Global Information Assurance Certification Paper

Custom IIS Authentication and Access Control using ISAPI Filter

by Arsène von Wyss

Version 3.1

Abstract

This paper is about custom IIS authentication ISAPI filters. There is only little information around about the limitations, advantages and disadvantages of this method of authentication, and even Microsoft provides little information on this topic.

There are many pitfalls and limitations in the way IIS handles authentication and access control, which will be described in detail in this paper.

This paper will also describe how an ISAPI authentication filter can help fix these and includes information about a specific implementation as well as some source code to help any potential filter programmer with his own implementation.

Contents

Introduction

What this paper is about

Microsoft has invested much work to create a complete web server solution. Their web server, IIS (Internet Information Services), works hand-in-hand with the Windows operating system to control all security aspects. Different methods to do authentication and access control are shipped with IIS and therefore always available. But in the end, the authentication methods IIS provides always map to a Windows account which is then used to handle all the security aspects in relation with that particular request.

There are several situations, however, where this behavior is not really desirable. Especially hosting providers who use IIS as their hosting platform and want to provide multiple sites hosted on a single server as well as additional services like access to SQL databases with transparent authentication or to allow the domain owner to define his own user accounts and associated access rights will struggle with this model.

The good news is that this behavior can be changed. IIS allows hooking into the process of servicing any request using a custom ISAPI filter which does some of the processing required to make IIS act differently. The filter can hook into any stage of the process and therefore has almost full control over any aspect of the request.

This paper will discuss the aspects of such a custom ISAPI filter which handles authentication and access control. Also, tips and tricks as well as some source code showing key parts of my particular implementation will be available in the appendix to help anyone who wants to implement such a filter.

The information in paper will be most useful for ISP provider administrators.

The Windows ISP administrators wish list for hosting

The following key points are probably on top of an ISP wish list when trying to create the optimal hosting environment:

- The servers shall of course be secure against any kind of attacks (direct or by worms or viruses).
- Web sites should run in a sandbox[1] to prevent any interaction between sites which could be used to exploit or compromise the server or other sites.
- The server shall deliver good performance even if it is hosting a large number of sites simultaneously.
- Customers should be able to set up and easily use protected areas in their web sites, without the need of complex and error-prone application-level logic.
- Customers accessing databases should be prevented from storing any database access information (user name, password, etc.) in their script pages to better secure the database server.

[1] The concept is well-known with human children: put them in a sandbox where they can build and destroy their sand castles, but cannot access or affect anything outside the sandbox. On computer systems, the same concept is, for instance, being used for Java applets.

- Customers should not be able to log onto the machine through services which are not for direct customer use, like maybe telnet.
- The servers should be ready for load-balanced clustering of the web service, so that multiple web servers can serve the same content simultaneously.

The default out-of-the-box setup of IIS doesn't provide much of these key points. Especially IIS up to version 5 is very problematic, since many unneeded and unwanted services are installed by default and some cannot be disabled properly without special tools or hacks. If a Windows 2000 box is directly connected to the internet without a firewall, the latest patches and/or antivirus software installed, the system will be infected by Nimbda and other virus in the matter of minutes.

When multiple sites and customers share one web server, protecting the different web sites one from the other, thus creating sandboxed webs, is essential. There is no sandbox concept present, and I cannot remember having seen any documentation on how to properly separate web sites in such a setup - even though this would be important for improved security, as I will show later on.

Performance is not bad, and fortunately Microsoft has added many important features to IIS6 to enhance web service stability and scalability especially when used with several web sites in parallel.

Customer manageability of web site security is not very well implemented. While customers can be delegated the right to access the IIS console, this does not allow the customers to manage users and add new logins to their machine. Also, customers can do many things wrong in the IIS console and compromise server security and performance, and if they have access to a system component like IIS management, chances are high that they also have (unwanted) access to other parts of the system.

When a database is being used, the common approach seems to be to put complete connect strings which include user names and passwords into script files. If someone manages to read one of these script files and if the database is publicly available (and there are many, as seen by the impact of the Slammer worm in January 2003), the whole database contents can be read and usually also modified including schemes.

The way IIS works, users which shall be able to use FrontPage or any built-in authentication will be granted rights on the whole computer system. In fact, they usually have to have the right to log on locally so that IIS can properly authenticate the users. Because of this, telnet and several other services like file shares could be accessible when not blocked out using an adequate protection like firewalls.

And last but not least, Microsoft makes it hard to implement good load balancing on web servers, even though Windows 2000 Advanced Server and all Windows .NET Server versions have a well working TCP/IP load balancing built-in. I will go more into details on this subject later.

The classic IIS security model

How does it work?

The normal security is pretty straightforward. Whenever a request is received, a Windows user token is obtained through some sort of authentication and then used to handle all the security aspects of that request. This includes file access, impersonation of the user for script execution, and possibly also for database access if integrated authentication is used with Microsoft SQL Server.

The user account credentials are taken from the authentication information passed on by the web browser, or from the IIS Metabase in the case of an anonymous request without authentication information.

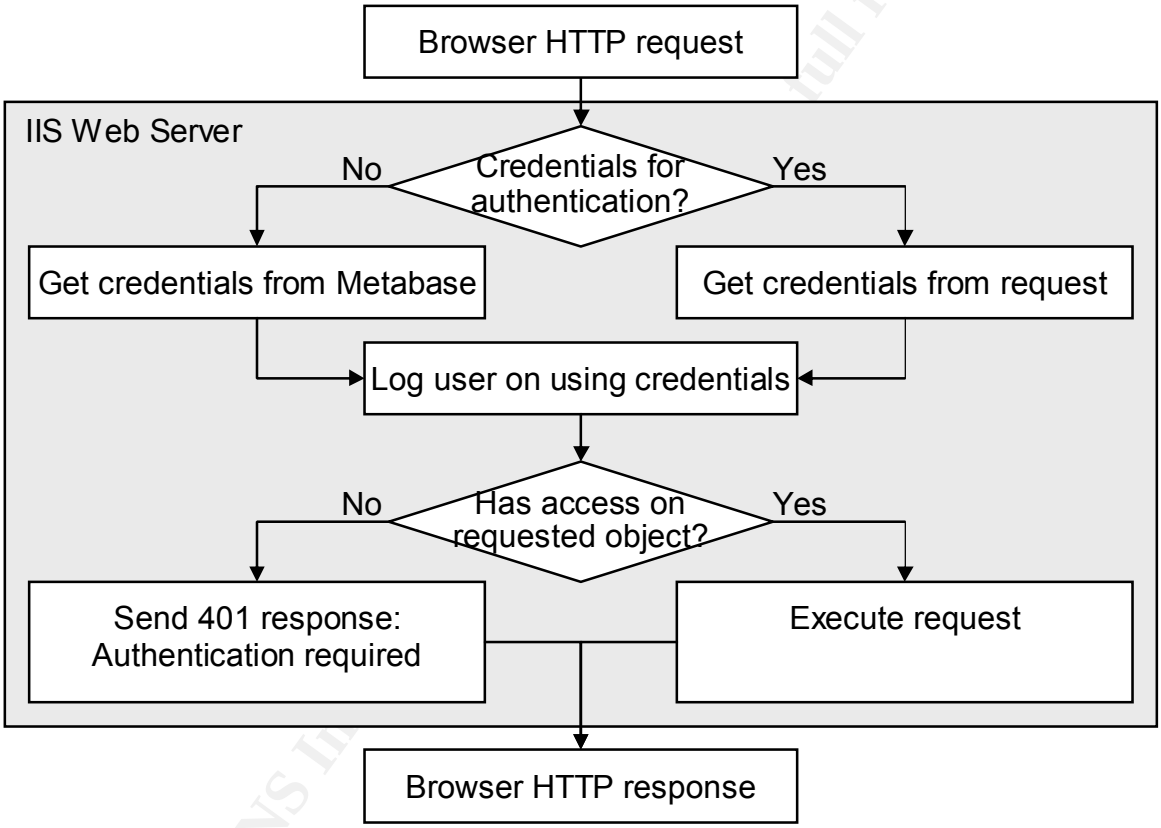

Figure 1: Simplified flowchart of the native IIS request handling

Let's have a look at the different built-in authentication types, including anonymous access.

Authentication types

Anonymous

This will be the most common case: a user tries to access a web page without authentication. Since there is no such thing as "no user" in Windows NT/2000/XP, IIS uses stored user credentials to handle the request.

