Security consists of more than just some firewalls at the edge of your network protecting you from the outside. It is a difficult and complex set of actions and procedures that strive to strengthen your systems as much as is appropriate. This article discusses many aspects of security in general, including the IBM® WebSphere® Application Server security architecture, and discusses hardening a WebSphere Application Server environment. This revision of an earlier article has been significantly updated for WebSphere Application Server V6.1 and V7, and has been edited to focus solely on hardening. Part 1 of 2.

View more content in this series

Introduction

IBM WebSphere Application Server’s security continues to evolve with each release. In addition to adding new function in each version, we also strive to enhance the default security of the product. By improving the default settings, we continue to improve how we meet the critical security principle of secure by default. The previous version of this article focused on WebSphere Application Server V6 and the hardening steps required for that version. In subsequent releases of WebSphere Application Server, the number of hardening steps was significantly reduced and, more importantly, most of the steps that remain became less critical. Therefore, it was time to update this article with current information.

This updated article begins with a brief discussion on why security is important and the challenges around hardening systems, and then discusses how to harden a WebSphere Application Server environment to address a variety of security vulnerabilities. So that this article can focus primarily on hardening, some information will be presented at a high level without delving into details. Wherever possible, references to appropriate resources are provided so that you can further explore related subtopics.

While the information in this article is based on IBM WebSphere Application Server V7, most of the issues discussed here apply equally V6.1. Where an issue is unique to a specific version, it will be
Why security?

Build security into applications by design

Secure workloads against the latest threats, simplify access control, and comply with regulatory requirements efficiently with Security services in IBM Bluemix.

Sign up for a free trial

Hopefully, you already realize that security is a key aspect of enterprise systems. Nonetheless, let's briefly go through the exercise of justifying security anyway, just to introduce some common ways of thinking about it.

The fundamental purpose of security is to "keep the bad people out of your systems." More precisely, security is the process of applying various techniques to prevent unauthorized parties, known as intruders, from gaining unauthorized access.

There are many types of intruders out there: foreign spy agencies, corporations in competition with you, hackers, perhaps even your own employees. Each of these intruders has different motivations, different skills and knowledge, different access, and different levels of need. For example:

- An employee might have a grudge against the company. Employees have tremendous levels of internal access and system knowledge, but for the most part probably have limited resources and hacking skill.
- An external hacker is probably an expert in security attacks, but might not have any particular grudge against you.
- A foreign spy agency might have a great deal of interest in you, depending on your business, and could possess tremendous resources.

Intruders might be after your systems for one of two reasons: to gain access to information that they should not have, or to alter the behavior of a system in some way. In the latter case, they might seek to perform transactions that benefit them by changing the system behavior, or they might wish to simply cause your system to fail in some interesting way in an effort to harm your organization.

The point is that there are many different types of intruders, many different motivations for intruding, and, as you will see later, many different types of attacks. You must be aware of this as you plan your security.

Focus on internal as well as external threats

Security should not be seen as simply a gate that keeps the "outsiders" out. That is far too simplistic a view. Many organizations today focus their security efforts entirely on people outside of

identified as such. If you are using an earlier version WebSphere Application Server, refer to the earlier article, as there are significant differences.
the organization in the mistaken belief that only outsiders are a danger. This is simply not the case. For a large corporation, there are literally thousands of people -- many of whom are not employees -- that have access to the internal network. These people are all possible intruders, and since they are on the inside, they have better access to the network. It is often a simple matter of plugging a laptop into a network connection to gain access to a corporate network. Several studies have indicated that perhaps as many as half of all break-ins are caused by or involve employees or contractors within an organization.

Even assuming you believe that every person on your network is trustworthy, can you also assume they never make a mistake? Given the rise of e-mail-based viruses that readily flow over e-mail accounts, JavaScript™-based attack programs, and programs brought in via unknown USB keys and CDs that get plugged into your computers and then attack from within the company network, it is reckless to assume your internal network can be trusted -- it cannot.

It is crucial that your security efforts protect your systems from all potential intruders, which accounts for why this article is as lengthy as it is. Security consists of more than just firewalls at the edge of your network protecting you from the "outside." It is a difficult and complex set of actions and procedures that strive to strengthen your systems as much as is appropriate.

Limitations and reality

It is important that you realize there is no such thing as a perfectly secure system. Your goal is to protect the system as well as you can within the constraints of the business. When thinking about security, you ideally should:

1. Analyze the various points of attack.
2. Consider the risk of an attack at each point.
3. Determine the potential for damage from a successful attack that results in a security breach.
4. Estimate the cost of preventing each attack.

When estimating the damage of a security breach, never forget that security breaches can cause users to lose faith in the system. Thus, the "cost of security breach" might include some very high indirect costs (for example, loss of investor confidence).

Because some hackers break into systems simply for the fun of it, what you can hope is that by creating a reasonably secure environment, intruders will move on to easier targets.

Once you have performed the above steps, you can then determine appropriate tradeoffs of risk versus cost. Essentially, the goal is to make the intruder’s cost of breaking into your system exceed the value of what is gained, while at the same time ensuring that the business can bear the costs of running a secure system (see sidebar).

Ultimately, the level of security that is required is a business decision, not a technical one. However, as technicians, we must help all parties understand the value and importance of security. That said, with the exception of protection against internal application attacks, most of the security hardening steps suggested in this article are fairly low cost. Most organizations should be reasonably able to implement most of them. What this article does not discuss is more...
sophisticated (and expensive) security approaches — stronger authentication, auditing, intrusion detection — that go far beyond the native WebSphere Application Server product capabilities.

Security is a large topic, and it is impossible to completely cover all aspects of security in one article. This information is not intended to be an introduction to security or a tutorial on how to secure systems. Rather, it is a high-level overview, or checklist, of the core technical issues that need to be considered as they relate to WebSphere Application Server security. The information in this article should be used in conjunction with a much larger effort that is designed to create a secure enterprise.

Be sure to review the Resources provided at the end if you are interested in learning more. In particular, my Web site provides a high-level, if somewhat dated, overview of the basics of application security.

Social engineering

Since this is a technical article, the focus is on technical solutions to securing systems, and specifically on the WebSphere Application Server piece of the security puzzle. Nonetheless, be aware that it is often easier to compromise systems using social engineering techniques. That is, attackers are often able to gain access to systems and information to which they should not have access by tricking the human beings that work for an organization. Perhaps the one relevant conclusion that you can learn from social engineering attack techniques is the fact that by using social engineering, your attackers might be coming from within your network. This again serves to emphasize the earlier point that focusing security solely on keeping the intruders out of the network is insufficient. The discussion here, then, will focus on security at multiple levels. Each level tries to thwart different types of attacks and also provides more barriers to attackers.

Total system view: The details matter

Before delving into specific point-by-point recommendations, let’s take a moment to outline the fundamental techniques for creating secure systems. The fundamental view is to look at every system boundary or point of sharing and examine what actors have access to those boundaries or shared components. That is, given that this boundary exists (presuming reasonable trust within a subsystem), what can an intruder do to break this boundary? Or, given that something is shared, can intruders share something inappropriately?

Most boundaries are obvious: network connections, process-to-process communication, file systems, operating system interfaces, and so on, but some boundaries are more subtle. For example, if one application uses J2C resources within WebSphere Application Server, you must consider the possibility that some other application might try to access those same resources. This occurs because there is a system boundary between the first application and WebSphere Application Server, and between a second application and WebSphere Application Server. Perhaps both applications can access this common resource (in fact, they can). This is a case of possible inappropriate sharing.

In a WebSphere Application Server environment, the operating system protections for APIs are of limited value because they are based on process identity, which is a very coarse-
The way you prevent these various forms of attack is to apply a number of well-known techniques. For lower-level network-based attacks, apply encryption and network filtering. You essentially deny the intruder the ability to see or access things they should not see. You also rely on operating systems to provide mechanisms to protect operating system resources from abuse. For example, you wouldn't want ordinary user-level code to be able to gain access to the system bus and directly read internal communication. You also leverage the fact that most modern operating systems possess fairly robust protections for system APIs (see sidebar). At a high level, you apply authentication and authorization rigorously. Every API, every method, and every resource potentially needs to require some form of authorization. That is, access to these things must be restricted based upon need. And, of course, authorization is of little value without robust authentication. Authentication is concerned with reliably determining the identity of the caller. Robust is specified here because authentication that can be easily forged is of little value.

Where appropriate authentication and authorization are not available, then you must resort, frankly, to clever design and procedures to prevent potential problems. This is how you protect J2C resources. Because WebSphere Application Server does not provide for authorization of access to J2C resources, you instead apply other techniques to limit (based on configuration) the ability of applications to inappropriately reference J2C resources.

As you might imagine, examining all of the system boundaries and shared components is a difficult task. And, in fact, securing a system can lead to thinking deeply about complexity. Perhaps the hardest truth about security is that creating a secure system works against abstraction. That is, one of the core principles of good abstraction is the hiding of concerns from higher-level components. That is a highly desirable and good thing. Unfortunately, intruders aren't kind. They don't care about your abstractions or your good designs. Their goal is to break your systems any way they can, and look for holes in your design as they do so. Therefore, in order to validate a system's security, you have to think about it at every level of abstraction: at the highest architectural level, but also as the lowest level of detail. While there are a number of application scanning tools that can assist in code reviews (such as IBM Rational® AppScan®), it's still essential that you manually review every line of code and design decision to prevent application attacks, even if you employ a scanning tool. Rigorous reviews of everything are required.

