Global Information Assurance Certification Paper

Internet Application Security

GIAC Security Essentials

Practical Assignment

Version 1.4b

Option 1

Andy Charles

07/15/2004

Deleted: [An
Deleted: Handbook]
Deleted: [
Deleted:]
Deleted: [
Deleted:]
Deleted: [
Deleted:]
Deleted: 01

Table of Contents

Deleted: Abstract . 2¶
Introduction / Executive Summary . 2¶
SOAP Security . 3¶
IIS Security . 5¶
COM + Security . 7¶
Summary . 8¶
Section Two Title Here . 9¶
Section Three Title . 10¶
Section Four Title Here . 11¶
References . 12¶

Deleted: List of Figures¶
¶
Figure 1: Sample Drawing . 3¶
Figure 2: Second drawing sample . 3¶
¶
¶
¶

Andrew T. Charles

Deleted: <your name>

Deleted: References

Abstract

This paper will discuss the different options available to a security analyst tasked with securing an Internet application. Benefits and pitfalls of each technology will be discussed to support the importance of risk mitigation. Topics highlighted include SOAP, IIS and COM+ security techniques.

Introduction / Executive Summary

The birth of the internet has empowered the consumer; WWW addresses have enhanced the way we live and changed the way we conduct business. Gone are Friday paydays standing in line at that bank - now we find the nearest MAC. Gone are the often misplaced department store catalogues - now we browse the latest fashions from our laptop. We have become so accustomed to online "on my time" services that we take them for granted. As our dependence on web services increases, our patience for system unavailability decreases. Expectations are approaching one hundred percent. While near perfect availability may seem absurd, the technology industry expects this exceptional level of service.

Deleted: ,

Deleted: ,

Deleted: ,

Deleted:

Deleted: for

Deleted: ,

Deleted: w

Deleted:

Deleted:

Some appreciate the application architecture that keep these systems flexible and up-to-date, while most overlook the importance that security plays in the symbiotic web environment. Just think about this when you demand one hundred percent availability; your organization is constantly under attack internally and externally. The intruder has all the cards, desire, a challenge, almost infinite time and with todays peer-to-peer drone networks nearly infinite computing power. Now it becomes a waiting game. It is only a matter of time before the forthcoming of your favorite Murphy's Law. To avoid these uncomfortable situations it is important to understand the technologies available within the internet space, the exposures they introduce and the appropriate ways to mitigate the risks.

Deleted:

Deleted:

Deleted: ,

Deleted: i

SOAP Security

Secure communication from the DMZ (demilitarized or un-trusted zone) to the trusted zone (internal network) is essential. However, until Simple Object Access Protocol (SOAP) came along secure remote procedure calls were, at best, difficult to construct. For example, Microsoft's Distributed Component Model (DCOM) technology is great for development purposes, coming equipped with intelli-sense and the capabilities of a role-based security model. DCOM seemed so promising, incorporating the ease of rapid development with built-in security controls. This was great news for development teams. That is until the firewall analysts were consulted. Unfortunately, DCOM relies on dynamically assigned

Deleted:

Deleted: Everything

Deleted: ,

Deleted:

Deleted: <your name>

Deleted: References

ports for method instantiation. It is impossible to configure DCOM for single port communication. While DCOM is a great technology for an Intranet application, you will be hard pressed to find a border protection analyst willing to open up an external firewall to any and all traffic. As you can imagine, it quickly became apparent that that there needed to be a secure and standardized way to leverage the positives of DCOM (Schneier). It was time to start perfecting SOAP.

Secure SOAP was once considered an oxymoron. That perception has changed. When configured and implemented properly, SOAP can be a viable technology for implementing secure Remote Procedure Call (RPC) solutions. To make the claim that a SOAP message is secure, the integrity and the confidentiality of the message must be guaranteed. The receiver must be able to identify the sender while trusting that the message contents have not been altered. In addition, if the message body contains information that is sensitive, the sender and the receiver must be able to trust that the message has been masked from those without the appropriate credentials.