This information is stored as plain text in the Metabase and can easily be extracted, for instance using a small VBS script[2]. This is disappointing, since any user which has enough privileges can just go and enumerate all passwords from all anonymous users used on the web server, even if IIS synchronizes the passwords[3].

Basic

User name and password are passed as BASE64-encoded cleartext string. This is pretty bad by the means of security, but it's the only authentication which every web server will recognize, and every web browser supports. Because of that, it is still widely used. And in conjunction with HTTPS (HTTP with Secure Socket Layer), it becomes a reasonably secure authentication method.

However, don't forget, there is a worse method of authentication which is commonly used: simple web forms[4].

Integrated security (NTLM)

When a Microsoft Browser is used to connect to IIS, NTLM can be enabled to authenticate the user. This method is pretty elegant since it allows transparently authenticating and impersonating users; no login dialog is presented if the NTLM authentication succeeds. Passwords are never transmitted as clear text. For more details on the aspects of NTLM, see the SANS Courseware for more details.

This transparent, password-free authentication is especially great in intranet environments, where users are always logged into their Windows workstations and therefore transparently keep their identity even if they browse the intranet web site. But in my opinion, it is quite useless and confusing when accessing web sites outside the Windows domain the user belongs to. If the user is not using a Microsoft browser, NTLM authentication will not work.

Another drawback is that is typically breaks over proxy connections, so that not even the login attempts with username, password and domain succeed, even if a Microsoft browser is being used.

Digest authentication

The digest authentication was introduced to replace the basic method. Passwords have to be stored using reversible encryption on the server, so that a digest can be created

---

[2] A script which does this can be found in Appendix A: Source Code Snippets, DISPTREE2.VBS (p. 27).

[3] This feature will make that IIS generate a random password, assigns it to the account of the anonymous user, and stores the credentials in the MetaBase for anonymous access. Therefore, the main benefit of this solution is that the administrator does not have to set up the password manually fort hat anonymous user account – but since the password can be read out of the MetaBase, this feature does not really enhance security.

[4] These forms are the de-facto standard on many sites and are implemented using the <FORM> HTML tag. This results in a HTTP GET or POST request when the user logs in, sending completely unencoded authentication data across the network and possibly even displaying this sensible information in the browsers URL. They are used on many sites to implement authentication and access control on application level using server-side scripts. This is often the only way of implementing security on normal web hosting services.

with the password and other shared information, allowing comparison of the MD5 hash of the result. If the exact password was not known, creating the hash of the digest would be impossible.

Storing passwords using reversible encryption is a pretty bad idea, since an attacker which somehow got the encrypted passwords could extract the originals easily. Because of this limitation, I have never even tried this method and I don't know of any web site using it for authentication.

Personal Certificates

When a web site is using SSL (Secure Sockets Layer) to encrypt the communications, certificates can be used to authenticate users. The functionality is pretty similar to the NTLM method since it also allows transparent authentication. Since Windows 2000 provides its own CA (Certificate Authority) service, issuing certificates is not costly or complicated.

This may be the most secure and flexible solution to authenticate corporate users accessing a corporate web server over a WAN. However, setup on the server side (issuing the certificate) and the client side (installing the certificate etc.) makes it unattractive for normal internet use.

Processing the request after authentication

Since IIS now has a Windows user context, even if the web user is anonymous, the user request will be used to control all security aspects of the request from now on. File access is done using this user context, so that NTFS ACLs (access control lists) apply when loading a static file (typically HTML, images, or downloads). This is, in fact, how IIS handles access control to the objects stored locally. There is no other way to specify who has access to what except for the possibility to remove anonymous access.

If a script mapping matches the extension of the object requested, IIS will launch the associated script engine or start the ISAPI extension or the CGI application using the very same user context.

Because of this, all subsequent access inside the script engine will also be restricted to the user rights of this user. Also, if MS SQL Server is being used as database server and if it is configured to use SSPI integrated authentication, there is no need to specify any user name to log onto SQL server, since credentials will transparently be passed on just as with NTLM. This is great, since it allows the complete removal of any (cleartext) usernames and passwords in scripts which would be necessary to connect to the database otherwise.

Standard configuration using local files for serving contents

As long as only anonymous access is being used, or if authentication is implemented on application level, it's pretty clear that just the anonymous Windows user account needs to have sufficient rights to perform the operations necessary for the web page to be servable.

However, imagine that the web site in question also requires a restricted area which is not accessible for the anonymous user but rather for a couple of privileged users. Note that FrontPage access already has to be considered as such a restricted area.

Therefore, we need to allow more users to read and possibly write to the web contents. If a privileged user accesses the web page, it is necessary to make sure that he also has rights to all contents which the anonymous user has rights to, or the web site will display errors to privileged users when they access objects which belong to the anonymous part of the site. Scripts and FrontPage may need to have write access to some files inside the web for counters, guest books, and other dynamic contents, making the right assignments on these objects somewhat complex.

FrontPage Extensions try to solve this issue (and others) by using three groups for each web site (basically users, authors and administrators for the site). Users are then assigned to the appropriate group. Rights on the objects in the web are assigned on a per-group-basis, therefore preventing to change effective rights on the objects.

However, this does not really solve the issue. What if we need finer control over the paths every user is allowed to browse? Also, if we use the great SQL Server integrated authentication, each of the users – even if they are in groups – needs to be added to the database server and its security settings or the database access will fail if a privileged user is browsing the site.

This is quire a mess and does not only add a lot of management overhead, but also usually leads to loosen security. In a typical configuration where the SQL server is not on a separate network, one of the privileged users could also use Query Analyzer or another SQL tool to directly connect to the database and extract sensitive information which would not have been available through the web site.

In the default configuration, IIS uses one IUSR account to manage anonymous access on all webs. This automatically allows any script running as anonymous user to access the data files (scripts included) of other web sites since they all have to allow the IUSR access to serve the contents.

If that were not enough trouble, Microsoft added some brand new problems with the introduction of ASP.NET, especially when impersonation is enabled. Since the .NET framework does not know whom to impersonate before the very first request, the system account is being used. However, subsequent requests are then processed using the impersonated user.

If well trained security staff is available, it is very well possible to fix most of these problems by carefully designing the network and by assigning rights on a per-user basis and by using workaround in the scripts. But most typical small hosting customers will not have the knowledge or the possibility to do this, so that the site security is just left unmodified in such cases.

Alternative approach using UNC file shares for serving contents

A little known fact[5], even though it is properly documented by Microsoft, is the different security behavior when a file share is being used for the web site contents instead of local files. In this case, authentication is done just the same as with local files, but the

[5] I can't remember seeing it in the SANS materials, even though is has serious impact on security.

actual security context used to serve the page is by default[6] the one used to access the UNC share.

Therefore, this behavior creates sort of a sandbox where you always know what user credentials will be used to access any resource, such as files or SQL Server in integrated authentication mode independently of the logged on user. Sounds like a good thing, but there are very important issues making this feature nearly useless in the default configuration. And because of these issues, IIS cannot take advantage of the possibility to offer load-balanced services with a central data file storage. The other solution to get this by keeping different server file systems in sync is very hard to achieve reliably with fast changing contents, especially with some multi-master replication solution which would be needed if the files on any web server must be writeable (because of counters, file-based forums, FrontPage, etc.).

Unfortunately, Microsoft seems not to have done very extensive tests with hosting sites on UNC shares, so that there are serious problems attached to this solution. Their root lies in the fact that the user token of the user which actually got authenticated is by default just thrown away by IIS and replaced by the user token of the UNC share. Therefore, while an authentication does take place, the newly authenticated token will never be used to access any resource.

- Because of this, it is impossible to specify which user can access what part of the site! And I do mean impossible: any user able to log onto the server can access any file on the UNC share, since only the share user is used to access the data. Even worse, because of this, any user which does have credentials for that server or domain can just log onto any web site which uses UNC paths and access any restricted page. By design, IIS uses file ACLs for fine-grained access control. The only flag which controls access which IIS takes from the MetaBase is whether anonymous access is allowed or not, therefore possibly requiring an authentication. But since the UNC token is used on file access, every user which can somehow authenticate can access everything on the share.
- A MetaBase setting can be used to enable pass-through authentication, but this feature has its own problems (for instance, it fails with most built-in authentication types) and its proper use is apparently not very well documented[7].
- FrontPage extensions (and possibly other extensions as well) are not set up so that this feature can be used securely. FrontPage works by using CGI applications and ISAPI extensions which all reside in a directory called _vti_bin. This directory must be accessible by everyone since things like the counter component and form mailer reside in there. Authoring is done via the two _vti_aut and _vti_adm directories, which are by default protected by file ACLs from being used by the anonymous user (IUSR_servername).