The smallest mistake can undermine the integrity of an entire system. This is best exemplified by the technique of taking control of C/C++ based systems by using buffer overrun techniques. Essentially, an intruder passes in a string that is too large for some existing buffer. The extra information then overlays a part of the running program and causes the runtime to execute instructions that it should not execute. With care, an intruder can cause a program to do almost anything. As a security architect, to even identify this attack, you have to understand deeply how the C/C++ runtime manages memory and executes running programs. You also have to review every line of code to find this particular hole, assuming you understood that it existed. Today, we know about the attack, yet it continues to be successful because individual programmers make very small bad decisions that compromise entire systems. Thankfully, this particular attack seems
to be infeasible in Java™, but do not believe for an instant that there aren't other small errors out there that lead to compromise.

Think hard about security; it is hard.

**Hardening security overview**

The Java EE specification and WebSphere Application Server provide a powerful infrastructure for implementing secure systems. Unfortunately, many people are not aware of all of the issues surrounding creating a secure WebSphere Application Server-based system. Because there are many degrees of freedom and many different sources of this information, some users tend to overlook security issues and deploy systems that are not particularly secure. To address this, this section attempts to summarize the key issues of greatest importance.

Security hardening is the act of configuring WebSphere Application Server, developing applications, and configuring various other related components in a way to maximize security — in essence, to prevent, block, or mitigate various forms of attack. In order to do this effectively, it is important to consider the forms of attack. There are four basic approaches to attacking an application server:

- **Network-based attacks**: These attacks rely on low-level access to network packets and attempt to harm the system by altering this traffic or discovering information from these packets.
- **Machine-based attacks**: In this case, the intruder has access to a machine on which WebSphere Application Server is running. Here, Your goal is to limit the ability to damage the configuration or to see things that shouldn't be seen.
- **Application-based external attacks**: In this scenario, an intruder uses application-level protocols (HTTP, IIOP, JMX, Web services, and so on) to access the application, perhaps via a Web browser or some other client type, and uses this access in an attempt to circumvent the normal use of the application usage and do inappropriate things. The key is that the attack is executed using well-defined APIs and protocols. The intruder is not necessarily outside the company, but rather is executing code from outside of the application itself. These types of attacks are the most dangerous, as they usually require the least skill and can be done from a great distance as long as IP connectivity is available.
- **Application-based internal attacks** (also known as application isolation): In this case, you are concerned with the danger of a rogue application. In this scenario, multiple applications share the same WebSphere Application Server infrastructure and you do not completely trust each application.

To help you tie these techniques back to these classes of attack, each technique will use this key to represent these vulnerabilities:

- **N**: Network-based attack.
- **M**: Machine-based attack.
• **E**: Application-based external attack.
• **I**: Application-based internal attack.

The appropriate key(s) will be shaded to indicate the type of attack(s) each technique helps to prevent. Keep in mind that internal applications can always take advantage of external methods of attack as well. Therefore, **I** (internal) will not be explicitly indicated when **E** (external) is already present.

Be aware that one other form of technical attack is not considered here: denial of service (DoS) attacks. While very important, DoS attacks are beyond the scope of this article. Preventing DoS attacks requires very different techniques that go well beyond what an application server can provide. You need to consider network traffic monitors, rate limiters, intrusion detection tools, and more, in order to defend against a DoS attack.

**Hardening approach**

Let’s identify the various known steps you should take to protect the WebSphere Application Server infrastructure and applications from these four forms of attack. (We say “known” steps because, of course, it is possible that other weaknesses exist that have not yet been identified.) It would be ideal to be able to organize the information into four buckets, one for each form of attack. Unfortunately, attacks don't neatly divide along those lines. Several different protection techniques help with multiple forms of attack, and sometimes a single attack can leverage multiple forms of intrusion to achieve the end goal. For example, in the simplest case, network sniffing can be used to obtain passwords, and those passwords can then be used to mount an application-level attack. Instead, hardening techniques are generally organized into a logical structure based on when the activity occurs, or based on the role of the person concerned with these issues:

• **Infrastructure**: Actions that can be taken to configure the WebSphere Application Server infrastructure for maximum security. These are typically done once when the infrastructure is built out and involve only the system administrators.
• **Application configuration**: Actions that can be taken by application developers and administrators and are visible during the deployment process. Essentially, these are application design and implementation decisions that are visible to the WebSphere Application Server administrator and are verifiable (possibly with some difficulty) as part of the deployment process. This section will have many techniques, further reinforcing the point that security is not bolted-on; security is the responsibility of every person involved in the application design, development, and deployment.
• **Application design and implementation**: Actions that are taken by developers and designers during development that are crucial to security but might have little impact on the deployment process.
• **Application isolation**: This is explicitly discussed separately because of complexity of the issues involved.

The various techniques presented here are ordered within each section by priority. Prioritization is, of course, subjective, but the threats within each area have been prioritized roughly using this thinking:
• Machine-based threats are less likely than network threats because access to the machine in production is usually restricted. If this isn’t the case in your environment, then these threats become very likely and you should first act to restrict access to those machines.
• Attacks that can be performed remotely using only IP connectivity are the most serious. This implies that all communication must be authenticated.
• Traffic should be encrypted to protect it, but encryption of WebSphere Application Server internal traffic is less important than encryption of traffic that travels "beyond" WebSphere Application Server because the network traversed might have more points where an attacker can snoop traffic.

SSL/TLS overview

SSL/TLS (hereafter referred together as SSL) is a key component of the WebSphere Application Server security architecture, used extensively for securing communication. SSL is used to protect HTTP traffic, IIOP traffic, LDAP traffic, and SOAP traffic. SSL requires the use of public/private key pairs, and, in the case of WebSphere Application Server, these keys are stored in key stores. Because of the key importance of SSL in securing the infrastructure, let's digress momentarily to cover some key aspects of SSL as they relate to WebSphere Application Server. (This discussion is intentionally superficial and discusses only the key points you need in order to properly configure SSL.)

Public Key Cryptography is fundamentally based upon a public/private key pair. These two keys are related cryptographically. The important point is that the keys are asymmetric; information encrypted with one key can be decrypted using the other key. The private key is, well, private. That is, you must always protect that private key. Should anyone else ever gain access to the private key, they can then use it as "proof" of identity and act as you; it is like a password, only more secure and more difficult to change. Possessing the private key is proof of identity. The public key is the part of the key pair that can be shared with others.

If there was a secure way to distribute public keys to trusted parties, that would be enough. However, Public Key Cryptography takes things a step further and introduces the idea of signed public keys. A signed public key has a digital signature (quite analogous to a human signature) that states that the signer vouches for the public key. The signer is assuring that the party that possesses the private key corresponding to the signed public key is the party identified by the key. These signed public keys are called certificates. Well-known signers are called Certificate Authorities (CA). It is also possible to sign a public key using itself. These are known as self-signed certificates. These self-signed certificates are no less secure than certificates signed by a certificate authority. They are just harder to manage, as you'll see in a moment.

Figure 1 shows the basic process of creating a certificate using a CA and distributing it, in this case, to perform server authentication with SSL. That is, the server possesses a certificate that it uses to identify itself to the client. The client does not have a certificate and is therefore anonymous to SSL.
When looking at this figure, notice that the client must possess the certificate that signed the generated public key of the server. This is the crucial part of trust. Since the client trusts any signing certificate it has (which in this case includes the CA certificate), it trusts certificates that the CA has signed. It's worth noting that if you were to use self-signed certificates, you would need to distribute manually the self-signed certificate to each client rather than relying on a well-known CA certificate that is likely already built into the client. This is no less secure, but if you have many clients, it is much harder to manage distributing all of those signing certificates (one for each server) to all clients. It's much easier to distribute just one CA certificate that signs many certificates.

That's pretty much it for server authentication using SSL. After the initial handshake, SSL will actually switch to secret key encryption using a key created during the handshake to secure the channel, but the details of that aren't germane to this discussion.

When a client authenticates itself to a server, the process is similar, although the roles are reversed. In order for a server to authenticate a client (this is often called client certificate authentication, or when combined with server authentication this is known as two way SSL), the client must possess a private key and corresponding certificate, while the server must possess the corresponding signing certificate or a copy of the client's certificate. That's really all there is to it. Notice what wasn't required: SSL certificate authentication merely determines that the certificate is valid, not who the certificate represents. That is the responsibility of later, post-SSL processing. This has significant implications as you'll shortly.
In summary, since SSL uses certificate authentication, each side of the SSL connection must possess the appropriate keys in a key store file. Whenever you configure SSL key stores, think about the fundamental rules about which party needs which keys. Usually, that will tell you what you need.

**Limiting access using only SSL**

As mentioned above, once SSL validates the certificate, the authentication process is over from SSL's perspective. Ideally, what should happen next is that another component will look at the identity in the certificate and then use that identity to make an authorization decision. That authorization decision could be the client deciding the server is trusted (Web browsers do this when verifying that the name in the certificate is the same as the Web server’s hostname) or the server extracting the user’s name and then using that to create credentials for future authorization decisions (this is what WebSphere Application Server does when users authenticate).