To avoid confusion it is important to make the separation between Extensible Markup Language (XML) documents and SOAP messages. A strong definition of SOAP can be found at the website Business Computing. SOAP is a platform and protocol independent light weight messaging protocol used to encode web services requests and responses prior to network transmission (SOAP). A SOAP message consists of header and body information (that information is commonly in XML). The header portion of a SOAP message holds metadata, the description of the data. The header is an ideal location for digital signature information and session keys storage. The SOAP body contains the data being passed; a likely requirement would be to encrypt this data (Powell).

Deleted: the

Formatted: Underline

Deleted: C

Deleted: ,

Deleted: [http://sbc.webopedia.com/TERM/S/SOAP.html]

To implement secure SOAP there are several techniques that should be exercised, many of which revolve around understanding and correctly implementing properties of XML documents. Taking advantage of the functionality made available by XML is an excellent way to deploy a reusable application. XML is a standard supported by the World Wide Web consortium. A property especially important to secure SOAP messaging is XML-Signature Syntax and Processing. This is a technique that should be used for tying an individual SOAP envelope to a creator and/or a sender. As described by Powell, the XML-Signature syntax works by creating and verifying digital signatures for all or part of an XML message. It also indicates an XML-compliant mechanism for holding a digital signature within an XML document. This mechanism works by passing an XML document with a digitally signed node, KeyInfo. KeyInfo is used to validate the digital signature using the client's public key, and a unique identifier to eliminate the chance of replay attacks (Powell).

Deleted: a

While the KeyInfo property will guarantee that the sender sent this message it does not do anything as far as encryption is concerned. If the message contains sensitive information, XML encryption should be used. Malicious activity such as

Deleted: <your name>

Deleted: References

copying another client's certificate and using it as their own can be avoided by utilizing the XML signature capability. Whenever encryption is brought into the mix of application development, application performance analysts cringe. Luckily it is possible to only encrypt parts of an XML message. Powell describes three situations where partial encryption is beneficial.

- Sensitive data is hidden from users and applications not authorized to view it.
- Unencrypted portions of information can be shared with services without having to worry about compromising sensitive information.
- Services can obtain needed information without encrypting and decrypting the entire message over and over (Powell).

Even though the functionality for secure SOAP is available through the properties of XML, this type of implementation may not fit the architecture of an organization. The development team's knowledge of XML encryption and digital signing may not be robust enough or maybe the application is being plugging into current "legacy components" that do not have the functionality of parsing through encrypted messages. If problems like this exist it does not mean that an insecure application will have to be implemented (Powell).

Deleted: . M

Deleted: L

Deleted: ¶

Deleted: ,

SOAP is platform and transport layer independent; that flexibility opens up a huge door of opportunity. Encryption at the transport layer is a great way to leverage existing DMZ technology. SOAP calls over an SSL (secure sockets layer) connection will guarantee that the message and the contents of the message are masked during the transmission from the client to the server. Additional security features can be included if message integrity is an issue. The integrity of the message and the sender can be guaranteed by utilizing symmetric or an asymmetric key exchange. To take things a step further authentication could easily be added to the mix for another layer of abstraction. An SSL connection utilizing key exchange and authentication creates a very secure communication with very little need for technical education, these technologies are well with in the comfort zones of most analysts and their effectiveness is tried and proven.

A file essential to the communication capabilities of SOAP is the Web Services Description Language or WSDL file. A WSDL is a standard XML document supported by the World Wide Web Consortium. WSDL files describe network services and the capabilities of the web service as a set of network endpoints (WSDL). They are commonly used to exchange interface information between services. WSDL files also contain information such as URL locations of needed web services, protocols, method definitions and data types required during a procedure call. This information can be extremely sensitive. If the file is compromised and definitions are changed bad things will happen (Christensen).

Deleted: WSLD

Deleted: Containing

Deleted:

Deleted: url

Deleted: l

Deleted: . http://www.w3.org/TR/wsdl#_service

Deleted: <your name>

Deleted: References

Unrestricted WSDL files may be available for download from the Web server. These files contain network endpoint information. It is important to restrict access to WSDL files. The following provided by MSDN are good preventive measures to follow.

Deleted: http://sbc.webopedia.com/TERM/W/WSDL.html¶

- Authorize access to WSDL files using NTFS permissions.
- Remove WSDL files from Web servers.
- Disable the documentation protocols to prevent the dynamic generation of WSDL.
- Capture exceptions and throw a SoapException or SoapHeaderException — that returns only minimal and harmless information — back to the client (Meier).