[6] IIS6 offers an option to always use the credentials of the logged-in user for UNC share access. While IIS5 seems to have such a setting in the Metabase, there is no GUI option to use this feature. When this option is enabled, the behavior gets about the same as in the local file scenario.

[7] The KB article Q214806 does describe how to enable this feature, but there is no information on how well it performs. After listening to the Webcast which discusses IIS 6 and NAS, I have the impression that the feature does have ist own kind of problems and was not exposed because of this.

But since the UNC credentials are being used, anyone who has access to the site (especially anonymous users) is able to author and administer the whole site without having to enter any password! If the ACLs are corrected so that they don't allow access to these two directories, no authoring is possible at all, since the logged on user is not being passed along and thus the ACL always locks any user out which accesses these directories through the web site.
The proper solution is to use the IIS console and to remove the "Allow anonymous access" checkbox on the two directories, leaving the actual file access rights without modification. Note that even the most recent FrontPage extensions have this problem.

- The exact behavior of WebDav would have to be examined, but I suspect it to suffer the same kind of problems as FrontPage does. I suggest disabling WebDav in this configuration to prevent any additional potential security problems.
- In the default configuration of a clean Windows Server installation, the file server as well as the redirector service will have problems servicing the requests on a busy web site, especially if it is using cached script files. This is due to the different resource demands of the web server compared to normal users accessing files via UNC. For instance, IIS uses file monitors to keep track of what files have been modified. This information is needed to flush files from the cache and properly reload them. The ASP engine also does virtually the same in top of that to know when to discard cached precompiled code. Quickly, fixed size client and server resource pools are drained, and because of that, subsequent network requests fail randomly. This problem can be solved by tuning registry values which control these services. However, this is not a security issue, so that I will not discuss this in more detail. Some valuable information on this topic can found in the examples in the "IIS 6.0:UNC and NAS WebCast" (link is to be found in "Appendix B: Links and References" on page 33).
- IIS 6.0 will address some of these problems, but not to the same extent as the filter does. More information on what it does can also be found in the "IIS 6.0:UNC and NAS WebCast".

The common practice when UNC paths are being used in IIS is, as it seems, to use some administrator account to connect to the network share, even if Microsoft does state that administrative accounts shall not be used[8]. In consequence, all scripts on a web page configured like this will be run in that very administrator context! The misunderstanding of the implications of feature can lead to severe security problems.

A separate chapter on page 12 will show how a filter can properly address this issue.

[8] See „IIS Security Recommendations When You Use a UNC Share" in „Appendix B: Links and References", „IIS Security", on page 33.

An alternative security model

The basic idea

With the thoughts from the previous chapters in mind, another security model would make more sense, especially in larger environments where multiple sites are hosted on a single server and if load balancing is desired. That other model would fulfill some points in the wish list by creating a sandbox for each web, therefore making security settings on the web very straightforward and secure. From the view of IIS, one and only one Windows user would browse the web pages of a single web site. This is achieved by creating one Windows user which is assigned to just one web site's anonymous user credentials with full access to just exactly this web site.

That approach makes the assumption that, from the system's point of view, there will only be the anonymous user browsing or updating the site. Therefore, only one user has full access to the web pages (including write), and just one user has to be configured to use integrated SQL Server access to the database as dbo (DataBase Owner). This user will, however, exclusively have rights on that web and on some absolutely necessary system resources, but on nothing else.

Moreover, since both the name of the Windows account used for this sandbox as well as the user's password is not public, the risk of intrusion other than through the web site is reduced to a minimum. And since the purpose of this account is clearly defined, the account can be locked down[9] using well known procedures for securing Windows. The actual implementation of the sandbox is explained in more details later.

The last element in the puzzle is an authentication and access control layer which will control the access to the objects inside the sandbox. This is where the custom authentication filter comes in. Actually, it is not only an authentication filter, but also an access control filter at the same time. This is illustrated in figure 2.

The filter communicates with an external service to process the custom logon and the access check on the object requested, therefore allowing integration with a per-site (or domain name) user management which is completely independent of Windows accounts.

[9] We chose to do this using an organizational unit in the domain group policy, removed all file access rights except on the site directory and the required temporary directory. How the account is secured exactly depends much on the needs and the structure of the organization where the site is being deployed, but the concept of the sandbox is there in any case.

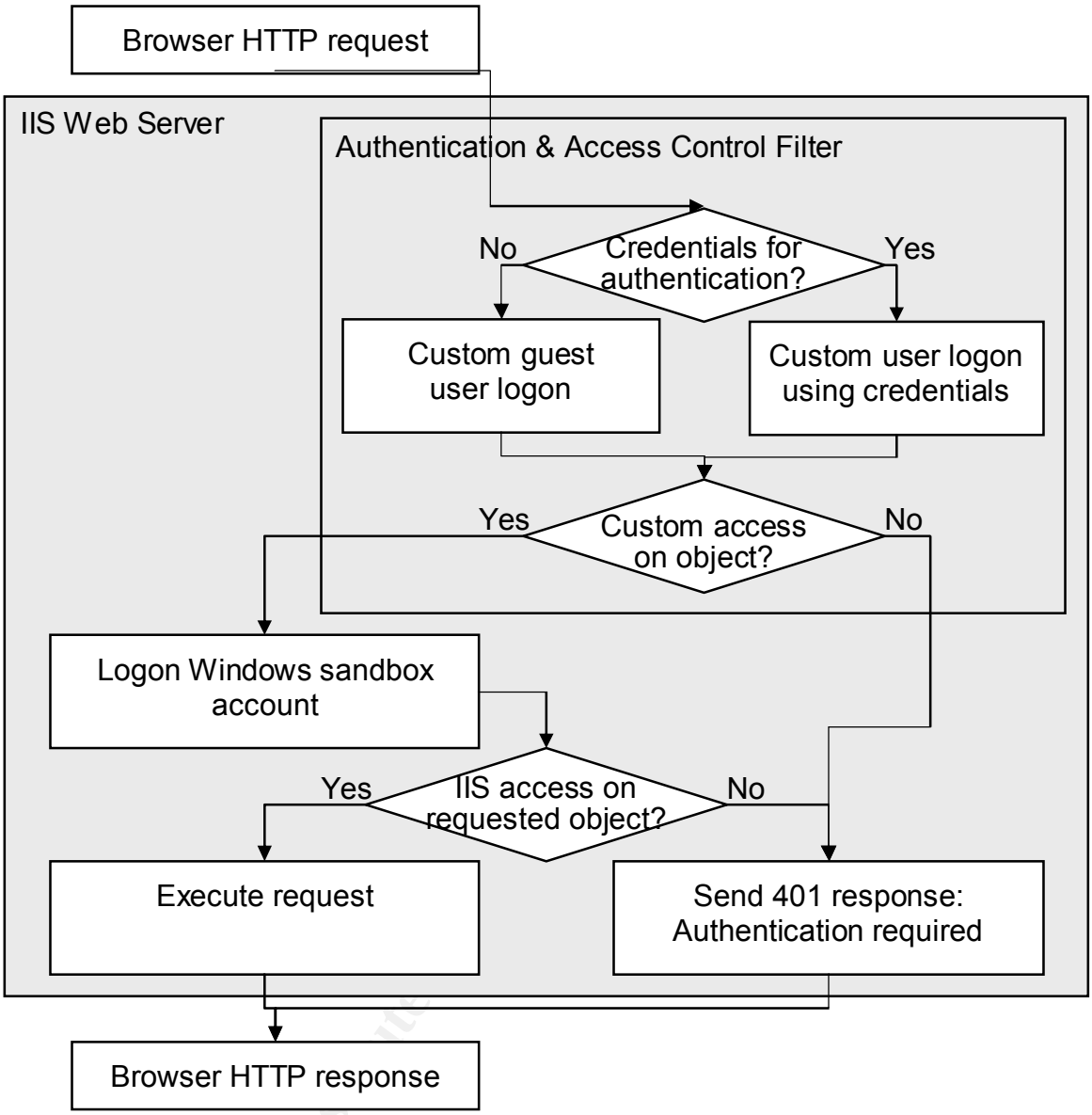

Figure 2: Simplified flowchart of request handling with an authentication and access check filter

Security considerations

- Sandboxing in this case works on both webs running on UNC file paths as well as on ones based on local files. You will find more details on this in a later chapter. Since the user login credentials processed by the filter are completely independent of Windows accounts, there is no possibility to use these credentials to gain access to anything outside the sandbox. Therefore, even in the case of a failure or exploit of the filter, the security of the infrastructure would not be compromised.
- SQL Server can run in native Windows authentication mode and use integrated security. This allows to effectively get rid of all database user names and passwords from script files without any security compromise or usage drawback. Only pages in that web can access the database as owner through the sandbox user, and no other database can be accessed. And, last but not least, the web

site authors have easier to understand connection strings to open the database connections since no credentials have to be included: since it is even possible to define a default database for any SQL user, the connection string does not even have to contain the actual database name.