Unfortunately, not all systems have that capability. This is where you can take advantage of a popular SSL trick: limiting the valid certificates.

In the previous scenario involving client authentication, the client presents a certificate which is validated by the server against the set of trusted certificate signers. Once it is validated, the SSL handshake completes. If you limit the signers you trust on the server, you can limit who can even complete that SSL handshake. In the extreme with self-signed certificates, you can create a situation where there is only one signer: the self-signed certificate. This means that there is only one valid client side private key that can be used to connect to this SSL endpoint: the private key you generated when you created the self-signed certificate in the first place. This is how you can easily limit who can connect to a system over SSL -- even if the server side components don’t provide authorization. Think of this as creating a secure trusted tunnel at the network layer. Assuming you’ve configured everything correctly, only special trusted clients can even be connecting over this transport. That’s very useful in several situations in WebSphere Application Server which will be discussed later.

**Managing SSL**

As you have already seen, WebSphere Application Server manages keys in key store files. There are two types of key files: key stores and trust stores. A trust store is nothing more than a key store that, by convention, contains only trusted signers. Thus, you should place CA certificates and other signing certificates in a trust store and private information (personal certificates with private keys) in the key store.

Unfortunately, there is a catch to this simple system. Most of WebSphere Application Server uses the PKCS12 format. (WebSphere Application Server SSL configurations in fact support three modern key database formats: JKS, JCEKS, and PKCS12.) The IBM HTTP Server and the WebSphere Application Server Web server plug-in use an older key format known as the KDB format (or more correctly the CMS format). The two formats are similar in function but are incompatible in format. Therefore, you must be careful not to mix them up.
WebSphere Application Server SSL configurations

As of WebSphere Application Server V6.1, a robust infrastructure is provided for managing certificates and SSL within the product. The remainder of this article assumes you are familiar with that information.

Hardening WebSphere Application Server

WebSphere Application Server V6.1 and later versions were designed with the security principle of secure by default. While not perfectly achieved, the goal was to release a product whereby, in the most common configurations and simpler environments, the product is configured reasonably securely by default. Obviously, more complex environments will possess unique issues that simply cannot be anticipated, but for simpler environments the objective was a default installation and configuration that resulted in a reasonably secure system; not perfectly secure because such a thing is not possible. Nor did we eliminate every vulnerability because many are minor and closing them by default would greatly complicate application development, management, or compatibility. But where a vulnerability could reasonably be eliminated in a way in which most clients we felt would concur, we did so.

Infrastructure-based preventative measures

When securing the infrastructure, you must first understand the components to be secured. As with any vulnerability analysis, start by identifying the components and their external communication paths. (This analysis doesn’t uncover internal application vulnerabilities, but does expose most others.) It’s useful to review a standard WebSphere Application Server topology and see all of the network links and protocols (Figure 2). As someone concerned about security, you need to know about all of these links and focus on securing them. These links represent the coarsest grained system boundaries mentioned earlier.
In Figure 2, the letters on the links indicate the protocols used across those communication links. Each protocol is listed below with its usage and some information on firewall friendliness, since that will be important later. The protocols are:

- **H** = HTTP traffic
  - Usage: Browser to Web server, Web server to app server, admin Web client.
  - Firewall friendly.
- **W** = WebSphere Application Server internal communication
  - Usage: Admin clients and WebSphere Application Server internal server admin traffic.
  - WebSphere Application Server internal communication uses one of several protocols:
    - RMI/IIOP or SOAP/HTTP: Admin client protocol is configurable.
    - File transfer service (dmgr to node agent): Uses HTTP(S).
    - DCS (Distributed Consistency Service): Uses private protocol. Used by memory to memory replication, stateful session EJBs, dynacache, and the high availability manager.
    - SOAP/HTTP firewall friendly. DCS can be firewall friendly.
- **I** = RMI/IIOP communication
  - Usage: EJB clients (standalone and Web container).
  - Generally firewall hostile because of dynamic ports and embedded IP addresses (which can interfere with firewalls that perform Network Address Translation).
- **M** = SIB messaging protocol
  - Usage: JMS client to messaging engine.
As Figure 2 shows, a typical WebSphere Application Server configuration has a number of network links. It is important to protect traffic on each of those links as much as possible to stop intruders. The remainder of this section discusses the steps required to secure the infrastructure just described. The following list is in priority order and details for each item follow. The most important (in fact, critical) items are listed first. As you progress through the list, items become less critical. It is up to you to decide for your organization how far you wish to go on this list.

1. Put the Web server in the DMZ without WebSphere Application Server
2. Separate your production network from your intranet
3. Use HTTPS from the browser
4. Configure secure file transfer
5. Keep up to date with patches and fixes
6. Enable application security
7. Restrict access to WebSphere MQ messaging
8. Protect inter-message engine traffic
9. Harden the Web server and host
10. Remove JDKs left by Web server and plug-in installers
11. Harden proxies
12. Configure and use trust association interceptors carefully
13. Use certificate authentication carefully
14. Consider authenticating Web server to WebSphere Application Server HTTP link
15. Don't run samples in production
16. Choose appropriate process identity
17. Protect configuration files and private keys
18. Encrypt WebSphere Application Server to LDAP link
19. Ensure LTPA cookie flows only over HTTPS
20. Ensure LTPA encryption keys are changed periodically
21. Don’t specify passwords on the command line
22. Create separate administrative user IDs
23. Leverage administrative roles
24. Consider encrypting Web server to Web container link
25. Use only the new LTPA cookie format
26. Encrypt WebSphere MQ messaging links
27. Encrypt distribution and consistency services (DCS) transport link
28. Protect application server to database link
29. Consider restricting cookies to HTTP only
30. Train users to properly understand certificate warnings
31. Carefully limit trusted signers
32. Restrict to strong ciphers
33. Enforce CSiV2 transport SSL use
34. Consider port filtering
35. Disable unused ports
36. Consider disabling password caching
37. Consider enabling FIPS compliance

1. Put the Web server in the DMZ without WebSphere Application Server

In a typical DMZ configuration, there is an outer firewall, the DMZ network containing as little
as possible, and an inner firewall protecting the production internal network.

There are three fundamental principles of a DMZ (see sidebar) that need to be considered here:

- Inbound network traffic from outside must be terminated in the DMZ. A network transparent
  load balancer such as Network Dispatcher doesn’t meet that requirement alone.
- The type of traffic and number of open ports from the DMZ to the intranet must be limited.
- Components running in the DMZ must be hardened and follow the principle of least function
  and low complexity.

Therefore, it is normal to place the Web server in the DMZ and the WebSphere Application Server
application servers inside the inner firewall. This is ideal, as the Web server machine can then
have a very simple configuration and require very little software. It also serves as a point in the
DMZ that terminates inbound requests. And finally, the only port that must be opened on the inner
firewall is the HTTP(S) port for the target application servers. These steps make the DMZ a very
hostile place for an attacker. It is also appropriate to put a secure proxy server in the DMZ instead
of or in addition to the Web server if that is preferred.

If you place WebSphere Application Server on a machine in the DMZ, then far more software
must be installed on those machines and more ports must be opened on the inner firewall so that
WebSphere Application Server can access the production network. This largely undermines the value of the DMZ.

You may choose to place something other than a Web server in the DMZ. It is also reasonable to put a secure proxy into the DMZ, such as IBM Tivoli® Access Manager WebSEAL, the WebSphere Application Server V7 secure proxy, or a hardened appliance such as IBM WebSphere DataPower®. The key is that what you put into the DMZ must NOT be a complex application server, must be hardened, and must terminate inbound connections.

2. Separate your production network from your intranet

Most organizations today understand the value of a DMZ that separates the outsiders on the Internet from the intranet. However, far too many organizations fail to realize that many intruders are on the inside. As mentioned earlier, you need to protect against internal as well as external threats. Just as you protect yourself against the large untrusted Internet, you should also protect your production systems from the large and untrustworthy intranet.

Separate your production networks from your internal network using firewalls. These firewalls, while likely more permissive than the Internet-facing firewalls, can still block numerous forms of attack. After applying this step and the previous step, you should end up with a firewall topology like the one shown in Figure 3. (For more information on WebSphere Application Server firewall port assignments, see Firewall port assignments in WebSphere Application Server.)

Notice that wsadmin has been placed on the edge of the firewall. This is attempting to show that while it is preferred that wsadmin be run from only within the production network (within the protected area), wsadmin access can also be limited to selected addresses, corresponding to administrator desktops, fairly easily through a firewall. Figure 3 also shows EJB clients on the edge, as they might be on either side of the firewall.
A single firewall and not a full DMZ facing the intranet is shown here, as this is the most common topology. However, we increasingly see full DMZs (with a Web server in the internal DMZ) protecting the production network from the non-production intranet. That is certainly a reasonable approach.

3. Use HTTPS from the browser

While LTPA tokens can be transmitted over an unencrypted channel, for maximum protection, it is best that they are sent over an encrypted link. If an LTPA token is successfully captured, the thief can impersonate the user identified until it expires.

If your site performs any authentication or has any activities that should be protected, use HTTPS from the browser to the Web server. If HTTPS is not used, information such as passwords, user activities, WebSphere Application Server session cookies, and LTPA (see sidebar) security cookies can potentially be seen by intruders as the traffic travels over the external network.

