to run in the System logon session in the noninteractive window station. The noninteractive window station is not 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 device or 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 only as a gateway into a more structured environment built with COM+ components. From a security standpoint, this architecture makes a lot of sense and can help you avoid an overabundance of application-level security checks. The less security-related code in your application, the better off you'll be.

Since 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 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. Since the WAM is a configured component, it runs in a process where dynamic cloaking is enabled. 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 thread token; without cloaking turned on, COM ignores the thread token for outgoing calls. Things didn't work nearly as smoothly in Windows NT 4.0, as I discussed in my November 1999 Security Briefs column (see <http://www.microsoft.com/msj/1199/security/security1199.htm>). Figure 5 shows the natural three-tier architecture in which IIS acts as the gateway into a collection of COM+ components in the middle tier.

#### Figure 5 IIS Three-tier Architecture

If you plan on co-locating the Web server and your COM+ applications on the same machine (as in Figure 6), 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. Since dynamic cloaking is turned on in the WAM by default, when you make calls from your scripts into your local COM+ components, role-based access checks will occur naturally against the client's security context.

#### Figure 6 Web Server and COM+ Apps on Same Machine

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 7), you've got to make sure that the client's logon session on the Web server has network credentials 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 using a `NULL` session. If your client was in fact authenticated, you've basically just dropped that information on the floor.

#### Figure 7 Web Server and COM+ Apps on Separate Machines

Figure 8 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. It 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 (not using the anonymous account), unless you're using Basic authentication, you're not likely to have network credentials. You will therefore not 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 multi-hop 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 simply delegate calls to the remote object. The beauty of this model is twofold.

First, you shouldn't make 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 like Microsoft Visual Basic®, the Java language, 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, your role-based checks will now work as desired since the COM+ interceptors in the library application will see the client's security context. Note that this trick requires Windows 2000, as MTS library packages have no notion of role-based security.

#### Miscellaneous Topics

Here are 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. While 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 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** This 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 probably 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, the computer that hosts the Web server has been designated as trusted for delegation, and the client's account has not been marked as 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 planets that must align correctly for this to work that it may not be worthwhile to even enable this option, but it's probably no worse than a hardcoded user account.

When you hardcode a user account and password, IIS only verifies that the client can be authenticated according to the AuthFlags attribute for the target resource. This means if you've allowed anonymous access, everyone gets past this hurdle. Once the authentication requirement is satisfied, IIS ignores the real client's identity and instead establishes an interactive logon session for the user specified via UNCUserName. This of course, will 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 of INETINFO.EXE). While this can be dangerous, it can also allow you to do very powerful things. However, since 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 code shows how this can be done. I've packaged this in a nonconfigured, in-process apartment-neutral 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!
    // your code goes here
    SetThreadToken(0, htok);
    CloseHandle(htok);
    return S_OK;
}
```

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 replaces the token 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 in the System logon session. Rather it will typically be IWAM\_MACHINE, which is not very privileged at all, at least by default.

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 (<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,

including the details of the thread and process tokens, into a printed HTML stream. 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 software itself. Just go to a machine that has IIS installed and surf to <http://localhost/iishelp>. As of this writing, the IIS 5.0 documentation is somewhat out of date, but hopefully the next service pack will fix this. The Platform SDK also includes this documentation. I would expect this to be updated sooner than the IIS software documentation, but only time will tell.

-----  
For related articles see:

<http://www.microsoft.com/mind/1099/inside/inside1099.htm>  
<http://msdn.microsoft.comhttp://msdn.microsoft.com/workshop/security/default.asp>

For background information see:

[http://msdn.microsoft.com/library/psdk/certsrv/crtsvnode\\_intro\\_8f3n.htm](http://msdn.microsoft.com/library/psdk/certsrv/crtsvnode_intro_8f3n.htm)  
[http://msdn.microsoft.com/library/psdk/crypto/cryptovrvw\\_8395.htm](http://msdn.microsoft.com/library/psdk/crypto/cryptovrvw_8395.htm)

-----  
This article was adapted from Keith Brown's Distributed Security in Windows NT (Chapter 11) © 2000 Addison Wesley Longman, Inc. Reproduced by permission of Addison Wesley Longman. All rights reserved.

-----  
Keith Brown works at DevelopMentor researching, writing, teaching, and promoting an awareness of security among programmers. Keith authored Programming Windows Security, to be published by Addison-Wesley in June 2000. He also coauthored Essential COM, also published by Addison-Wesley (December, 1998).

-----  
>From the July 2000 issue of MSDN Magazine.

-----  
Send feedback to MSDN

© 2001 Microsoft Corporation. All rights reserved.

"Larry Hastings" <[greq.NOSPAM.@NOSPAM.unixsucks.com](mailto:greq.NOSPAM.@NOSPAM.unixsucks.com)> wrote in message news:uu7sbilkv7jsba@corp.supernews.com...

> Logon type if different for both events. One if local logon (logon 2) and  
> second one network logon (logon 3).

> Check if that user can access file shares on that machine from remote  
> location. Is there any other event messages about failed logon?

>

> --

>

> G

> <http://www.unixsucks.com>

> "John" <[jonashbaugh@hotmail.com](mailto:jonashbaugh@hotmail.com)> wrote in message

> news:O40022X1CHA.1356@tkmsftngp04...

> > I have turned on all security auditing and this is what I have found.

When

> > trying to connect using the code I get this:

> > Logon Failure:

> >

> > Reason: Unknown user name or bad password

> >

> > User Name: USerX

> >

> > Domain: VCNET

> >

> > Logon Type: 3

> >