- User management can be delegated to customers without having to grant them rights on any Windows account or even the domain infrastructure. In fact, since the users do not have direct access via Windows accounts, the domain structure is untouched from the customer's actions in the user management. This is a great benefit, since it allows customers to manage their own web site users without having to have access on the Windows domain.
- The service handling the access control may have blocking rules which are always applied. For instance, we block all guest access to the authoring directories of FrontPage. Dangerous URLs (for instance, referring to some well-known extension which is critical like EXE or IDA) can also be blocked using a globally valid rule. This lowers the risk of security breaches.
- All those well-known exploits which try to access executables on the server do not work (Unicode path traversal etc.), if the base directory of the web resides on an UNC share or on another drive than the system, and the sandbox user only has rights to this one directory. This stays valid for any possibly not yet discovered vulnerabilities of the same kind.

Drawbacks

- The security service handling the requests of the filter may be a single point of failure. If the filter fails to get information from the security service, it must deny access. Therefore, if the security service is completely down, all web sites on all web servers are also down. This problem can be solved by using several security service servers and distributing access via load balancing or any failover technique of your choice. I'll write more about that later.
- An exploit of the security service would in the worst case allow an attacker to add, change or remove rights and users on any web site. Since the security service doesn't need any special privileges or rights on the infrastructure, there is little risk of any more damage. Moreover, since the security server is best placed behind a firewall in an internal network, risk of a successful attack is very small (a malformed URL passed on by the filter would have to fail the security service).
- Unfortunately, the IIS FTP service doesn't seem to allow any hooking or custom authentication. Therefore, you cannot use the built-in FTP server with the user accounts of the security service for authoring. Of course, a custom made FTP server does resolve that problem. There may also be 3rd party FTP servers which allow external authentication modules, or use text files or a database for user authentication.

How the sandbox works, and why it also works with UNC paths

The sandbox user

The filter cannot change the behavior of IIS on both local files and shares. But, fortunately, this is not necessary in order to achieve the sandbox. We have to keep in mind that the filter adds an additional authentication and access control layer on top of IIS and therefore allows combining both the filter security with IIS security.

To implement the sandbox, a single Windows NT user is created which is used for this web site only (I will call this user account the "sandbox user" from now on)[10]. Let's also assume for now that IIS will somehow always use this user account to handle any requests on this particular web site.

Rights of the sandbox user

The sandbox user has to be restricted security-wise to the very bare minimum necessary to fulfill all its tasks. Since the scope of the sandbox user is very well defined, it is not too hard to create a sandbox user.

File System rights: This user will have read and write right on the web site files as well as on virtually mandatory directories like the TEMP directory, but no access at all on any other part of the file system. Therefore, the sandbox is in effect for the file system.

Local System rights: Using Group Policy, you can also restrict the other rights and therefore take unneeded rights away from the sandbox user (like, log on as service, etc.). When this is done, our sandbox is also secured in this respect.

How to tell IIS to always use the sandbox user as security context

It's much easier than one may expect: just make sure that only the anonymous authentication is enabled, and specify the sandbox user credentials as credentials for the anonymous user. If UNC paths are being used, also use the sandbox user credentials to connect to the share which holds the web files[11].

That's it! Now IIS will always use the sandbox user.

[10] Obviously, there will be a Windows NT security group which carries all these sandbox users, so that restrictions defined in group policy or the file system automatically apply to all sandbox users. However, for the sake of simplicity, I will only talk in the singular form of a sandbox user.

[11] It is possible to use a single network share and still keep things secured by the proper use of ACLs if you do it like this:
- using the ACL editor, assign the special right „browse folder" for everyone to the root where the web files are, but restricted to that folder only (not the files nor subfolders). This is needed so that directory monitoring works properly.
- assign full permissions for the sandbox user on the directory of this web site.
Using this setup, the share can be used properly, but sandbox users cannot even enumerate the files in other web sites, nor do any other access on these directories.

How to handle exceptions

There may be situations where a certain site needs to run in a context which has more rights than the normal sandbox, for instance for some web site which is used to communicate with a piece of hardware (maybe a web cam, etc). In this case, just make a customized sandbox user (which is consequently not in the same group as the other sandbox users) which has the necessary rights.

Implementation

Filter overview

The ISAPI interface does not only define the well known ISAPI extension which is used as binary to handle specific page requests (for instance, script engines are ISAPI extensions), but also the ISAPI filter interface. Filters are installed per web server or site and can hook into several stages of the request handling. For our filter, we have used four hooks:

PreProc Headers hook

This hook is called when IIS has received a request and assigned it to a specific web site, but hasn't handled the rest of the request yet. The filter has the possibility to read, change, add and delete header lines and therefore modify the request headers before IIS starts interpreting them.

This is the moment for us to extract and remove any "Authenticate" header which may be present in the request. By removing the "Authenticate" header, we can guarantee that IIS will not be able to do any authentication itself, even if there are any authentication modes defined in the MetaBase. This will never happen anyways if the web site has been properly set up.

The filter decodes and stores the authentication header for later processing in the AuthComplete hook.

Authenticate hook

This step represents the mapping of any authentication used to the Windows account which will be used by IIS for further processing. This means that the filter has to return a Windows account name and its password, which will then be used by IIS to authenticate and then further handle the request. If both user name and password are left empty, IIS will assume anonymous access and use the name and password stored in the MetaBase.

Since we use the IIS anonymous user as sandbox user, the filter always returns both an empty user name and password in order to have IIS use the anonymous user credentials.

AuthComplete hook

At this point, IIS has finished initializing the request including Windows user account mapping. The next step is starting to process the actual request. But before that, a hooked-in filter can do access checks and abort the request if necessary.

At this point, the filter gathers information about the request: cleaned URL (without query string), SSL information, physical directory, the site ID from the MetaBase, and user name and password retrieved from the previously stored "Authenticate" header[12]. All this

[12] We currently only support basic authentication, but may add other authentication methods at a later time as needed. This is not a technical limitation; any authentication method including certificates can be implemented.

information is then somehow transmitted to the security service, which returns an error if access is denied (bad username/password combination or unauthorized resource) and if access is granted, a user identifier which the filter puts into a header so that an interactive script can easily find out who is logged on for this particular request.

EndOfNetSession hook

This hook is being called when the request is over. We use it to release any resources allocated in any of the previous filter hooks to avoid any memory leaks. Therefore, this is not a security relevant hook.

Security service overview

The filter is just a small and fast gateway component to allow an external service to handle login and access checks on web requests. The security service will do the actual processing and basically needs to process the information received from the filter and return a simple allow or disallow response. In addition to that, in the case of authorized access, a user token has to be returned to identify the user used in the request.

Since we wanted to have a universal service which can also be used to handle authentication and access control on other custom services such as e-mail and FTP, we chose to use universal interfaces and protocols, such as XML; HTTP; SOAP, etc.

Request processing

Whenever the service receives an authentication and access check request, it tries to log in the given user. If login fails (especially when no credentials have been specified), a special guest user is used for further processing. The next step is to analyze the request and to check access to the path specified.

Security model of the service

To allow efficient, yet easy and robust right distribution, we defined right groups which contain rights. Each of these rights could be allowed or disallowed for a path. Rights would also apply to any sub path of the given path except if explicitly redefined otherwise. See the examples below for a better understanding.