For applications that enable HTTP traffic prior to authentication, make sure you pay close attention to cookies. If a cookie (such as the JSESSIONID) is set prior to HTTPS being used, that cookie is a potential risk after HTTPS is used because it might have been altered or captured by an intruder. This is why WebSphere Application Server has a separate cookie for authenticated users. An even more subtle attack is that any page returned over HTTP can be potentially altered by an intruder --
even URLs embedded in the page. Thus, a user might click on a "secure" URL on your page and actually be directed to an intruder's site.

4. Configure secure file transfer

(Addressed by default in V7)

The deployment manager communicates configuration updates to the node agents using the file transfer protocol. In V6.1 and earlier, this is an unauthenticated protocol by default. More precisely, node agents pull admin configuration updates from the deployment manager using an unauthenticated file transfer service. Thus, any foreign client can potentially connect to a deployment manager and upload arbitrary files. If numerous large files are uploaded, the operating system could run out of disk space, resulting in a total failure. It is also theoretically possible to download files that are being replicated from the deployment manager to nodes. However, given the brief and transient nature of these files, this is less likely.

To ensure that the deployment manager only responds to file transfer requests from trusted servers in the cell, you must install the filetransferSecured application and replace the existing insecure application. This application is not normally visible because it is a system application, but it is there. IBM provides a script for this that is documented in the WebSphere Application Server Information Center. In brief, Listing 1 shows are the steps to install the filetransferSecured application (this is a Windows® example, but UNIX® is essentially the same). Listing 1 assumes WebSphere Application Server Network Deployment; if you are using WebSphere Application Server base, the server name is probably server1 rather than dmgr.

Listing 1. Installing filetransferSecured

```bash
cd <profilehome>/bin
wsadmin.bat -user <wasadminuser> -password <waspassword>
wsadmin>source ../../../bin/redeployFileTransfer.jacl
wsadmin>fileTransferAuthenticationOn <your cell name> <dmgr node name> dmgr
wsadmin>$AdminConfig save
```

If you can't remember the cell name and node name, you can find them by looking under the config directory for the profile. Remember that the node is the node of the deployment manager and thus the name likely ends in "CellManager."

5. Keep up to date with patches and fixes

This article assumes you have applied the most recent fixpacks (6.1.0.27 and 7.0.0.7 at the time of writing), as well as all recently published security APARs. These fixpacks and APARs address issues or close other vulnerabilities that are not documented in this article, and so it is critical for you to first ensure you are at recent fix levels and that you have validated your patches for any published vulnerabilities. You should of course continue to keep up with
As with any complex product, IBM occasionally finds and fixes security bugs in WebSphere Application Server, IBM HTTP Server, and other products. It is crucial that you keep up to date on these fixes. It is advisable that you subscribe to support bulletins for the products you use and, in the case of WebSphere Application Server, monitor the security bulletin site for your version. Those bulletins often contain notices for recently discovered security bugs and the fixes. You can be certain that potential intruders learn of those security holes quickly. The sooner you act the better.

More general information on WebSphere Application Server security, including recommendations on hardening the WebSphere Application Server infrastructure, is available on the WebSphere Application Server security page.

6. Enable application security

By default, WebSphere Application Server enables administrative security. Thus, for the most part, the infrastructure provides for reasonable authentication, authorization, and encryption of administrative traffic by default. When administrative security is enabled, the WebSphere Application Server internal links between the deployment manager and the application servers and traffic from the administrative clients (Web and command line) to the deployment manager are encrypted and authenticated (Figure 3). Among other things, this means that administrators will be required to authenticate when running the administrative tools. There are exceptions which will be discussed later.

In addition to leveraging the application server's security for administration, it is strongly recommend that you leverage it for application security. Doing so gives your applications access to a strong and robust standards-based security infrastructure. Applications that did not leverage application server security were typically found to have serious security holes. Designing and implementing a secure distributed infrastructure is not easy.

To enable application security, go to the global security panel and select Enable application security (there is no need to enable Java 2 security; as will be discussed later, it is usually inappropriate), as shown in Figure 4.
Caution: Simply enabling application security does not make your applications secure. It merely makes it possible for your applications to leverage the security features provided by the application server (including Java EE security). Upcoming sections will further address this topic.

7. Restrict access to WebSphere MQ messaging

WEBSphere MQ security warning

Because this article is focused on WebSphere Application Server security, this section is concerned solely with securing the link from the application server to WebSphere MQ. This does not make WebSphere MQ secure. To properly harden WebSphere MQ, substantial additional steps are required.

One option to address this is to implement your own custom WMQ authentication plug-in on the WebSphere MQ server side to validate the user ID and password sent by WebSphere Application Server. A second, and likely simpler technique is to configure WebSphere MQ to use SSL with client authentication and then ensure that only the WebSphere Application Server server possesses acceptable certificates for connecting to WebSphere MQ. (See Securing connections between WebSphere Application Server and WebSphere MQ for more information; this article is a bit dated but the same principles and techniques apply to newer versions of both products. Remember to take into account the product changes to implement the guidance.)

8. Protect inter-message engine traffic
(Addressed by default in V7.)

Prior to V7, the SIBus provides for authentication and authorization of clients by default, but does not require that messaging engines authenticate to each other. This is a security hole because an intruder could potentially pretend to be a messaging engine and compromise the bus traffic. Inter-engine authentication (and implicit authorization) is easily configured on the bus security panel by specifying an authentication alias, as shown in the Figure 5.

**Figure 5. Messaging engine authentication**

![Figure 5. Messaging engine authentication](image)

9. Harden the Web server and host

If you are following the standard topology recommended in step 1, your Web server is running in the DMZ. Since a DMZ is the front line defense against potential intruders, special care must be taken to harden this server.

This article does not discuss the specifics of Web server hardening, but you should look specifically at things like operating system hardening, limiting the Web server modules being loaded, and other Web server configuration steps. (See Apache hardening information and the book *Building Secure Servers with LINUX* for more information.)

When hardening your Web server, there is one WebSphere Application Server specific item that you need to consider. It is possible for the WebSphere Application Server administrative infrastructure to manage Web servers. While this seems like a good thing from an ease of use perspective, this raises serious security issues. There are two ways that a Web server can be managed:

- **Using a managed node** requires that a node agent be placed on the Web server machine (which is typically in the DMZ) and that this agent be part of the WebSphere Application Server cell. This is completely unacceptable from a security perspective and thus should not be used except in those rare cases where there is no need for a DMZ. This is unacceptable for two reasons:
  - First, a node agent is a fully functional member of the cell and thus has full administrative authority. If it is compromised in the DMZ, the entire cell is compromised.
  - Second, WebSphere Application Server is a large and powerful product and therefore can be challenging to harden, as this article shows, and so such products don’t belong in the DMZ.
- **The second approach** requires that IBM HTTP Server be used and that the HTTP admin server be configured. In this scenario, the deployment manager will send administrative
requests to the HTTP admin server running on the Web server host. While better than the other approach, many still consider this to be risky and prefer to have no administrative function in the DMZ.

10. Remove JREs left by Web server and plug-in installers

When you install IBM HTTP Server, the installer leaves behind a JRE. Remove this JRE, as it provides functions that are not needed by the Web server or plug-in under normal conditions. Keep in mind that this will make it impossible to run some tools such as iKeyman on this Web server. This isn’t a significant issue because running such tools in the DMZ is problematic anyway.

When you install the WebSphere Application Server HTTP Server plug-in using the IBM installer, it also leaves behind a JRE. As before, you should remove this post install.

If you do decide to remove the JREs, you should make a backup copy just in case for future use. One technique might be to zip or tar the JRE and replace it later, such as when it would be needed for the WebSphere Application Server update installer when applying fixes, and then zip/tar and remove it again when the process is complete.

11. Harden proxies

If you have chosen to use a secure proxy server in the DMZ instead of (or in addition to) a Web server, the previous advice applies to the proxy, although specific details won’t be provided here as hardening is proxy specific.

One important point about WebSphere DataPower: Web security proxies are typically not appropriate for proxying Web services traffic because they cannot block the types of threats that are possible when transmitting XML. In order to provide secure Web services (or any XML-based protocol), use an XML firewall such as WebSphere DataPower.

12. Configure and use trust association interceptors carefully

TAIs are often used to enable WebSphere Application Server to recognize existing authentication information from a Web SSO proxy server, such as Tivoli Access Manager WebSEAL. Generally, this is fine. However, be careful when developing, selecting, and configuring TAIs. A TAI extends the trust domain. WebSphere Application Server is now trusting the TAI and whatever the TAI trusts. If the TAI is improperly developed or configured, it is possible to completely compromise the security of WebSphere Application Server. If you custom-develop a TAI, ensure that the TAI
WebSphere Application Server V7 advanced security hardening, Part 1: Overview and approach to security hardening

carefully validates the parameters passed in the request and that the validation is done in a secure way. We've seen TAIIs that perform foolish things such as simply accepting a username in an HTTP header. That's useless unless care is taken to ensure that all traffic received by WebSphere Application Server must be sent via the authentication proxy, for example, using the techniques described above, and that the authentication proxy will always override an HTTP header set by the client because HTTP headers can be forged.