Deleted: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh12.asp¶

IIS Security

Deleted: ¶

IIS (Microsoft's Internet Information Services) or any web server is a component of web application design where security must not be overlooked. The configuration of a web application includes application design, physical security and the configuration of IIS properties. While physical security and the configuration settings of the entire website may not be within the scope of your application there are several controls that are manageable and will require attention. Virtual directory properties, auditing, error messages, file permissions and client/server authentication are aspects of a web application that should not be overlooked. Web servers are out there and available for attack twenty-four hours a day, seven days a week, and users expect a functional application. To mitigate the risk of application failure strict security policies must be in the works from the planning phase of an application.

Deleted:

A strong security design includes physically separating application components into logical groups. Each application should have a dedicated physical system file folder. The physical application folder or “root application folder” should be separated into several other sub folders. Typical divisions are application script content, display content and compiled content. Depending on the nature of the application this list can become very granular. Segmenting the application is important because it aids the implementation of the least privilege model. The least privilege model, simply put, means that if a resource does not have a justified purpose for accessing another resource it should not be authorized to do so.

When creating the virtual directory for an application it is good practice to disable directory browsing and write permissions. A good mindset from a security perspective is that if something is not needed remove it. For example, by default, a virtual directory is created with several different extension mappings; if you know that your application only uses .asp extensions remove everything but the .asp extension. Logging visits is an auditing technique available within IIS and

Deleted:

Deleted: <your name>

Deleted: References

implementing it should be discussed. An audit log can be beneficial for future incident prevention and post incident reenactment. Unfortunately, there will be maintenance required. If the application is a heavily used application the log files will need to be monitored, possibly even moved or purged to save disk space (Meier).

When an application virtual directory is created a mapping to a physical system file folder occurs. The physical file folder contains all the content that will be made available to web users. If the content is changed in the physical file folder those changes are published once the virtual directory or website is refreshed. It is vital to ensure that only authorized users have the capability to make production file changes. Only a defined number of authorized users should have access to the physical folder. DMZ environment standards should be consulted for core access requirements. When changing default system access to physical file folders it is important to review existing environment documentation. It would be undesirable to either lock service accounts needed for web server maintenance or grant access that is not necessary.

Deleted:

Deleted:

Important and often overlooked implementation techniques are custom error messages. Error messages that will be displayed to the end user should never reveal information about the application. Only provide the bare minimum information to the user, detailed error information can easily reveal intrinsic core application knowledge that will help reverse engineer an application. If a proper support channel exists, refer users to the channel. Error messages should be reviewed by both the application support area and security analysts prior to implementation.

Deleted:

Deleted: ¶

Application of the least privilege model is essential to DMZ security because of the inherent threats. Because local accounts and groups can be tailored specifically to an application, having well named groups and accounts makes support for the application less complicated. A good practice for account management and implementing the least privilege model is creating local group, user, service and application accounts then nesting the appropriated user, service or application account within the application specific local group. Once the local group is created it is then applied to the physical application folders or files. This will help prevent unauthorized access while logically grouping accounts. For auditing purposes, naming conventions for the different types of accounts should be discussed. The naming convention can be a simple prefix. Local user accounts could begin with the prefix USR_ while service accounts could be prefixed with SVC_ and so on. The options are endless; having a distinct pattern for naming accounts can be a huge time saver when reviewing audit logs.

Deleted: <your name>
Deleted: References

Authentication

Depending on the content of the application, eliminating the default setting of anonymous access may be necessary. If anonymous access is removed, the application will not function unless another authentication method is selected.

Secure communication between the client and server is vital. IIS offers several ways to secure communications; basic authentication, digest authentication, Windows Integrated Authentication (NTLM), and Kerberos. Before an authentication method is selected it is important to understand what each offers.

Basic authentication is very insecure. The user name and password are sent Base64-encoded over the wire without encryption. Because this information is sent in the clear, sniffing the traffic will easily reveal the user name and password the application is using to authenticate. This is a serious threat because the account information being passed could be an elevated access level account. Either way, the threat of more valuable resources being compromised exists. One way to mitigate this is to use encryption. Basic authentication coupled with an SSL connection is a very secure approach. The SSL handshake happens before the authentication handshake. Therefore the user name and password are passed over a secure connection (Gavrylyuk).