```
> > Logon Process: IIS
> >
> > Authentication Package: MICROSOFT_AUTHENTICATION_PACKAGE_V1_0
> >
> > Workstation Name: WRK1
> >
> > When connecting to the site via a browser I get this:
> > Successful Logon:
> >
> > User Name: USerX
> >
> > Domain: VCNET
> >
> > Logon ID: (0x0,0x5FC98)
> >
> > Logon Type: 2
> >
> > Logon Process: IIS
> >
> > Authentication Package: MICROSOFT_AUTHENTICATION_PACKAGE_V1_0
> >
> > Workstation Name: WRK1
> >
> > Logon GUID: {00000000-0000-0000-0000-000000000000}
> >
> >
> > Is there a security policy I am not setting up correctly?
> >
> >
> > "nu-k-ar" <nospam@plz.com> wrote in message
> > news:ejxB5fWlCHA.2240@tkmsftngp04...
> > > 1.) turn on all your security policy logging
> > >
> > > 2.) check if u have an real login (means kerberos token)
> > > 3.) if u go to an remote sql-server (integrated sec), mark u're
> > > web-service
> > > server "for delegation" in the AD
> > > http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q283201
> > >
> > > 4.) disable integrated sec on the web-server cause integrated is
> > > stronger
> > > than basic
> > >
> > > 5.) if u go with integrated then u deliver a kerberos token
> > >
> > > isent it NetworkCredential.getGredential(URI)
> > > cred.user
> > > cred.pass
> > > cred.realm
> > >
> > > check your security logs...
> > >
> > > blaaa....
> > >
> > > .::[nu-k-ar]::
> > >
> > >
> > > "John" <jonashbaugh@hotmail.com> wrote in message
> > > news:#21tUjVlCHA.2036@tkmsftngp07...
> > > > We have basic authentication on through IIS. We are running a win xp
> > > > pro
> > > > server. When trying to access the site via a browser I get the login
```