Path: http://domain.com	User: Guest	Allow Browse
Path: http://domain.com/_private	User: Guest	Forbid Browse
Path: http://domain.com/_private/fun	User: Guest	Allow Browse

This would allow a guest user to browse to any location in the domain.com web site, except for anything inside the _private directory. However, the fun part in _private would also be accessible.

The right groups are collections of rights which apply to a specific resource. For instance, on a web path, other access rights apply than on a mail path. The group which is relevant in this case contains at least the following rights: Browse, Read, and Write.

Between rights, implications can exist. This prevents strange and hard to find access troubles. Therefore, if any user has the right to read a path, he may also browse the path. And if he is allowed to write to the path, reading and consequently browsing are allowed as well. If we forbid browsing on a sub path, he won't be able to read or write to the path any longer either.

Implementation details

Filter main code

The filter is a DLL written in Delphi. I chose Delphi as programming language since it has great dynamic string handling and therefore does good prevention against buffer overflow problems, and because I know it pretty well. However, the actual choice of the programming language does not really matter as long as the language can create the proper DLLs with the required exports for the ISAPI filter interface.

The code uses several custom helper libraries like an ISAPI filter API object wrapper, MD5 hash calculation, and HTTP communications. Since all these components are not within the scope of this document, I will not elaborate on these. Please note that due to these unavailable libraries, you will not be able to directly reuse the code snippets provided without modification. The objective of the snippets is only to provide you with a good understanding of the implementation.

Snippets of the source code can be found in Appendix A: Source Code Snippets.

Communication of the filter with the security service

The filter needs a fast yet flexible method to communicate with the security service. We chose to use SOAP over HTTP for best interoperability. Since the filter always uses the same request, there is no need to use DOM or any other XML creator code. Instead, we have a fixed template in which we insert the different values as needed. This makes sure that the requests are computed fast without much overhead.

HTTP is a thin protocol, and with the keep-alive extensions (HTTP/1.0 with Keep-Alive header, or the default communication in HTTP/1.1) there is only very little overhead and latency on requests since the connection may remain open after a completed request.

Communication is not encrypted except for the password which is only transmitted as MD5 hash. Encryption of the complete request would add latency and therefore slow down all requests, and since the communication is purely internal between two servers, there is no great benefit in doing so. However, one may of course also choose to transparently secure this communication via IPSec.

The security service

When a request is received, the security service performs the requested logon and access check and returns one of the following:

- A NULL value, which means access denied. We chose not to return any detail about the nature of the denial, since doing so would enable an attacker to better target the trials to gain access.
- A guest user token, which is a normal user except that it has the least rights. In fact, the guest token is identifiable as such and restrictions can be enforced. We use this feature to make sure that no guest can use the online editing tool, since we decided that a guest cannot do any editing for security reasons.
- A full user token, representing a logged on user. The token allows identification of the user, but also has a limited validity time. I'll elaborate about the kind of token we chose to use later on.

We currently use a SQL database as backend to the service. The database has tables for users with their password hashes, login names which are mapped to a specific user and therefore allow one user to use more than one logon name, and the access lists which define user-right-path-combinations. The definitions of right groups with their rights and rights dependencies are also placed in the database.

The service is designed as .NET remoting server (RPC) which listens for HTTP connections on a custom port. We use ASP.NET for interactive web pages, so it is obvious that writing the security service using the .NET framework greatly simplified interoperability with the different modules of our web sites.

The user token

The service is called using the following interface (used programming language: C#):

```
IUserAuthority ISecurityManagerForWeb.LoginAndCheckWebAccess(
    string userName, string password, int siteNr, string siteRootPhysical,
    string uri, bool isSsl)
```

As you can see, the return value is not just an identifier, but a reference to a user interface. The IUserAuthority interface defines the properties of a user, such as login names, display name, but also allows to do access checks for that user. The idea is that this user token can be used for server-level user identification with high comfort when used in an environment like ASP.NET (or probably any other language which supports deserializing objects from SOAP), and it may also just be used as token to identify the user by other script languages.

The reference does not point to a local object, but rather to the user object “living” on the security service. Therefore, it is impossible to tamper with the user object data from within the web server, unlike application-based authentication techniques where a maliciously uploaded web page usually can alter the whole user database.

The validity lifetime of the user token is limited. This is due to the design: there can be a bunch of different tokens around for the very same user. Therefore, if the token is not actively refreshed using the security service, it will be invalid sometime later on.

Since the user token is not cookie-based and cannot be injected by the client (not even by defining the UserToken header in the request), there is hardly any tampering possible from the client.

Enhancing the filter

Performance

The implementation as shown in this document requires requesting a new user token for every request from the security service. This slows down performance somewhat, especially when a large number of small static elements are being requested. To avoid this, the server and filter could be implemented with some caching of access information for a specific user token.

More precisely, the server could return an allowed path (called scope later on) and zero or more forbidden sub paths (called scope exclusions later on) which define where the returned user token does have access. If we look back at the example on page 15, this would mean that for the request on the path “http://domain.com/page.aspx”, the service would return the scope “http://domain.com” and the exclusions array containing one

entry “http://domain.com/_private”. The response time-to-live should be reasonably short to minimize the risk of users still having access to objects right after the rights have been removed, I’d suggest about one minute. This is long enough to usually handle all subsequent requests for images, style sheets, script files and other static elements without any additional roundtrip to the security service.

Security

Native browser authentication methods which are more secure than Basic Authentication may be implemented and/or enforced by the service to avoid the risk of username and password snooping. However, as long as only Basic seems to be the commonly supported authentication across all browsers and operating systems, this is not really an option using today’s browsers.

Also, the SOAP data transmitted to the security service could be encrypted to avoid eavesdropping. However, the communication should only occur on a private network and the most dangerous payload – the password – is always transmitted as MD5 hash which cannot be used to derive the original password[13].

Accessibility

As stated before, only basic authentication is currently supported. Apart from other native browser authentication modes, the filter could also implement completely different authentication methods. Since directly reading and writing data can also be hooked by ISAPI filters, it would be possible to implement any authentication you can imagine on filter level without any requirement from the web application – working just as transparently as the native authentications.

For instance, a web forms authentication could be implemented which uses cookies. The application would receive just the very same user token as with native authentications.

Optimizing the Security Service

Performance and availability

As mentioned before, a slowdown or failure of the Security Server is affecting every request on any of the sites which use the filter. Therefore, using a load balancing technique would allow both an improvement in performance and availability.

However, precautions need to be taken when using load balancing with a shared IP address. Since any server may give out a user token, which is effectively a reference to a user object residing on that server’s Security Service, it must be enforced that the reference connects to the correct server if used later-on. The solution is to have a shared IP for token distribution, but to insert server-specific connection information for the remote reference, so that one reference is always handled by the same server.

[13] As far as I know, the only currently known approach to find a matching string for a MD5 hash is to do a brute force attack. This makes it virtually impossible to get the password for a certain hash if the password is not extremly short. Also, if the used charset (especially ASCII or Unicode) is unknown, it gets even harder to find a matching MD5 hash.

User restrictions and lockout

Since every access to the web site is controlled by the Security Service, enforcement of access restriction rules can easily be implemented. For instance, a company could forbid access to intranet parts of the web site to normal employees to prevent any possible abuse of the company's services during non-working hours.

Another relevant feature may be the user lockout when a user's password has been entered wrong several times in a row[14]. This is a feature that is actually already implemented in my implementation of the Security Service. Even a very loose lockout policy can help to prevent successful password guessing: if the account is being locked after 3 wrong passwords and only unlocked after 5 minutes without any wrong password entry, guessing becomes very slow and therefore hard. And the fact that the service does not give a reason for access denial makes it even harder for guessing by humans or robots. Even a guessing robot would probably not be very successful doing a brute force attack with a maximum of 36 guesses per hour…

Logging and Auditing

Another feature which might be useful is user login logging and access auditing. Note, however, that auditing would limit the cache functionality as described in the "Performance" section on possible filter enhancements: each directory with active auditing would require that it was in the exclusion list to prevent caching, so that the Security service is called on every request and auditing can take place.

[14] Windows 2000 actually also allows the use of password lockouts for windows accounts. The lockout policy is defined in the computer security group policy. However, it seems that this policy does not apply to all kinds of authentication, so that it is mainly useful for machines where users log in interactively.

Possible real world scenario

Fictive situation

We have an ISP which provides top-level IIS hosting in complete packages and uses a filter like the one described here for security. Redundant web servers without web data replication delay, two-site online authoring tool[15], and the possibility for customers to define custom users and protected directories. This ISP uses a centralized storage file server to hold all user files. This allows centralized backup as well as using web servers without any user data on them.