Deleted: a
Deleted: Au
Deleted: This combination is some type of account
Deleted: whether local or at the domain
Deleted: Therefore,
Deleted: T
Deleted: Au
Deleted: ,
Formatted: Font: Arial
Deleted: Au
Deleted: a
Deleted: fairly
Deleted: ,
Deleted: D
Deleted: ,
Deleted: http://www.whitemesa.com/soapauth.html#S4

Digest authentication is easy to use and does not reveal user name and password. The digest mechanism allows a client to authenticate itself by presenting credentials consisting of an MD5 digest transmitted in a request message. It is based on the principle that the client and server are in possession of a shared secret, a password string. The advantage of this method is that the client password is only used in calculating the digest, so it remains safe from network exposure. If not coupled with SSL the digest is susceptible to replay attacks. Since digest authentication requires an SSL tunnel it is better to use basic authentication and take advantage of the performance gain (Cunnings).

Deleted: a B
Deleted: a
Deleted: A
Deleted: S
Deleted: s
Deleted: T

Windows Integrated Authentication (NTLM) is the native Windows authentication scheme. Like digest authentication, hashes are exchanged instead of user name and password. NTLM is great for intranet applications where the platform is a known factor. If a username and password combination is not specified, the current user logon credentials are used. The major problem is that if the client and server are not Windows based machines, NTLM will not work. However, NTLM is very secure and it is not susceptible to replay attacks and it is very fast in a Windows environment (Gavrylyuk).

Deleted: .

Kerberos is another network authentication protocol. It is designed to provide strong authentication for client/server applications by utilizing secret-key cryptography. The first Microsoft operating system to support Kerberos is the

Deleted: <your name>

Deleted: References

Windows 2000 platform. Kerberos requires a KDC (key distribution center) server to request a service ticket from. It is not recommended to expose your KDC server on the Internet, so Kerberos is normally restricted to intranet applications.

COM+ Security

Deleted: ¶
¶
¶
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoap/html/soapsecurity.asp¶

Deleted:

To enforce security at the mid-tier level of a multi-tier application or on the DMZ server directly, a combination of strategies defined in the COM+ security model, along with file and registry permissions, should be implemented. The COM+ security model encompasses four primary goals; activation control, access control, authentication control and identity control. Activation control is used for determining who is permitted to launch COM+ components. Access control determines who can touch the components objects or methods. Authentication control is used to verify that the network transmission is valid and that only authorized viewers can see the data. Identity control refers to the set of credentials which the COM+ component will execute under (Eddon). Each of these strategies are independently important exposure reduction controls.

Deleted: permissions

Deleted: .
http://www.microsoft.com/msj/1199/comsecurity/comsecurity.aspx

Deleted: is

Most Microsoft-centric web applications include some type of compiled code, usually in the form of an application controller. An application controller can serve many purposes. The most general is a home for thin business logic. Thin business logic is classified as non-strategic application logic. A good example that helps delineate between thin business logic and business logic is the customizable portal or “one stop shop” applications we are all familiar with. The thin business logic of a portal application consists of the logic that helps the user change the colors and layout, while the business logic consists of the logic that determines what account information to retrieve and display. An important difference between thin business logic and business logic is where each lies within a web application. The natural home for thin business logic is the web-tier while business logic resides in the internal network. Thin logic and business logic are dependant on each other. Without the thin logic, the business logic would not mean much to the user, something like an XML blob of information. The reverse also holds true; thin logic without the core displayable information is basically useless. Because thin business logic resides in the DMZ surrounded by firewall rules, a defined way to call the mid-tier to get the information the user requests is needed. This is where SOAP comes in handy, as discussed previously. SOAP has the ability to make secure remote procedure calls (method instantiation) from the DMZ to a destination on the trusted side.

Deleted: ,

Deleted: xml

Deleted:

If the thin business objects require information from the trusted side, confidentiality items need discussion. How will the application handle the destination server address and user name and password for authentication? If the device requires an authenticated call then the call will have to include a set of credentials that the destination server understands. Both the server address and the credentials are pieces of information that can not be revealed, these are keys

Deleted: <your name>

Deleted: References

to internal network security. To mitigate the risk of credential theft, consider implementing the strategy of compiling thin business logic. Retrieving source code from a compiled object is much more difficult than simply viewing the source of an .asp page. To take things a step further, consider creating a credential storage strategy.