- **WebSEAL TAI configuration**
  To make clear the importance of careful configuration, this article specifically discusses the IBM provided legacy WebSEAL TAI, but any TAI requires careful design and configuration to be secure. Proper configuration of the trust relationships between WebSphere Application Server and Tivoli Access Manager WebSEAL is crucial to the creation of a secure configuration. In order to create this secure configuration, steps must be taken in both WebSphere Application Server and WebSEAL. Within WebSphere Application Server, both the Web container configuration and the WebSEAL TAI configuration must be set properly. The trust relationship between the two products is crucial because the WebSEAL TAI within WebSphere Application Server is accepting identity assertions from WebSEAL. If that link can be compromised, an intruder could then assert any identity and completely undermine the security of the infrastructure. The trust relationship between WebSphere Application Server and WebSEAL can be established through one of two mechanisms: mutual SSL authentication and password-based authentication. Either mechanism is appropriate within a secure environment. However, each must be configured properly or serious security breaches are possible. In either case, WebSEAL sends the end user's user ID as an iv-user header on the HTTP request. The difference between the two configurations is in how WebSEAL proves itself to the application server.

- **WebSEAL password configuration**
  When password authentication is used, WebSEAL sends its user ID and password as the basic auth header in the HTTP request (the user's user ID is in the iv-user header). Password-based authentication is configured in two places. First, WebSEAL must be configured to send its user ID and password to the application servers for the junction being configured. This password is of course a secret that must be carefully protected. The WebSEAL TAI will validate this password against the registry when it is received. However, there is one subtle and easily overlooked point. If the LoginId property is not set on the WebSEAL TAI properties, then the TAI will verify the user ID and password combination sent from WebSEAL and trust it if it is any valid user ID and password combination. This is not a secure configuration because this then implies that any person knowing any valid user ID and password combination can connect to WebSphere Application Server and assert any user's identity. When you specify the LoginId property, the WebSEAL TAI will ignore the inbound user ID in the basic auth header and verify the LoginId and WebSEAL password combination. In that case, there is then only one (presumably closely guarded secret) valid password that can be sent from WebSEAL. You should of course configure SSL from WebSEAL to the application server to ensure that the secret password is not sent in cleartext.

- **WebSEAL mutualSSL configuration**
  Mutual SSL is configured through three separate and critically important steps:
1. WebSEAL must be configured to use SSL to communicate with WebSphere Application Server and that SSL configuration must include a client certificate known only to the application server Web container.

2. The application server Web container must be configured to perform client certificate authentication, and its trust store must be altered to include only the client certificate that WebSEAL is using. This step is crucial because this is how you guarantee that requests to the application server Web container are coming only from WebSEAL and not some intruder (simply using mutually authenticated SSL is not sufficient). You must also remove the non-HTTPS transports from the Web container to ensure that only mutually authenticated SSL is used when contacting the server.

3. The WebSEAL TAI must be configured with `mutualSSL=true` in its properties. However, you must understand that this last step merely states to the TAI that it should assume the connection is secure and that it is using mutually authenticated SSL. If either of the two previous steps are not configured exactly correctly, your environment is now completely insecure.

   Thus, the choice to use mutualSSL must be taken with great care. Any configuration mistakes will result in an environment where any user can be impersonated.

   If you add a Web server to the mix, things get even more complicated. In this case, you must carefully configure a mutualSSL configuration between WebSEAL and the Web server and a second between the Web server plug-in and the WebSphere Application Server Web container.

**Multiple WebSEAL TAIIs**

There are currently three different TAIIs that can be used to provide SSO between WebSEAL and WebSphere Application Server:

- **Legacy WebSEAL TAI** (WebSealTrustAssociationInterceptor) that ships with WebSphere Application Server: Avoid using this TAI because it is deprecated in V7 and can be configured dangerously insecurely, as described above.

- **Tivoli Access Manager Interceptor Plus TAI** (TAMTrustAssociationInterceptorPlus) that ships with WebSphere Application Server. This TAI addresses the security vulnerabilities in the previous TAI and is preferred. However, it has some functional differences relative to the legacy TAI (including a requirement for a TAM client configuration), so some users prefer not to use it.

- **Enhanced Tivoli Access Manager TAI** (TAMETai) that can be downloaded from IBM. This TAI is properly hardened like the TAMPlus TAI but includes substantial additional function, including the ability to run without a Tivoli Access Manager client, just like the legacy WebSEAL TAI.

   In general, you should use the second or third TAI as appropriate to your needs.

13. **Use certificate authentication carefully**
Certificate authentication raises two very specific risks:

- **Revocation**: Certificates can be compromised and provisions must be made to revoke compromised certificates.
  Certificates provide for a powerful form of authentication and are highly desirable from security perspective. However, you must take into account the problem of revocation. Since users control their private key, there is always the risk that it could be compromised. Therefore, all CAs provide for certificate revocation; basically the CA is stating that the certificate should no longer be trusted. For certificate revocation to work properly, the receiver of the certificate must check to see if it is still valid. Many people overlook this. Without proper support for revocation, using certificates for authentication is foolhardy. There are a variety of techniques that are used (and will not be discussed here in detail here); in brief, you can choose from:
  - Certificate Revocation List, found via a single end point or distribution point information embedded in the certificate.
  - Self Signed Certificates, if you have a small number of certificates, you can simply issue self signed certificates. Revocation is then just a matter of deleting the corresponding signer.

  All of these techniques are supported by WebSphere Application Server, but all require configuration. Make sure you take steps to do so.

- **Web authentication trust risk**: The mechanism used for validating certificates must be configured securely; by default, it is not for Web traffic.
  When certificate-based authentication is used for Web authentication, a very subtle potential trust issue occurs. When a Web client authenticates to the Web server, the Web server validates the certificate. Then, the WebSphere Application Server Web server plug-in forwards the certificate information from the Web server to the application server. This information is forwarded so that the Web container can map that certificate to a Java EE identity. The problem is that the information is just a description of the certificate (information that is available in the public certificate). If an intruder can connect directly to the Web container and bypass the Web server, the system is now vulnerable to compromise because the intruder could forge certificate information and trick the runtime, enabling them to become anyone. This means that if you are using certificates for authentication (Java EE based authentication or custom application code directly examining the certificate), you must block the vulnerability.
  There are two cases to consider. If your intention is to use certificates to authenticate to the Web server and then have those certificates be available to the Web container for authentication, you will need to authenticate the Web server to Web container link (see the next section). If your intention is to use certificates to authenticate directly to the Web container (meaning there is no Web server in your scenario), you'll have to configure the Web container to ignore certificate information in the HTTP headers (in this scenario, such information would always be a forgery). To do this, you must configure the "trusted" custom property on the Web container for each application server and set its value to false as shown in Figure 6.
If your goal is to support certificate authentication to the Web server and the Web container, custom code will be required because neither of the solutions just mentioned are sufficient; both are vulnerable to attacks from the other connection path. Instead, a custom TAI or application code will need to be developed to leverage IBM specific features that make it possible for code running in the Web container to determine if the certificate information available via the Java EE APIs is that of someone connecting directly to the Web container (and thus trustworthy) or derived from HTTP headers (in which case not inherently trustworthy). If it’s the latter case, custom code can look directly at the certificate information presented to the container as part of the SSL handshake and validate if the party that set the HTTP headers is trustworthy; for example, custom code can examine the SSL client certificates (available via the request property com.ibm.websphere.ssl.direct_connection_peer_certificates) to see if the direct container connection is from the WebSphere Application Server plug-in and, if it is, then choose to accept the asserted certificate information in the HTTP headers. This feature was added in 7.0.0.7 and is documented in the WebSphere Application Server Information Center.

14. Consider authenticating Web server to WebSphere Application Server HTTP link

The WebSphere Application Server Web server plug-in forwards requests from the Web server to the target application server. By default, if the traffic to the Web server is over HTTPS, then the plug-in will automatically use HTTPS when forwarding the request to an application server, thus protecting its confidentiality.

Further, with some care, you can configure the application servers (which contain a small embedded HTTP listener) to only accept requests from known Web servers. This prevents various sneak attacks that bypass any security that might be in front of or in the Web server and creates a trusted network path. This type of situation might seem unlikely, but is in fact very possible. Some examples, which by no means is all inclusive are:
• You have an authenticating proxy server that just sends the user ID as an HTTP header without any authentication information. An intruder that can access the Web container directly can become anyone simply by providing this same header. (IBM Tivoli Access Manager WebSEAL does not have this weakness.)
• You have a proxy server that performs important authorization to limit who can access what applications at a coarse grained level.
• You have a proxy server that performs critical auditing and do not want that bypassed.
• You are using client certificate authentication to the Web server as discussed in the previous section.

To create a trusted network path from the Web server to the application server, you configure the application server Web container SSL configuration to use client authentication. Once you have ensured that client authentication is in use, you need to ensure that only trusted Web servers can contact the Web container. To do this, you must limit the parties that have access by applying the SSL trick from Limiting access using only SSL. Specifically, in this case you'll need to:

1. Create a key store and trust store for the Web container and a key store for the Web server plug-in.
2. Delete from all key stores (including the trust store) all of the existing signing certificates. At this point, no key store can be used to validate any certificates. That's intentional.
3. Create a self-signed certificate in the two key stores and export just the certificate (not the private key). Make sure you track when those certificates expire. Once the plug-in certificate expires, it will no longer be able to contact the Web container! Import the certificate exported from the Web container key store into the plug-in key store. Import the plug-in certificate into the Web container trust store. Now, each side contains only a single signing certificate. This means that each can be used to verify exactly one certificate -- the self-signed certificate created for the peer.
4. Install the newly created key stores into the Web container and Web server plug-in.