John is a website manager of the fictive startup firm "Online" with the web site "www.online.com". He is a client of the ISP mentioned and maintains a large web site which must be updated often.

To accomplish this, John has now hired a new coworker, Karen, whose task will be to update existing pages and prepare the new contents for online publication. Karen will not, however, actually publish the pages herself, but John will do this after reviewing them[16].

After hiring Karen, John logs into the User Manager using his user name "john@online.com" at the ISP's web site and adds a new user, Karen. She is being assigned the user name "karen@online.com"[17]. John assigns her "OnlineEdit" rights on some directories which she shall keep updated, allowing her only to edit certain parts of the site.

Whenever Karen has made changes that need to be published, she gets in touch with John, who reviews the changes and publishes them. After some weeks, John finds that Karen shall be able to publish some of the pages she has access to. Therefore, he logs

[15] One of the sites is a restricted test site where editing is enabled. The other site is the actual public production web site which cannot be edited directly. Single pages from the test site can be published to the online site, or unpublished. This allows making changes which can be reviewed before putting them online, while still using exclusively data which is stored online. This helps avoiding problems like version conflicts if several people are working on the web site or even if a single person is using more than one computer with copies of the web which are not in sync.

[16] Since editing a page in the two-site online authoring tool does not automatically make it visible to the web site visitors, editing and publishing can be treated as two different tasks. In fact, in such an environment, the rights shown in „Security model of the service" on page 15 can be extended to also include the following rights: BrowseTest (user may access test site), OnlineEdit (user may edit pages, implies the BrowseTest right), and OnlinePublish (user may publish contents from the test site on the production site, implies the OnlineEdit right).
Note that the OnlineEdit and OnlinePublish rights are not being checked by the filer via Security Service, but by the online editing software. The user token allows the online editing tool to interactively check these rights. However, the Security Service can make the distinction between production and test sites (for instance, by using the host header) and check the Browse and BrowseTest rights appropriately.

[17] As you may have noticed, all user names have the domain name attached. This enables an ISP to allow any syntactically correct user name for a given domain without the possibility for the names to conflict with existing names of other domains. If the e-mail addresses are the same as the user credentials, it makes it even easier for employees to remember their correct credentials since they have a single identity for both logon and e-mail.

into the User Manager again and gives her OnlinePublish right on the directories she shall be able to publish herself.

Later on, management wants a guy named Peter from the PR department to be able to review some of the newly created pages before they get online. To accomplish this, John gives Peter BrowseTest rights on the parts he shall review before publication.

Some of the benefits resulting of the use of an authentication filter

In this example, the filter allows using UNC paths for the web site, enables the full functionality of the online editing tools by supplying the user token, and also adds all the security benefits of sandboxing and non-NT-accounts for authentication. The ISP's administrator does not have to care about any access right issue, since John can take care of the settings without any security risk for the ISP.

If a filter was not used, there would be two possibilities to implement security:

- NT-based accounts. This would practically prevent John from being able to do the changes, since file ACLs have to be edited and user accounts must be created. The ISP will rather not expose this much of its internal infrastructure.
- Application-based authentication (Forms, etc.). This would require MUCH more application logic to implement security, and in fact, all responsibility for security would be in the web site programmer's hands – and web site programmers are usually not trained to implement security. The added complexity of the code would very probably also affect runtime performance.

Note: If John wanted to use FTP, a custom FTP service would have to be used which allows separate user accounts (or even better, some sort of integration with the Security Service). However, if a web based file manager was used, the filter could handle the authentication just fine.

History of this filter implementation

The filter has evolved through different stages to get to what it is today. Let me give you some background on the creation of the filter, which may also be useful if you consider writing your own filter.

I work for a small hosting company which provides much customized services to the customers. As you will see later in this chapter, there were several different reasons that led to the development of this filter. This filter has therefore not just been designed and written for this paper, but for a real-world situation.

IIS Annoyances

We started out using IIS configured with what I assume to be the most used configuration with local files and basic authentication for public sites and NTLM on sites where only domain users would connect.

The behavior of IIS in this configuration is described in the chapter “The classic IIS security model” on page 3 of this document. We had trouble to get file ACLs right, since removing the inadequate “Everyone” group on public objects would result in logged-in users not having access to these public objects even though anonymous users had that right. Even if the access lists for each object was corrected, the FrontPage extensions would not respect this when the web was being accessed via FrontPage or Visual InterDev and sometimes completely replace ACLs when the server-side FrontPage Extension fixing was called.

The suggested Microsoft way is, as written before, to solve the problem by using three groups (Browse, Author, and Administer user groups) for each FrontPage web. This would solve this problem partially, but makes it harder to manage user rights and does not help at all for any user which shall be able to access the web in a protected area but does not have author rights. So we didn’t even bother using that setup.

Also, webs accessing SQL Server and providing secure areas required the use of native SQL authentication and cleartext passwords in the script files. Usually the login was moved to a include file. Therefore, anyone which managed to read the raw include file could do about anything in the database.

And because of the machine-global IUSER, in the case of a site hack where an attacker would be able to modify files on a web site (or if any of our customers wanted to snoop in foreign web site data), let me remind you that he could use scripting to access any of the other sites on the same server and therefore in fact access any web site’s source code and open any web site database. Thank you, Microsoft, for this great security design on servers hosting multiple sites…

Web sites with public, automated user registration

One of our customers hosted a site which should allow users to register and log in. Instead of some application-level web forms authentication, we wanted to use a native browser authentication. To accomplish this, we created an OU in the Active Directory, and used a special anonymous user in IIS that was delegated the necessary rights just in this OU so that new users could be created on the fly. This worked well, but the idea that any user on the internet may create new objects in our Active Directory was not very nice for me as administrator.

First tries with filters

To get around all these limitations and problems, we designed the web site sandbox using a separate anonymous user. But for this to work properly, we had to have the same user credentials no matter whether a user was logged on or just a guest. That was the requirement to the very first version of the filter.

The first release of the filter would directly access an SQL database which held the user and access information, and store a user GUID as identification in the header. This worked, but the use of COM objects in the filter was not very stable and required COM initialization for the threads in which the request ran, making the whole process somewhat complex and slow, since connections could not be pooled.

Also, debugging the filter proved to be a difficult task[18]. So the idea of a simple and fast filter which would authenticate to an external service was born. This would enhance the stability of the IIS servers and allow better debugging and securing of the security service. The service would then also be available for queries at application level to provide user information and management.

Load-balanced clustering with the filter

It is a great feature of the server that files can be hosted on an UNC share or on a local file system without any difference from the security point of view. After migration to UNC paths for web sites, we were able to enable load-balancing without any modification on the filter or web sites, without compromising security.

Why this strange user token?

Especially, the whole security had to integrate well with an application-level online authoring system. This system needs to have access to the user which is logged on to initialize the editing features for the user and also when the user chooses to edit a web page.

At first, a GUID was being used instead of the remoting reference, but this made it necessary to convert the user's GUID back to a user object on every request, thus requiring an additional roundtrip to the security server. The use of the remoting reference therefore enhanced performance.

Since the user object is only available through a remoting reference anyways, this very reference may also be passed on to other services or even other web pages on other servers, while remaining valid. This allows our online authoring system to use the credentials of the logged in user even tough it is an external service.

To integrate online editing into the security model, additional rights were added to the basic Browse, Read, and Write: OnlineEdit and OnlinePublish. These define whether the logged on user shall receive the option to edit the page and also if the user may publish a page after editing.

[18] IIS can be tweaked to run as normal process, so that a debugger can connect to it. However, I never managed it to have IIS fully functional in this mode; the ASP scripting always failed due to reasons I didn't find out. Look in the references for a link to an KB article which describes ISAPI debugging.

If you plan your own implementation, you may have the need for some other token contents to identify the user. What will probably remain the same is the deposit of the token in a custom header line.

Conclusion

While there is no predefined interface available to just add your custom authentication and access control mechanisms, a filter DLL can do the job pretty well. Since the filter implementer has the power to control most aspects of the request handling in IIS, almost anything is possible.

It does, however, require a good understanding of the programming language and the ISAPI interface, since documentation is sparse and available samples are very limited or don't seem to work at all with the current IIS releases (5 and 6). And you have to allow a considerable amount of time for planning and implementation.