M2(G6-(JTJ4ZE+9T2#!AQ0H-`A160.\$"Q &+`2:8:,&"A(2L#2/LR%)FRPT%
M`J':O\*C `XL1#A 2NF2IKU],801NZ"-L7+MVXXC]X5#P0Q]?W]:),\?L\$H^\$
M'<1H"M9-G3MWZ+H-PY1%0L(2CJRQ;.9&0 9U/3\_NT,/7;U],> 0R^#(J6CG0
MUU2!@4M@X+;8FP!(P19;6Y2!8KS%CH<)0L\$0D<)-!UD-30"#7\!-\_U=%8]ST
M'LEB%P. P]CT8D0(/HN-R B\ [>^H`&BS\_2,M!@-MP4U\_++&2`4%J\$#@-.N?]
M\$ILL+(BW'3LX&& `O/U) LU!&:CQCL\$QG.(0&:\$6!,Q%@\$Q'TFQM:#@SWQ
MDDN(XF@@4(8MYLC2. R\X)&.'ZT"0 ++`%G3B[&!:"(?-QKII!"C3%>.-M]L
M1\463K\*\$9)8?Z=(DESE.8EY-OT#1P007W,! (3^R<XLMVO\*!A1BK];;D=@]M-
M`Q>./4EBQ!.W]#<,'&G\$LETVL65A4!CQJ,/, \* 'H\4<(.2L:8P\$">;&=G3[#D
M\$, (4\$O;D38I\LA3\*0!A4&9LR#P@D`" ] .O\_310D2?"<0#!X4-U :VUE!T Z:
MPMA3-Q0,M,ATXVP`0\*DA:4'0+-,Q.5 D3K\*C33&L0,\*&#@<2Q,ETZXQ \$ CK
M-#B=\*Q21,=TY'BP[G1 \$H3(=%@2M&9MT.:(3S!XF"1L3=,L0)!<(&W\*DB(\$
M>;%NN\R"! 1! =, ;&!\$&-3+?\*F#J6\T<#S4P'S'4KN?@02X 0],7"[L86!,33
M)4'Q='Z\$DJ4NW4S'LCFQA:(R2@W\_-'\*`T7<DQ(OQZ;' ">> &1LH.(L\W<X#
MG1P;NRGW!+1 0M=\$]\$ 5QS8(`\$.HIC1+3 \<<D\&AP2U0%+W1+7/\5P-0-8L
M;2U0USU]#4 'CV#\_/\_+:09N?<T)H`M\$W3V)/)37=(=@. =TUZQ\7&\*MJ,\_<MU
ML#FM,\]3,YPXR[\$U\_CA+@APD00]J\*%(,\*),4#Z8UI`[G (MHJC\VVVYZK#/K0
M1>=]0 @Y:'&')G\$8),\$-E+BC8^/J"EX3X8:SA+CN04\G^G3:/),T2.A4=!!S
M. ?9B'0`6\*!/LT[<?GK05NVO=NXG;M%'"`@,,P, \*B"BO8S\*" "-+Q^9N+6L\^
M5[W00:]%ZIC&,I9!C79,9QIC2QM(H#= ZF'->@<,R2W(P25?M("#8)+@1RC8
MN:K51'\$8Y-IT[M"%+'7C!0!X@XDXI+G!I4]ZSMA^J6P0D"``X.U!\$R\_V8P
MD\$P0!:\* <!Y+2([`%1C.D,@B"NF,[%I3<<0`M%!+5K\$C4# 3B #4,1V7G\$`
M04SG!\283B\$(PJC8M.,#`+#%-\*1!QSI.@QHC&4@EJE%.U)#;G[@8Q\_I6 T[
M#"0`lZ@\$ ,X)8DV[4H@[ \* .@@27G&VCV#C#P@`P!T\$V<=I3,,&J\*!&'Y%AC&=4
M@PX\$F0(G.ZD,&R% `0F(I2P3H "T".0`L)QE+!4P`((48 &ZC.4" "F"0`ZA
M"6200QWF8 8HV\* "#2F!%>!0!S04H5@"^64"# ```<Q2`+B<91\*V\$()A\$F0`