15. Don't run samples in production

WebSphere Application Server ships with several excellent examples to demonstrate various parts of WebSphere Application Server. These samples are not intended for use in a production environment. Do not run them there, as they create significant security risks. In particular, the showCfg and snoop servlets can provide an outsider with tremendous amounts of information about your system. This is precisely the type of information you do not want to give a potential intruder. This is easily addressed by not running server1 (which contains the samples) in production or not installing the samples during the profile creation. If you are using WebSphere Application Server base, you'll actually want to remove the examples from server1.

16. Choose appropriate process identity

WebSphere Application Server V7 advanced security hardening, Part 1: Overview and approach to security hardening
The WebSphere Application Server processes run on an operating system and must therefore run under some operating system identity. There are three ways to run WebSphere Application Server with respect to operating system identities:

- Run everything as root.
- Run everything as a single user identity, such as "was."
- Run the node agents as root and individual application servers under their own identities.

IBM tests for and fully supports the first two approaches. The third approach might seem tempting because you can then leverage operating system permissions, but it isn't very effective in practice for these reasons:

- It is very difficult to configure and there are no documented procedures. Many WebSphere Application Server processes need read access to numerous files and write access to the log and transaction directories.
- By running the node agent as root, you effectively give the WebSphere Application Server administrator and any applications running in WebSphere Application Server root authority.
- The primary value of this approach is to control file system access by applications. This can be achieved just as well using Java 2 permissions.
- This approach creates the false impression that applications are isolated from each other. They are not. The WebSphere Application Server internal security model is based on Java EE and Java 2 security and is unaffected by operating system permissions. Thus, if you choose this approach to protect yourself from "rogue" applications, your approach is misguided.

The first approach is obviously undesirable because, as a general best practice, it is best to avoid running any process as root if it can be avoided. This leaves the second approach, which is fully supported. In rare cases, where application isolation is not a concern but running applications under different operating system identities for accounting purposes is desirable, the third approach can be used. This has no value from a security perspective and in fact slightly increases risks, but does make it possible to use operating system level accounting.

17. Protect configuration files and private keys

Don't take the idea of tightening permissions too far in development. We've seen far too many cases where developers -- during development -- aren't permitted to even view application server log files. Such ultra-sensitivity is unwarranted. During production, of course, you should lock down WebSphere Application Server as much as possible. During development, however, maximum security is neither necessary nor productive.

You should limit file system access to WebSphere Application Server's files by leveraging operating system file permissions. WebSphere Application Server, like any complex system, uses and maintains a great deal of sensitive information. In general, almost no one should have read or write access to most of the WebSphere Application Server information (see sidebar). In particular, the WebSphere Application Server configuration files (<root>/config) contain configuration information as well as passwords.
Also, be careful to avoid incidental sharing of these keys. For example, do not use the same keys in production as in other environments. Many people will have access to development and test machines and their private keys. Guard the production keys carefully.

If you do not carefully control who has write access to the file system, a user can subvert the product security controls (such as auditing) by simply hand editing the configuration files.

18. Encrypt WebSphere Application Server to LDAP link

When using an LDAP registry, WebSphere Application Server verifies a user's password using the standard `ldap_bind`. This requires that WebSphere Application Server send the user's password to the LDAP server. If that request is not encrypted, a hacker could use a network sniffer to steal the passwords of users authenticating (including administrative passwords!). Most LDAP directories support LDAP over SSL, and WebSphere Application Server can be configured to use this. On the LDAP user registry panel (Figure 7), check the use SSL enabled option and then configure an SSL configuration appropriate to your LDAP directory. You'll most likely need to place the signing key for the LDAP server's certificate) into the trust store. It's best to create a new SSL configuration just for LDAP to avoid causing problems with the existing SSL usage.

Figure 7. Enable LDAP SSL

If you use a custom registry, you'll obviously want to secure this traffic using whatever mechanism is available.

19. Ensure LTPA cookie flows only over HTTPS

Web applications use cookies to track users across requests. These cookies, while typically not sensitive in themselves, connect you to your existing state on the back end system. If an intruder were to capture one of your cookies, they could potentially use the cookie to act as you. Since network traffic is often traveling over untrusted networks (consider your favorite WiFi hotspot), where capturing packets is quite easy, important Web traffic should be encrypted using SSL. This includes important cookies. Clearly, if SSL is used for all requests, the cookies are protected. However, many applications (perhaps accidentally) make some requests over HTTP without SSL,
potentially exposing cookies. Fortunately, the HTTP specification makes it possible to tell the browser to only send cookies over SSL.

In the case of WebSphere Application Server, the most important cookie is the LTPA cookie, and therefore it should be configured to be sent only over SSL.

**Figure 8. Limit LTPA cookies to SSL only**

You can also similarly limit the HTTP Session (JSESSION by default) cookie to SSL only as well by specifying a similar setting on the session management panel.

**Caution:** The Requires SSL flag does not work in WebSphere Application Server V7 prior to APAR PM00610. Make sure you install it.

20. Ensure LTPA encryption keys are changed periodically

WebSphere Application Server encrypts various user tokens (including the LTPA cookie) using what are known as the LTPA encryption keys. As with any cryptographic keys, these should be changed periodically. Depending on your WebSphere Application Server version and patch level, automatic key generation might be on or off by default; the more recent the version, the more likely it is off by default.

You should ensure that your LTPA keys are changed periodically. You can do this by either enabling the automatic LTPA key replacement as shown in Figure 9, or you can manually regenerate the keys as shown in Figure 10. Whichever approach you choose, make sure that you consider the problems of LTPA keys becoming inconsistent:

- First, if some nodes in your cell are left down for extended periods of time (likely twice the key lifetime), the nodes can lose the ability to communicate.
• Second, if your LTPA keys are shared with something else (such as another cell or WebSphere DataPower), then when the keys change, you'll need to update them everywhere -- typically causing an outage.

**Figure 9. Enabling automatic LTPA key updates**

**Figure 10. Manually generating new LTPA keys**

21. Don't specify passwords on the command line

Once security is enabled, the WebSphere Application Server administrative tools require that you authenticate in order to function. The obvious way you would think to do this is to specify the user ID and password on the command line as parameters to the tools. Do not do this. This exposes your administrative password to anyone looking over your shoulder. And, on many operating systems, anyone that can see a list of processes can see the arguments on the command line. Instead, you should ensure that the administrative tools prompt for a user ID and password. As
of WebSphere Application Server V6.0.2, all of the administrative tools automatically prompt for a user ID and password if one isn't provided on the command line. No further action is required.

If you are using an earlier version of WebSphere Application Server, you can force the tools to prompt by telling them to use RMI (the default is SOAP) communication. The RMI engine prompts when needed. To do this, simply specify:

-connertype RMI -port <bootstrap port>

Here's an example of starting wsadmin this way to connect to a deployment manager listening on the default port:

wsadmin.sh -connectype RMI -port 9809

If you find it annoying that the command line tools prompt graphically for a user ID and password, you can override this behavior and force the tools to use a simple text-based prompt. To do this, you must change the loginSource from prompt to stdin by editing the appropriate configuration file. By default, the administrative tools use SOAP and thus the soap.client.props file should be edited. If you are using RMI, edit sas.client.props. Look for the loginSource property in the appropriate file and change it to specify stdin.

22. Create separate administrative user IDs

The primary administrative ID is not the same as the security server ID that is used for server to server communication. As of V6.1, that identity no longer has to exist in the registry or even have an associated password. It's just used for internal communication by default. You can specify a server ID and password if needed, but this is not recommend except when absolutely necessary — when working with a mixed version cell (including V6.0 and earlier) or cross cell SSO with V6.0 or earlier.

When security is configured for WebSphere Application Server V6.1 and later, a single primary administrative ID is initially configured (see sidebar). This ID is effectively the equivalent of root in WebSphere Application Server and can perform any administrative operation (including all administrative roles mentioned in the next section). Because of the importance of this ID, it is best not to widely share the password. Ideally, this ID should never be used after initial configuration.

As with most systems, WebSphere Application Server does permit multiple principals to act as administrators. Simply use the administrative application and go to the system administration console Users (or Groups) section to specify additional users or groups that should be granted administrative authority. When you do this, each individual person can authenticate as himself or herself when administering WebSphere Application Server. As of WebSphere Application Server V5.0.2, all administrative actions that result in changes to the configuration of WebSphere Application Server are audited by the deployment manager, including the identity of the principal that made the change. As of V7, a new auditing framework has been introduced that can provide
even more detailed information on administrative actions. Obviously, these audit records are more useful if each administrator has a separate identity.

The approach of giving individual administrators their own separate administrative access can be particularly handy in an environment where central administrators administer multiple WebSphere Application Server cells. You can configure all of these cells to share a common registry, and thus the administrators can use the same ID and password to administer each cell, while each cell has its own local "root" ID and password.

23. Leverage administrative roles

WebSphere Application Server allows for a variety of administrative roles depending on the version: Administrator, Operator, Monitor, Configurator, AdminSecurityManager, iscadmins, Deployer, Auditor. These roles make it possible to give individuals (and automated systems) access appropriate to their level of need. It is strongly recommend that you take advantage of roles whenever possible. By using the less powerful roles of monitor and operator, you can restrict the actions an administrator can take. For example, you can give the less senior administrators just the ability to start and stop servers and the night operators just the ability to watch the system (monitor). These actions greatly limit the risk of damage by trusting people with only the permissions they need.