Essential components of a credential storage strategy include the physical storage of the credentials and the reusability of the solution. To avoid losing control of key-pairs, the reusable component should contain a common key used for encrypting and decrypting information. Once a common component for encryption and decryption has been created, it will be beneficial to determine a central storage location for the encrypted data. The location could be a central database or if a database is unavailable, like in the DMZ, use a local database or registry. Do not forget to secure the storage repositories, forgetting to do so negates the effort. Having all credentials stored in a central location makes restricting and monitoring access more manageable. Now that your enterprise has a reusable component to retrieve credential information, the thin business logic can make remote procedure calls without having to know a server name, a user name or a password.

Deleted: ¶

To recap, we have a secure call coming into our DMZ via an SSL connection and then to an .asp page. That .asp page understands that in order to display information, it needs to consult the application controller. The .asp page calls a compiled application controller; the application controller is invoked to retrieve the display information. To get the information from the trusted zone a SOAP call must be made. The SOAP call requires certain parameters; the destination server, the service to invoke, a user name and a password to authenticate to the destination. The application controller calls the referenced reusable component to retrieve the required information from the registry, the sensitive information is retrieved, the call is made and data is retrieved. The thin business object has the data it needs to finish the display logic. Now the customized page can be constructed for the customer.

Deleted: ,

Sounds like a secure approach, right? Well it is a good start. One vital component has been left out. COM+ security has not been implemented yet. The application never determined whether the calling .asp is actually authorized to invoke the application controller. With the absence of role-based security and activation control, any application can call any service from any application as long as the component name and required values are presented. This inherent insecurity exposes the need for another level of security. COM+ security is essential for marshalling these scenarios. A COM+ package must be constructed to house the application controller and to associate an identity to the application.

Assigning an identity to a package defines the system account used when the application is invoked. While it is possible to use the interactive user setting,

Deleted: <your name>

Deleted: References

which can be beneficial during code development and debugging sessions, not assigning a defined account to a COM+ package has its pitfalls. By nature a COM+ application is built by leveraging other reusable services; if those services reside in other COM+ packages the component being launched may not have access to launch the reusable services. If a component is not authorized to launch a referenced component the entire call stack fails. The resolution is not to allow all authenticated accounts across the board access to every COM+ package. That would not be secure at all, especially in a shared environment. Instead, map out the entire application flow. Include all new components and all existing components being referenced; add registry stores, databases and physical file locations. With your map in hand, pencil in the call chain. The chain must include all incoming and out-going identities. Verify that each authorization check has the appropriate list of users authorized to call the needed resource. If the components of your application are all contained in the same COM+ package, the mapping is fairly straightforward. When an application spans multiple packages remember to add the identity of your package to a role of the package being called. This is not necessary if the package is set as a library package. A library package assumes the caller's identity. Remember that once security is enabled and a package identity is set, the identity of the component being invoked changes to the identity of the COM+ package in which the component resides (COM+ Security Concepts).

Deleted:

Deleted: is

Deleted: to

Deleted:

Deleted: If the security set is more granular than package level, apply your identity to a role at the appropriate level.

Deleted: may not be

Deleted: (unless specified otherwise).

If security is enabled at the package level, role-based security will also be necessary. Role-based security works by mapping out a logical set of application users then creating corresponding system or domain accounts. Once the mapping is complete, roles are created for the COM+ package. These roles are then populated with accounts then applied to the package, component, interface or method of the application. Performance considerations should be discussed before implementing the most granular level of role-based security. Depending on the authentication control setting of the package, each and every call may be authenticated, verified and encrypted. As you know, performance numbers get worse with every step added to a call process. Also, keep in mind when enabling package level security that calling components will require physical access to the files in the call chain. Windows file permissions can also conflict with role-based security (Eddon).