MN=0E+&TEE7821 &!F,)" ;K&. (KCSGF"CP@K(@/^#"Y1A:R@@@QGR.(\$KN\*\$3
MZG@#!\*S0AB8(S E<N (6:C"&0K1 "4Z0P1EZ(! 9K&\$\*\$`L,G3APD'G6(+0A/
M<\$<W!I2&!BC#'=!"A0B@`8\_YM (!Q!@3)\*:8!0YI!QXG,\$,[J'&.8'1+I# )
MU!JF^(-=Q.,-+0"\$`#H1CU&8(!['F,\$ZP!&"-P@#"#X?9P \*BP(YME&"+6#!?
M%6P1CSAHXQU=.\$4\\_H#4F 3J!IJ(AQ',F 8F1 ,:N(C'0@P"W\$\$`Q=4`\$\$N
MOM&\*='3#`1)XACET``!8I34>(I#7(];1#EW8(A?2JBM\*9N2#3\2#"8:(AQR"
M\$0\VS"\$>GPA".Y\*1"#JSS\* 0\8@%#MQ1#@RP(AZDR \$+=A&/+# C'BGXK1Q4
M`P4Y[ (,HGU)+=S1@TVX8PF\$<,<9\ "/=#"' :>X@3K.H3QX7&(;\"#&-<\*A
MA'2XPR.BB(4[KH ,=Z `%>Z(PA\*N\9%@E)"Z\*/E "A2@@10PP (IL X,:H `
M%`1 \$O&PPQ(L\$8\:\$ '\$=A @DV @A2H@ ,>2(\$#1( "`W0@!:V2@ YHT\$L`
0.X4(O+C&-\*)!BDCJ)" ``.P``

end

begin 666 t2000.gif

M1TE&.#EA+ ``\_,3\_/\_/[F3@;>IK;6\_V:Z'<ZNBJI9Z;HU8-(QC2HQ=/UM@
M;2\:"BTP-B,3""(D\*1<8&P````# P ``````````````````````````````
M'``
M9&F>Z!F@RN"^^<"S'P6(:Z<G,+Y\*7N! ]IQQOXA\*(@\$D+D'9%\*9'/V%\$:%4UGU
M=Q4V%N!PN+&\$5=,OH&ZO(GM74Y6QLFSNGY#!ZRH[" ]\I\FQZ\*( \$F?BZ\$)X9I
M@(V/)BLG+6V5+F] :Y:5F"1VE)MLG2.,0X..<:=+I2.(1JA+KHIPL:I0:+\*P
M4K96N+Q<OD6SF:G"NI"/!@'+S,W.S]#1T 7(U9+5C)0Z"MS=WM\_WPPGVB:@
MH3RC(N4EY^@RZA#L).[O,/'S(\_7VES8E^2(>(!A(\$(\$!-@0\*\$CQ YE\P)V4`
MWOV50V(?BBDL"C\*V1.,AC"@\ELC5\2&5B":U/Z L!K\$DRY,N:W%\$(M+43"\$U
M6X\$DEW+.2IDM:??,\W/7S1\Y19 4^E)E3"0."@B82I5J@J+8T"3-2F(KUW4A
#````[

end

begin 666 websecure2fig01.gif

M1TE&.#EA+ 'S`/<`5F<I98:M!@E\_ ;?ZK,`6;@08[T@;L\$P>,9 @\M0C=1P
MHMF K-Z/MN.?P>>OR^R\_UO'/X'<[6OKO]:@F<\_'G[?CS]II4@\*%@B;>%I+Z1
MK8L[;I)'=ZALDJ]XF\6=M]2VR=O"TN+.V\RIP.G:Y)Q,C)%QIHY[K(N\$LHB.
MN>'E[#R]X67OT-,8&1QCXV>PY2DQYRKRZ2QS[. ^U\++W\K2X]'8YZNXT[K%
MV]G?ZW. E^CL\%H;?\_Y^Y&=IZ"KKS8[.VQW=08+"B Y-!,B'PT7%2TW=-GM