Complete documentation on the roles and the authorities they have is available in the WebSphere Application Server Information Center. However, pay particular attention to these three interesting roles:

- **Monitor**: By giving a user or system this access level, you are giving only the ability to monitor the system state; the state cannot be changed, nor can the configuration be altered. For example, if you develop monitoring scripts that check for system health and have to store the user ID and password locally with the script, use an ID with the monitor role. Even if the ID is compromised, little serious harm can result.
- **AdminSecurityManager**: (added in V6.1) Users with this role have the ability to grant other users administrative roles. The Administrator role itself does not have that authority. Now, you can grant people various authorities (even Administrator authority) and still know that they cannot grant those authorities to others.
- **Auditor**: (added in V7) Users with this role can configure the auditing system and nothing else. Administrators on the other hand can configure anything but the auditing system. This provides a clear separation of duties. An administrator can change the configuration, but cannot “wipe out” his tracks because he can’t disable auditing.

24. Consider encrypting Web server to Web container link
Even if you have chosen not to authenticate the link from the Web server to the Web container, you might want to consider encrypting it. The Web server plug-in transmits information from the Web server to the Web container over HTTP. If the request arrived at the Web server using HTTPS, the plug-in will forward the request on using HTTPS by default. If the request arrived over HTTP, HTTP will be used by the plug-in. These defaults are appropriate for most environments. There is, however, one possible exception.

In some environments, sensitive information is added to the request after it has arrived on your network. For example, some authenticating proxy servers (such as WebSEAL) augment requests with password information. Custom code in the Web server might do something similar. If that’s the case, you should take extra steps to protect the traffic from the Web server to the Web container.

To force the use of HTTPS for all traffic from the plug-in, simply disable the HTTP transport from the Web container on every application server and then regenerate and deploy the plug-in (Figure 11). Now, the plug-in can only use HTTPS and so it will use it for all traffic regardless of how the traffic arrived at the Web container.

**Figure 11. Ensuring HTTPS only**

<table>
<thead>
<tr>
<th>Application servers &gt; server2 &gt; Web container transport chains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use this page to view and manage a transport chain. Transport chains represent network protocol stacks that are operating within a client or server.</td>
</tr>
</tbody>
</table>

25. **Use only the new LTPA cookie format**

(Addressed by default in V7.)

WebSphere Application Server V5.1.1 introduced a new LTPA cookie format (LTPAToken2) to support subject propagation. While doing this, some theoretical weaknesses in the old format were also addressed. Bear in mind that these weaknesses are theoretical; no known compromises have occurred. Nonetheless, it is desirable to use the new stronger format unless you must use the older format.

The new LTPA token uses the following strong cryptographic techniques:
• Random salt.
• Strong AES ciphers.
• Data is signed.
• Data is encrypted.

For the curious, a 1024 bit RSA key pair is used for signing and a 128 bit secret key (AES) is used for encryption. The cipher used for encryption is AES/CBC/PKCS5Padding.

Both the old and new cookie format are supported simultaneously by default prior to V7 to ensure compatibility with older versions of WebSphere Application Server, IBM Lotus® Domino® (prior to V8), and Tivoli Access Manager WebSEAL (prior to V6) when it creates LTPA cookies. If you don't need this compatibility, you should disable it. To do so, navigate to the Security > Authentication Mechanisms > LTPA > SSO configuration panel and unselect interoperability mode (Figure 12).

Figure 12. SSO interoperability mode setting

Caution: Prior to V7, you cannot disable interoperability and security attribute propagation (also known as subject propagation) at the same time. With both disabled, authentication will fail.

26. Encrypt WebSphere MQ messaging links

If you are using WebSphere MQ rather than the default messaging provider, you should of course use SSL to WebSphere MQ. See Resources for more information on that. In WebSphere Application Server V7, WebSphere MQ client SSL configuration is a first class construct and can be done using the admin console much like other SSL configuration.
27. Encrypt distribution and consistency services (DCS) transport link

Core groups rely on DCS, which uses a reliable multicast message (RMM) system for transport. RMM can use one of several wire transport technologies. Depending on your environment, sensitive information might be transmitted over DCS. For example, data in DynaCache and the security subject cache are transmitted using DCS. To ensure this, select a transport type of channel framework and DCS-Secure as channel chain for each core group (Figure 13).

**Figure 13. Configuring DCS to use a protected link**

![Image of DCS configuration]

Be aware that DCS always authenticates messages when global security is enabled. Once the transport is encrypted, you then have a highly secure channel.

Once you have done this, all services that rely on DCS are now using an encrypted and authenticated transport. Those services are DynaCache, memory-to-memory session replication, core groups, Web services caching, and stateful session bean persistence.

28. Protect application server to database link

Just as with any other network link, confidential information can be written to or read from the database. Most databases support some form of network encryption and you should leverage it.
29. Consider restricting cookies to HTTP only

If hackers are able to compromise a Web application by inserting malicious JavaScript into the browser (this is commonly known as cross-site scripting), then they can do any number of malicious actions and the application is essentially compromised. One of the many malicious things they can do is steal sensitive cookies such as the LTPA cookie. Most recent Web browsers support a concept known as HTTP Only cookies.

WebSphere Application Server provides a way to ensure that the LTPA cookie is marked as HTTP Only. This is enabled by setting this security custom property to true: com.ibm.ws.security.addHttpOnlyAttributeToCookies. (This was added fairly recently via APAR PK82764 (V6.0 or V6.1) or PM03760 (V7).)

At present, this property only protects the LTPA cookie with the HTTP Only flag. For a properly written application that leverages Java EE security and enables session security (discussed later), that should be sufficient.

However, an upcoming APAR (PK98436) will make it possible to set the HTTP Only flag on arbitrary cookies. Use this new feature when it becomes available rather than the older feature, as it is more flexible and more complete. With this APAR, the cookies to protect are controlled by a Web container property, com.ibm.ws.webcontainer.httpOnlyCookies. That property is a comma separated listed of cookies to protect (with * indicating all cookies).

**Caution:** While this APAR may seem to be a solution for cross-site scripting, it is not. If a hacker can execute arbitrary code in your browser, he can do far more damage than just steal your cookies. They can actually see everything on the browser screen and capture every key stroke -- far more disturbing than stealing a cookie.

30. Train users to properly understand certificate warnings

When using SSL for communication, a key element of the secure exchange is validation of the server’s certificate against the client trust store. Should the server present a certificate that is not trustworthy because the corresponding signer is not in the trust store, most clients (Web browsers, SSH, wsadmin, and so on) will prompt the user to decide what to do. These clients typically warn the user about the unknown certificate and present a fingerprint (typically SHA) for the certificate and ask **should I trust this?** Unfortunately, most users blindly click **yes**. That’s a terrible decision. If you do this, you have no idea what server you are talking to. And, in the case of WebSphere
Application Server administrative clients, the next thing you will do is send your administrative user ID and password to an unknown source.

Instead, administrators (and ideally end users) should examine the fingerprint information and determine if it is the correct fingerprint. The admin tools provide a way to see the fingerprint for certificates. Selecting a personal certificate in the admin console displays its fingerprint.

**Figure 14. Certificate fingerprint**

![Certificate fingerprint diagram](image)

Users (particularly administrators) should be made aware of the fingerprint information and then, ideally, should validate it when prompted by a client, be it a Web browser or wsadmin.
**Figure 15. Web server certificate warning, Part 1**

![Certificate warning]

**Figure 16. Web server certificate warning, Part 2**

![Certificate viewer]

---

WebSphere Application Server V7 advanced security hardening,  
Part 1: Overview and approach to security hardening
**Listing 2. wsadmin certificate warning**

```
./wsadmin.bat
*** SSL SIGNER EXCHANGE PROMPT ***
SSL signer from target host localhost is not found in trust store
C:/IBM/WebSphere/AppServer/profiles/AppSrv02/etc/trust.p12
Here is the signer information (verify the digest value matches what
is displayed at the server):
Subject DN:    CN=keysbotzum, O=IBM, C=US
Issuer DN:     CN=keysbotzum, O=IBM, C=US
Serial number: 1151337276
Expires:       Tue Jun 26 11:54:36 EDT 2007
Add signer to the trust store now? (y/n)
```

Even if you don't follow this advice, at least when you get this challenge the first time, import the certificate into the client trust store. If you get message again, **find out why!** The prompt should not happen again until the certificate is changed. If you get that prompt unexpectedly, something could be terribly wrong.

### 31. Carefully limit trusted signers

When using certificate authentication (client or server), you need to understand that each signer in the trust store represents a trusted provider of identity information (a certificate). You should trust as few signers as possible. Otherwise, it is possible that two signers might issue certificates that map to the same user identity. That would create a serious security hole in your architecture.

You should review the trust stores on the clients and servers and remove any signers which are not needed. By default, the trust stores contain far fewer trusted signers than previous releases, in keeping with the goal of being secure by default. There are, however, still a few signer issues you might wish to address:

- Be default, the dummyclientsigner and dummyserversigner are in the trust stores for compatibility with cells from previous releases that use those defaults (which we have always recommended against). These are not present by default in V7.
- By default, the KDB/CMS key stores contain signers for most well known CAs. There is no good or useful reason for this and so they should be removed.
- In V7, the default cell trust store contains a WebSphere DataPower signing certificate, meaning that all DataPower machines can issue certificates that the application servers will trust. This should be removed for maximum security.