Deleted: ,

Deleted: t

Deleted: :

Deleted: U

Deleted: P

Deleted: N

Deleted: D

Deleted: M

Deleted: P

Deleted: D

Deleted: A

Deleted: E

Deleted: V

Deleted: Trojan

Deleted: :

Deleted: Au

Deleted: Au

Deleted: Au

Deleted: P

Deleted: I

Summary

The threats are overwhelming; unauthorized access, parameter manipulation, network eavesdropping, disclosure of configuration data, message replay, profiling, denial of service, arbitrary code execution, elevation of privileges, viruses, worms, Trojan horses and so on and so forth. With all of these threats the challenge of protecting our assets and keeping them available becomes apparent. It is important for each employee to apply diligence and the essential security philosophies; authentication, authorization, auditing, privacy, integrity,

Deleted: <your name>

Deleted: References

availability, and non-repudiation. It is as important to understand the risks involved as it is the technology used. Creating secure solutions encompasses protocol decisions, hardware requirements, software selection, code assurance, quality control, and access requirements. New applications often bring unknowns and uncertainties, none of which may be simply "skirted". Each risk must be identified and assessed. Appropriately identifying and mitigating risks up front will help avoid embarrassing situations. The importance of applying strong security architecture can not be stressed enough.

Deleted: A

Deleted: N

Deleted: ,

Deleted: e

An analogy for defense in depth helps emphasize the necessity for well defined security architecture. Imagine a king's castle. He builds his humble dwelling with heavy bricks and stacks them high to make walls. He digs as deep as he can all the way around his house, then he floods the ditch to create a moat. Finally, for kicks, he assembles everyone he knows, not for a party, but to fill the interior and surround the perimeter. Now this may seem a little excessive, but an important question should be asked. Why would a king be willing to go through all this trouble? For the very same reason an internet application should be secured, things we treasure should be kept safe.

Deleted: K

Deleted: ,

Deleted: h

Deleted: K

Andrew T. Charles

Deleted: <your name>

Deleted: References

References

Christensen, Erik. "Web Services Description Language (WSDL) 1.1." 15 March 2001. W3C. http://www.w3.org/TR/wsdl# service.

"COM+ Security Concepts." MSDN. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cossdk/htm/pgservices_security_4fw3.asp.

Cunnings, Robert. Salz, Rich. "SOAP Extensions: Basic and Digest Authentication." October 2001. http://www.whitemesa.com/soapauth.html#S4.

Eddon, Guy. "The COM+ Security Model Gets You out of the Security Programming Business." November 1999. MSDN. http://www.microsoft.com/msj/1199/comsecurity/comsecurity.aspx.

Gavrylyuk, Kirill. "Building Secure Web Services with Microsoft SOAP Toolkit 2.0". July 2001. MSDN. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoap/html/soapsecurity.asp.

Meier, J. D. "Improving Web Application Security: Threats and Countermeasures." June 2003. MSDN. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh12.asp.

Powell, Matt. "Real SOAP Security." 21 November 2001. MSDN. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnservice/html/service11212001.asp.

Schneier, Bruce. "SOAP." 15 June 2000. Crypto-Gram Newsletter. http://www.schneier.com/crypto-gram-0006.html.

"SOAP." 13 March 2003. Small Business Computing.com. http://sbc.webopedia.com/TERM/S/SOAP.html.

"WSDL." 25 June 2002. Small Business Computing.com. http://sbc.webopedia.com/TERM/W/WSDL.html.

Formatted: Underline

Formatted: Indent: First line: 0.5"

Deleted: .

Formatted: Indent: Left: 0.58", First line: 0"

Deleted: ¶

Deleted: http://www.whitemesa.com/soapauth.html#S4.

Deleted: ¶

Formatted: Font: Arial

Formatted: Font: Arial

Deleted: 13 July 2004.

Formatted: English (U.S.)

Formatted: Indent: First line: 0"

Formatted: Indent: First line: 0"

Formatted: Indent: Left: 0"

Formatted: Indent: First line: 0"

Deleted: "Web Services Description Language (WSDL) 1.1." 15 March 2001. W3C. 13 July 2004.¶ http://www.w3.org/TR/2001/NOTE-wsdl-20010315.¶ ¶ Gavrylyuk, Kirill. "Building Secure Web Services with Microsoft SOAP Toolkit ¶ 2.0". July 2001. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoap/html/soapsecurity.asp .¶

Formatted: Indent: Left: 0.5"