Let's go over our ISP wish list and see what the filter can do:

- *The servers shall of course be secure against any kind of attacks (direct or by worms or viruses).*
  The filter adds security by adding an additional layer which is independent of the Windows account used. Also, the filter can help to prevent successful traditional attacks on the server by the use of URL filtering.
- *Web sites should run in a sandbox to prevent any interaction between sites which could be used to exploit or compromise the server or other sites.*
  Through the filter, proper sandboxing can be implemented on IIS, which is not possible otherwise.
- *The server shall deliver good performance even if it is hosting a large number of sites simultaneously.*
  Performance is only affected very little, since the overhead is kept as small as possible. For heavy-traffic sites, secure filter-side caching can be implemented to virtually eliminate the slowdown created by authentication and access check requests against the security service
- *Customers should be able to set up and easily use protected areas in their web sites, without the need of complex and error-prone application-level logic.*
  The filter allows user management per site and without any dependency on Windows accounts. Applications can use the user token to efficiently integrate with the filter security.
- *Customers accessing databases should be prevented from storing any database access information (user name, password, etc.) in their script pages to better secure the database server.*
  By the use of the sandbox user, SQL Server integrated security can be enabled and user names and passwords can be removed from the scripts.
- *Customers should not be able to log onto the machine through services which are not for direct customer use, like maybe telnet.*
  Since the logins are not Windows accounts, the users cannot log onto any service not designed to use the security service. And the credentials for the Windows account used for sandboxing are never disclosed.
- *The servers should be ready for load-balanced clustering of the web service, so that multiple web servers can serve the same content simultaneously.*

The filter enables IIS to use network file shares securely and therefore makes load-balanced web sites accessing shared data storage possible.

Security aspects of IIS can be enhanced with such a filter, since it adds an additional restriction layer but does not really replace existing IIS security features. The possibility to do sandboxing for webs (as shown by this implementation) is just one example how to use this for your benefit.

Appendix A: Source Code Snippets

DISPTREE2.VBS (Microsoft VisualBasic Script)

This script is a modified version of the DISPTREE.VBS script which ships with IIS. It will display a tree of any ADSI container, showing all the nodes and subnodes under the container. However, it does display web-server nodes with some additional details, like the root path and the anonymous user credentials of that server.

The use of the switch --fix will find webs with FrontPage installed and fix the directory security in the MetaBase so that anonymous access to the _vti_aut and _vti_adm directories is never allowed. (see “Alternative approach using UNC file shares for serving contents”)

The original version is by default installed in “C:\INETPUB\AdminScripts”.

```
'---------------------------------------------------------------------------'
' Print the tree of administration objects starting either at the specified node or
' the root
' node of the local machine.
'
' Usage: disptree [--ADSPath|-a ROOT NODE]
'                       [--NoRecurse|-n]
'                       [--fix]
'                       [--help|-?]
'
' ROOT NODE      Optional argument specifies the ADSI path of the first node of the tree
' No Recurse     Specifying this keeps the script from recursing through the tree
'
' Example 1: disptree
' Example 2: disptree -a IIS://LocalHost/w3svc --NoRecurse
'---------------------------------------------------------------------------'

' Force declaration of variables.
Option Explicit

' On Error Resume Next

Dim oFirstNode, Recurse, FrontPageFix, CurrentObj, RootNodePath, StateNames(7)

' By default, we recurse.
Recurse = True

' By Default, no FP fix
FrontPageFix = False

' Set the default path
RootNodePath = "IIS://LocalHost"

' Set State Names
StateNames(1)="Starting"
StateNames(2)="Started"
StateNames(3)="Stopping"
StateNames(4)="Stopped"
StateNames(5)="Pausing"
StateNames(6)="Paused"
StateNames(7)="Continuing"

Dim oArgs, ArgNum
Set oArgs = WScript.Arguments
ArgNum = 0
While ArgNum < oArgs.Count
```

```
  Select Case LCase(oArgs(ArgNum))
    Case "--adspath","-a":
      ArgNum = ArgNum + 1
      RootNodePath = oArgs(ArgNum)
    Case "--norecurse","-n":
      Recurse = false
    Case "--fix":
      FrontPageFix = true
    Case "--help","-?":
      Call DisplayUsage
    Case Else:
      Call DisplayUsage
  End Select

  ArgNum = ArgNum + 1
Wend

Set oFirstNode = GetObject(RootNodePath)

If Err <> 0 Then
  Display "Couldn't get the first node!"
  WScript.Quit (1)
End If

' Begin displaying tree
Call DisplayTree(oFirstNode, 0)

' This is the sub that will do the actual recursion
Sub DisplayTree(FirstObj, Level)
  If (FirstObj.Class = "IIsWebServer") Or (FirstObj.Class = "xxIIsFtpServer") Then
    Dim Root, Found
    Found = False
    Set Root = Nothing
    For Each CurrentObj in FirstObj
      If CurrentObj.Name="ROOT" Or CurrentObj.Name="Root" Then
        Set Root=CurrentObj
'        WScript.Echo "Found Root: " & Root.Class
        Found = Root.Class = "IIsWebVirtualDir"
      End If
    Next
    If Found Then
      Dim Path
      Path=Root.Path
      WScript.Echo Space(Level*2) & Mid(FirstObj.Class, 4, 3) & ": " & FirstObj.Name &
          _" - " & FirstObj.ServerComment & " (" & StateNames(FirstObj.ServerState) & ")"
          _& vbCRLF & Space((Level+1)*2) & "on " & Path
      WScript.Echo Space(Level*2) & "  IUSR:     " & Root.AnonymousUserName & "  Pass: "
          _& Root.AnonymousUserPass
      WScript.Echo Space(Level*2) & "  UNC-User: " & Root.UNCUserName & "  Pass: " &
          _Root.UNCPassword
    End If
  Else
    WScript.Echo Space(Level*2) & FirstObj.Name & " (" & FirstObj.Class & ")"
    If FrontPageFix AND FirstObj.Class="IIsWebVirtualDir" AND FirstObj.Name="_vti_bin"
        _Then
      WScript.Echo Space(Level*2) & "  (FrontPage BIN dir, fixing...)"
      FirstObj.AuthFlags = 1
      FirstObj.SetInfo
      Dim FoundADM, FoundAUT
      FoundADM = False
      FoundAUT = False
      Set Root = Nothing
      For Each Subnode in FirstObj
        If Subnode.Name="_vti_adm" Then
```

```
          SubNode.AuthFlags = 2
          SubNode.SetInfo
          FoundADM = True
        ElseIf Subnode.Name="_vti_aut" Then
          SubNode.AuthFlags = 2
          SubNode.SetInfo
          FoundAUT = True
        End If
      Next
      Dim Subnode
      If Not FoundADM Then
        Set Subnode = FirstObj.Create("IIsWebDirectory", "_vti_adm")
        SubNode.AuthFlags = 2
        SubNode.SetInfo
      End If
      If Not FoundAUT Then
        Set Subnode = FirstObj.Create("IIsWebDirectory", "_vti_aut")
        SubNode.AuthFlags = 2
        SubNode.SetInfo
      End If
    End If
  End If

  ' Only recurse if so specified.
  If (Level = 0) or (Recurse) then
    Dim SortedObj, Keys, KeyMin, KeyMax, KeyNext, I
    Set SortedObj=CreateObject("Scripting.Dictionary")
    For Each CurrentObj in FirstObj
      SortedObj.Add CurrentObj.Class & String(200-Len(CurrentObj.Name), "0") &
          _CurrentObj.Name, CurrentObj
    Next
    Keys=SortedObj.Keys
    KeyMin=""
    KeyNext=KeyMin
    Do
      For I=0 To SortedObj.Count-1
        If (Keys(I)>KeyMin) And ((KeyNext<=KeyMin) Or (KeyNext>Keys(I))) Then
          KeyNext=Keys(I)
        End If
      Next
      If KeyNext=KeyMin Then
        Exit Do
      End If
      KeyMin=KeyNext
      Call DisplayTree(SortedObj.Item(KeyNext), Level + 1)
    Loop
  End If
End Sub

' Display the usage for this script
Sub DisplayUsage
  WScript.Echo "Usage: disptree [--ADSPath|-a ROOT NODE]"
  WScript.Echo "                      [--NoRecurse|-n]"
  WScript.Echo "                      [--fix]"
  WScript.Echo "                      [--Help|-?]"
  WScript.Echo ""
  WScript.Echo " Example 1: disptree"
  WScript.Echo " Example 2: disptree -a IIS://LocalHost/w3svc --NoRecurse"
  WSCript.Quit
End Sub

Sub Display(Msg)
  WScript.Echo Now & ". Error Code: " & Err & " --- " & Msg
End Sub
```