### 32. Restrict to strong ciphers

(Addressed by default in V7.)
When communication occurs over SSL, traffic is encrypted. For this traffic to be best protected, strong cryptographic ciphers should be used. Unfortunately, prior to V7, the default SSL cipher selection of strong includes some notably weak ciphers. You should remove those ciphers or a client might select one of them. Normally, clients select strong ciphers implicitly if the Web server supports them, but it is best to be sure.

**Figure 17. SSL ciphers**

![Cipher suite settings]

While it will not be covered here, you should also ensure that your Web server is configured to only accept traffic using strong ciphers.

**33. Enforce CSIv2 transport SSL use**

When WebSphere Application Server servers and clients communicate using CSIv2 IIOP, they negotiate the transport security. Whatever is acceptable to both parties is chosen. Generally, that is fine, but you should be aware of one potential weakness. WebSphere Application Server supports CSIv2 over SSL or clear text. By default, both parties will typically negotiate to use SSL, thus ensuring an encrypted communication channel. However, if either party in the negotiation requests clear text, then clear text will be used. You might not even realize your traffic is being sent in the clear! This might happen, for example, if a client was mis-configured. If you want to guarantee that traffic is encrypted (and you should), it is safer to ensure that SSL is always used.

You can ensure that SSL is used for IIOP by indicating that it is required, and not optional, on the CSIv2 inbound transport panel. You should do the same for the CSIv2 Outbound Transport (Figure 18).
Figure 18. Configuring SSL only

34. Consider port filtering

Sometimes it is desirable to limit who can connect based upon network information. While such configuration provides questionable security value, it is included here for completeness. Most of the transports in WebSphere Application Server (with the exception of IIOP) leverage the Channel Framework, which in turn makes it possible to filter inbound traffic based on IP address or DNS name using an inclusion or exclusion list.
Figure 19. Port filtering

Caution: IP address forgery is a fairly easy thing to do, and so relying on IP addresses for security is unwise. It is particularly unwise to filter based on IP address in WebSphere Application Server. Firewalls and switches are better equipped to recognize when packets are coming from IP addresses that could not be valid. They can also check MAC addresses.

35. Disable unused ports

A basic principle of security hardening is to minimize the attack surface for potential attacks. This is even true when there are no known security issues with a given service; if the service is not required for the system to correctly function, it should be removed to minimize the likelihood of an attacker taking advantage of this additional function at some point in the future. Examine Figure 20 and you'll see that a typical WebSphere Application Server application server is potentially listening on a lot of ports.
Figure 20. Default ports for a Network Deployment application server

Should a given service not be required, then its listening ports can be disabled. Looking at this list, potential candidates for disabling are:

- **SAS_SSL_SERVERAUTH_LISTENER_ADDRESS**: Used for compatibility with WebSphere Application Server V4 and earlier releases. This is the old IIOP security protocol. CSIv2 replaces it as of V5.
- **SIB_ENDPOINT_***: These are used by the built-in messaging engine. If you aren’t using messaging, then you don’t need either.
- **SIB_MQ_***: These are used by the messaging engine when connecting with WebSphere MQ.
- **WC_adminhost**: These are used for administrative Web browser access. If your application server is not a deployment manager, you should ensure that these are disabled. Unfortunately, most application server Web containers are listening on two administrative ports even though there is no administrative application on those servers. This is because servers are usually created based on the server1 template that includes those ports. You should disable or remove those ports from all of your application servers.
• **WC_defaulthost**: These are the default Web container listening ports. If you’ve added custom listener ports, then these might not be needed.

Different ports require different techniques for disabling them depending on how they are implemented:

- The SAS_SSL_SERVERAUTH_LISTENER_ADDRESS can be taken out of service by selecting CSI as the active protocol on the global security panel. SAS is disabled by default in V7 although the port is still listed.
- The WC_* ports are all for the Web container. They can best be removed, modified, or disabled from the Web container transport chain configuration panel (Application servers > servername > Web container > transport chain). The only listening Web ports you need are those used by your applications.
- The SIB_* ports are not started unless the messaging engine is enabled, so no action is necessary for them.

**Caution:** Use extreme care when determining which ports to disable and when actually disabling them. Otherwise, you might inadvertently disable one of the administrative ports, which will leave you without a mechanism to administer the process -- short of deleting and recreating the process (for example, server) definition.

36. **Consider disabling password caching**

When password based authentication occurs, the runtime caches the password as a one way hash for future validation. Since the hash is not reversible, there is no danger in the password being captured (even from a memory dump) but this cache does have implications. When future requests arrive that require authentication, and they use the same user ID and password combination, the cached password data (and user information) will be used. This means that if a user’s password is changed in the registry, he will still be able to authenticate using the old password until the cache expires (10 minutes by default).

Some consider this to be a security vulnerability (the author is not among them). If this concerns you, you can disable password caching with the JVM level property setting of `com.ibm.websphere.security.util.authCacheEnabled = BasicAuthDisabled`. Once set, passwords will no longer be cached and all password authentications will result in a query of the registry. Keep in mind that this can have significant negative performance implications. If you are using a protocol that is stateless and authenticates on every request (such as WS-security with UserNameTokens), this can generate heavy registry authentication traffic.

37. **Consider enabling FIPS compliance**
There are a variety of cryptographic algorithms to choose from when performing encryption. In addition, when establishing SSL connections there are, in fact, three choices: SSL V2 (disabled by default), SSL V3, and TLS. The United State Government has defined standards with respect to computer security (Federal Information Processing Standards) that cover a number of areas. As part of these standards, they specifically certify particular ciphers as being FIPS compliant and have also certified TLS (but not SSL V3).

Users wishing to restrict their applications to use FIPS compliant ciphers and TLS can either manually configure every endpoint, or simply enable FIPS compliance using the admin tools. Once FIPS is enabled, only FIPS compliant cryptography will be used.

**Summary**

Part 1 of this two-part article discussed several aspects of security, focusing on the core theme of hardening a WebSphere Application Server environment. Hopefully, this information is providing you with the foundation you need to truly secure your Java EE environment.

**Part 2** will cover even more ground, including application-based preventative measures, cell trust, cross-cell trust, administrative and application isolation, identity propagation, desktop development security, and much more.

If you are interested in learning more about WebSphere Application Server security, contact IBM Software Services for WebSphere for a customized on-site class in WebSphere Application Server security. The class covers security hardening, customizing authentication, integration, single sign on, and a variety of other related topics in depth.

**Acknowledgements**

I would like to thank my colleagues Martin Lansche, Bill Hines, Bill O'Donnell, Paul Illechko, Simon Kapadia, and Tom Alcott for their valuable input and assistance.
Resources

- WebSphere Application Server V7 advanced security hardening, Part 2: Advanced security considerations
- WebSphere Application Server V6 advanced security hardening, Part 1
- WebSphere Application Server V6 advanced security hardening, Part 2
- Rational AppScan information
- The Top 5 Internal Security Threats by Cindy Waxer
- Enterprise Application Security
- Redbook: IBM WebSphere 6.0 Security, SG-246316, IBM Corp, 2005
- Buffer Overflows – What Are They and What Can I Do About Them? by Larry Rodgers, CERT, 2001
- Understanding Malicious Content Mitigation for Web Developers
- Java Security Coding Guidelines
- Messaging security
- Securing connections between WebSphere Application Server and WebSphere MQ
- IBM Security Scanner Tool for WebSphere Application Server from IBM support
- Advanced Authentication in WebSphere Application Server
- Database identity propagation in WebSphere Application Server V6
- Apache hardening information (applies to IHS as well)
- Building Secure Servers with LINUX
- Firewall Port Assignments in WebSphere Application Server
- The (XML) threat is out there
- DB2 Technical Tip: Setting up SSL
- FIPS Configuration
- SSL, Certificate, and Key Management Enhancements in WebSphere Application Server V6.1
- WebSphere MQ Security heats up
- OWASP Web site discussing the risks of cookie stealing and the HTTP Only flag
- www.keysbotzum.com
- IBM developerWorks WebSphere
- Download WebSphere Application Server V7 trial version
- Download WebSphere Application Server for Developers no-charge offering
About the author

Keys Botzum

**Keys Botzum** is a Senior Technical Staff Member with IBM Software Services for WebSphere. Mr. Botzum has over 10 years of experience in large scale distributed system design and additionally specializes in security. Mr. Botzum has worked with a variety of distributed technologies, including Sun RPC, DCE, CORBA, AFS, and DFS. Recently, he has been focusing on J2EE and related technologies. He holds a Masters degree in Computer Science from Stanford University and a B.S. in Applied Mathematics/Computer Science from Carnegie Mellon University. Mr. Botzum has published numerous papers on WebSphere and WebSphere security. Additional articles and presentations by Keys Botzum can be found at [http://www.keysbotzum.com](http://www.keysbotzum.com), as well as on IBM developerWorks WebSphere. He is also co-author of IBM WebSphere: Deployment and Advanced Configuration.

© Copyright IBM Corporation 2010
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)