MZ:\*QKN+Q[NSV]/7Z^29%/CEG73-<4V:WI6"LFUF@D\$9^<4V)?\$!R9W"\JWG
ML(/%MHS)O);.P9\_2QX:EGK+;TK;/TKS@V,7DW<;DW<\_IXQHN\*:G7S%EN:=GM
MZ.+R[F^ZH\_7[^7B]H(# GBD\,2(Q)XG#FV>2=#A+.90)ED=90:S/DA8:\$F5V



ME5!I; ,RF#GS0`1LHJ1 # : IOATK4HE:- '%KS1C>6RM2F.O6I4(WJ4KUAU\*I :  
M]:IFBP;7GF&-KKGUJV -JUC'VE5A8/6L)>\$\$`WS `^"YU2H9^,(A-(%6@202  
M0= ;1BWVRM>^O6O@ WL7OU4G4\$&0!\O2V4^%ZNH"2H\$F>G+F4 X,96W6O:M  
M/JBJ82V(08',L)WL'\*E<DH2=]/E\$`L0"\*8+\$`i(>KO90KZ7/!C@006/\_=@1,  
MV+G@0&1C!BE<]K>\V8(/8%C7@;0"M.P\$1=R&50\*^^-!8QO\*, :C^ 1%BF0#\*N  
MS:X5\9<"(>;EB-WU# <PH#\_3"+\$#Q72L2 3\*Q7L=00EUN(+I\$ %\*DSAOEQ9  
M408\$<0,N:#-'W(PHOI2'S.E 1SH`!:Z8)'L2`V4GL@5QA?7,XXM5#& 4`>B\$  
M\* :0GCFIJ2W#D\$L%=1</U\$X++]1BK,QV>&(`J1<AI@S4H # W@XZDZ8BM\*DU  
M%]I&&M4(P!#%&(SG\$0\_J`\* >D)K')!<\*RRI+R&@Z)4O1J&]#9\"%:88@"E\$  
MN1Y8@,51BQR(8C> @A.L)@412. )/ ,9]\_\$\_X+S0V('4)"8O!"F?F0VLVC\*  
M.^<ZK\_F\*[7\*\*]!A,X]E,:HLVGJE!+1& =S "\$HP.@!7^\$(@U8K.A7\_AQ"^-(  
M7)2H[B3Q0&Q'8 ':\[#%PF]Q!4KH8I?\E)DTWBK!!U #:-/@!36"06)D!M%+  
MO! F,I Y`6D88X+)[/H#L^Y,?EYL\$'F\$N2 U;J:BB>8\$\*+QC#S0P8R#>06E+  
M,]1IF@XPQN8(`.TL)SIS=))TS#T/>.Q#.PFF#B"GQT]Y^C-, \D" (=]12"E4\$  
MH!?I2<LE,XF>\Z@GX/W^\_D,`L]ZD]-U\_9B BA4U3 #5,E' [= 'X[PV?@!JZ  
M9^V]L4%S3,(JE.J:W\_]N5;B#[ ,V\$O&9P3P%),-HY"DS64+DD@9Y 4!L!:V6K  
M! :4[IQ/L&9T]3(%X5HE9(#.GQ^68"\\) "OW4<87A\FZ<MV"J\$W^\_\$\_-#M2#  
M1U@)JI0A\$Z]X 6,L, ,E&B&#0 1"!H#0`B .\ ( N2.( )DE"Y%X#L0B\$7-9&O  
M:VE"/HM<&I)2)#'6[<XGSOA\$,=L@YH0-UQGD]9#CBP\_O, `FY, '2UAB#P\*#  
M@8DN<OD!U\* 3!\"!P\_ ! [KUONIN=%JK:L"/JCNR[\-8;FT@`E=OIH;;QP"?  
MXP2NWSR>R!JH#RB?R9RHI\$\H3B]=.OY'L<A%)<@L(B&]K>O?5I4X\_O@#[\_\_  
M^,=/\_O)\_/Q?7)]QU`GD/MP7 2[@5/\$\*&X(<V#!1