Filter code (Borland Delphi 5.0)

This source code is the core of the filter. You will find the SOAP call over HTTP to the Security Service and see how the filter handles that. Note that the helper libraries used are not freely available and therefore you will not be able to just go ahead and compile the filter without modifications. It is meant as example on the actual implementation.

```
library SecurityFilter;

{$R *.RES}

uses
  SysUtils,
  ISAPIFilterFramework,
  StringUtils,
  EventLog,
  HTTPClient,
  Base64,
  MD5;

type
  TAuthenticationGroup=class(TISAPIFilterGroup)
    procedure Install; override;
  end;
  TInitRequestFilter=class(TISAPIPreProcHeadersFilter)
    procedure Execute(var FilterContext: Pointer); override;
  end;
  TAuthenticateFilter=class(TISAPIAuthenticationFilter)
    procedure Execute(var FilterContext: Pointer); override;
  end;
  TCheckAccessFilter=class(TISAPIAuthCompleteFilter)
    procedure Execute(var FilterContext: Pointer); override;
  private
    function UrlWithoutQuery: string;
  end;
  TCleanupFilter=class(TISAPIEndOfNetSessionFilter)
    procedure Execute(var FilterContext: Pointer); override;
  end;

{ TPreProcHeaderFilter }

procedure TInitRequestFilter.Execute(var FilterContext: Pointer);
var
  Authorization: string absolute FilterContext;
begin
  Authorization:=GetHeader('Authorization:');
  if Authorization<>'' then begin
    DelHeader('Authorization:'); // remove authorization header if any
    if CompareTextLeft(Authorization, 'Basic ', 6) then begin
      Authorization:=Base64ToString(CopyToEnd(Authorization, 6))
    end else begin
      EventLog.Log(leWarning, 'Non-Basic Authentication tried, using anonymous'
          #13#13'URL: %s'#13'Metabase Site: %d'#13'Authorization: %s',
          [GetHeader('URL'), InstanceID, Authorization]);
      Authorization:='';
    end;
  end;
end;

{ TAuthenticationFilter }

procedure TAuthenticateFilter.Execute(var FilterContext: Pointer);
begin
  User:='';
  Password:='';
```

```
  ExitNotificationHandled; // return with anonymous user token (IIS)
end;

{ TAuthenticationCompleteFilter }

procedure TCheckAccessFilter.Execute(var FilterContext: Pointer);
var
  Authorization: string absolute FilterContext;
  Request: IHTTPRequest;
  SoapData, UserAuthority: string;
  I: Integer;

const
  TrueFalse: array[Boolean] of string=('false', 'true');

  procedure LogAccess(LogEntry: TLogEntryKind; Action: string);
  const
    SSL: array[Boolean] of string=('No', 'Yes');
  begin
    EventLog.Log(LogEntry, '%s'#13#13'SSL: %s:'#13'URL: %s'#13
        'Metabase Site: %d'#13'Authorization: %s',
        [Action, SSL[SecurePort], GetHeader('URL'), InstanceID, Authorization]);
  end;

begin
  // find host information
  I:=Pos(':', Authorization);
  // do access check
  Request:=THTTPConnection.ICreate('security-service:9999', 1.1, True).Request(
      'POST', '/SecurityServer.rem', // HTTP action and URI
      ['Content-Type: text/xml; charset="utf-8"', // additional headers
      'SOAPAction: "http://schemas.microsoft.com/clr/nsassem/Security.'+
      'ISecurityManagerForWeb/Security #LoginAndCheckWebAccess"'],
      Format(
      '<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"'+
      ' xmlns:xsd="http://www.w3.org/2001/XMLSchema"'+
      ' xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"'+
      ' xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"'+
      ' xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0"'+
      ' SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">'+
      '<SOAP-ENV:Body>'+
      '<i2:LoginAndCheckWebAccess'+
      ' xmlns:i2="http://schemas.microsoft.com/clr/nsassem/Security.'+
      'ISecurityManagerForWeb/Security ">'+
      '<userName>%s</userName>'+
      '<password>0x%s</password>'+
      '<siteNr>%d</siteNr>'+
      '<siteRootPhysical>%s</siteRootPhysical>'+
      '<uri>%s</uri>'+
      '<isSsl>%s</isSsl>'+
      '</i2:LoginAndCheckWebAccess>'+
      '</SOAP-ENV:Body>'+
      '</SOAP-ENV:Envelope>',
      [Copy(Authorization, 1, I-1), // name extracted from Authorization header
      WStringToMD5String(CopyToEnd(Authorization, I+1)), // hashed password
      InstanceID, // IIS instance number
      GetServerVariable('APPL_PHYSICAL_PATH'), // physical path
      UrlWithoutQuery, // URL with query part removed (if any)
      TrueFalse[SecurePort]])); // SSL connection (true or false)
  if (Request.ResultCode<>200) then begin
    LogAccess(leError, Format('SOAP request failed, error %d: %s'#13#13'Result: %s',
        [Request.ResultCode, Request.ResultText, Request.ResultData]));
    ExitAccessDenied;
  end;
  SoapData:=StringReplace(Request.ResultData, #13#10, '', [rfReplaceAll]); //remove CRLF
```

```
  SetHeader('SecurityUserToken:', SoapData); // store user token as new request header
end;

function TCheckAccessFilter.UrlWithoutQuery: string;
var
  I: Integer;
begin
  Result:=GetHeader('URL');
  I:=Pos('?', Result);
  if I>0 then
    SetLength(Result, I-1);
end;

{ TCleanupFilter }

procedure TCleanupFilter.Execute(var FilterContext: Pointer);
var
  Authorization: string absolute FilterContext;
begin
  Authorization:='';
end;

{ TAuthenticationGroup }

procedure TAuthenticationGroup.Install;
begin
  Install('Security Filter', 1.0,
      [TInitRequestFilter, TAuthenticateFilter, TCheckAccessFilter, TCleanupFilter],
      fpBoth, fpMedium); // bind to both normal and SSL, medium priority
end;

begin
  InstallFilter(TAuthenticationGroup);
end.
```

Appendix B: Links and References

General links

SANS Institute
http://www.sans.org

NT FAQ
http://www.ntfaq.com/

ISAPI

MSDN Online, ISAPI Filters Overview
http://msdn.microsoft.com/library/en-us/iisref/html/psdk/asp/isgu0q0n.asp
This link provides basic information about the ISAPI filters and contains links to filter development tasks as well as the reference.

Microsoft sample authentication filter
http://msdn.microsoft.com/library/en-us/iisref/html/psdk/asp/devs8coi.asp
This sample may help you if you want to write your own filter.

authentProtect
http://bob.firstcodings.com/programs/authentprotect/
Filter to deny access based on user names (even if they had access otherwise)

IIS Security

SANS Track 5 Courseware

General IIS articles, also including security.
http://www.iishelp.com

Untangling Web Security: Getting the Most from IIS Security
http://msdn.microsoft.com/library/en-us/dniis/html/Websec.asp

IIS Security Recommendations When You Use a UNC Share
http://www.microsoft.com/serviceproviders/support/how2_iis_secure_P116528.asp

IIS 6.0: UNC and NAS WebCast
http://support.microsoft.com/default.aspx?scid=/servicedesks/webcasts/wc022603/wcblurb022603.asp

Q214806: How to enable pass-through authentication for UNC paths in IIS
http://support.microsoft.com/default.aspx?scid=kb;en-us;214806

IIS Answers
http://www.iisanswers.com/

RFCs and other standards

RFC Database: All STD, RFCs, errata
http://www.rfc-editor.org/rfc.html

RFC 2616: Hypertext Transfer Protocol (HTTP/1.1)
ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt

RFC 2828: Internet Security Glossary
ftp://ftp.rfc-editor.org/in-notes/rfc2828.txt

World Wide Web Consortium: HTTP, XML, SOAP, and other web standards
http://www.w3.org/

eXtensible Markup Language (XML, used by SOAP)
http://www.w3.org/XML/

Simple Object Access Protocol (SOAP)
http://www.w3.org/TR/SOAP/