

WebSphere Application Server (WAS) 5.0 Security – Hardening

Keys Botzum, Senior Consulting IT Specialist

keys@us.ibm.com

<http://www.keysbotzum.com>

<http://w3.pittsburgh.ibm.com/~keys/internal> (IBM Internal)

IBM Software Services for WebSphere

swsvcs@us.ibm.com

<http://www.ibm.com/WebSphere/developer/services>

July 2003


Last update: July 23, 2003

WHY HAVE SECURITY?

A secure infrastructure protects your system from unwanted intrusions. WAS is one key part of that infrastructure. We are going to discuss how to secure WAS.

WAS isn't the only infrastructure component you need to secure. Identify and document all of the threats you wish to protect yourself from. Many are internal.

Agenda

- 
- Introduction
 - Areas of Security Concern
 - WAS Infrastructure
 - Protocol overview
 - Firewalls
 - SSL review
 - Network Link Security
 - WAS Infrastructure Access Control
 - Machine/Operating System Protection
 - Application Security
 - Performance
 - Operational Issues
 - Wrap-up

Prerequisite Knowledge and Scope

- I assume you know
 - WAS 5.0 architecture and administration
 - Some WebSphere Application Server security administration experience will help understand what is discussed
 - Some understanding of networking concepts (what TCP/IP sockets are, for example)
 - Some understanding of PKI certificates and SSL
 - Some experience administering operating systems
- Scope
 - WAS 5.0 Distributed (Unix and Windows)
 - WAS 5.0 on other platforms is similar, but not covered here
 - WAS Enterprise is not specifically covered
 - Web Services specific issues are not covered

Change is the Only Constant

This presentation reflects

- My current opinions regarding WAS security
- The product itself continues to evolve (even in PTFs)
 - Presentation is based on 5.0.1 w/ some 5.0.2 speculation
- This will be revised as we learn more
- Your thoughts and ideas are welcome

Intrusions

- People and systems with IP connectivity to your network
 - Outsiders on the Internet
 - Insiders on your Intranet
 - In many ways more dangerous as they have knowledge, access, and possibly a grudge
 - Several sources state that the majority attacks are internal
- Attack on multiple levels
 - Users that try to get around J2EE application/admin restrictions
 - Subvert network level protocols by altering traffic, or just looking at traffic with confidential information
 - Leverage machine access to see and modify what they shouldn't
 - Legitimate applications that try to get around J2EE application restrictions
 - I will refer to these throughout my presentation to help you determine risk
- WAS provides a robust infrastructure for addressing most of these challenges.... With some assembly required.

Agenda

- Introduction
- Areas of Security Concern



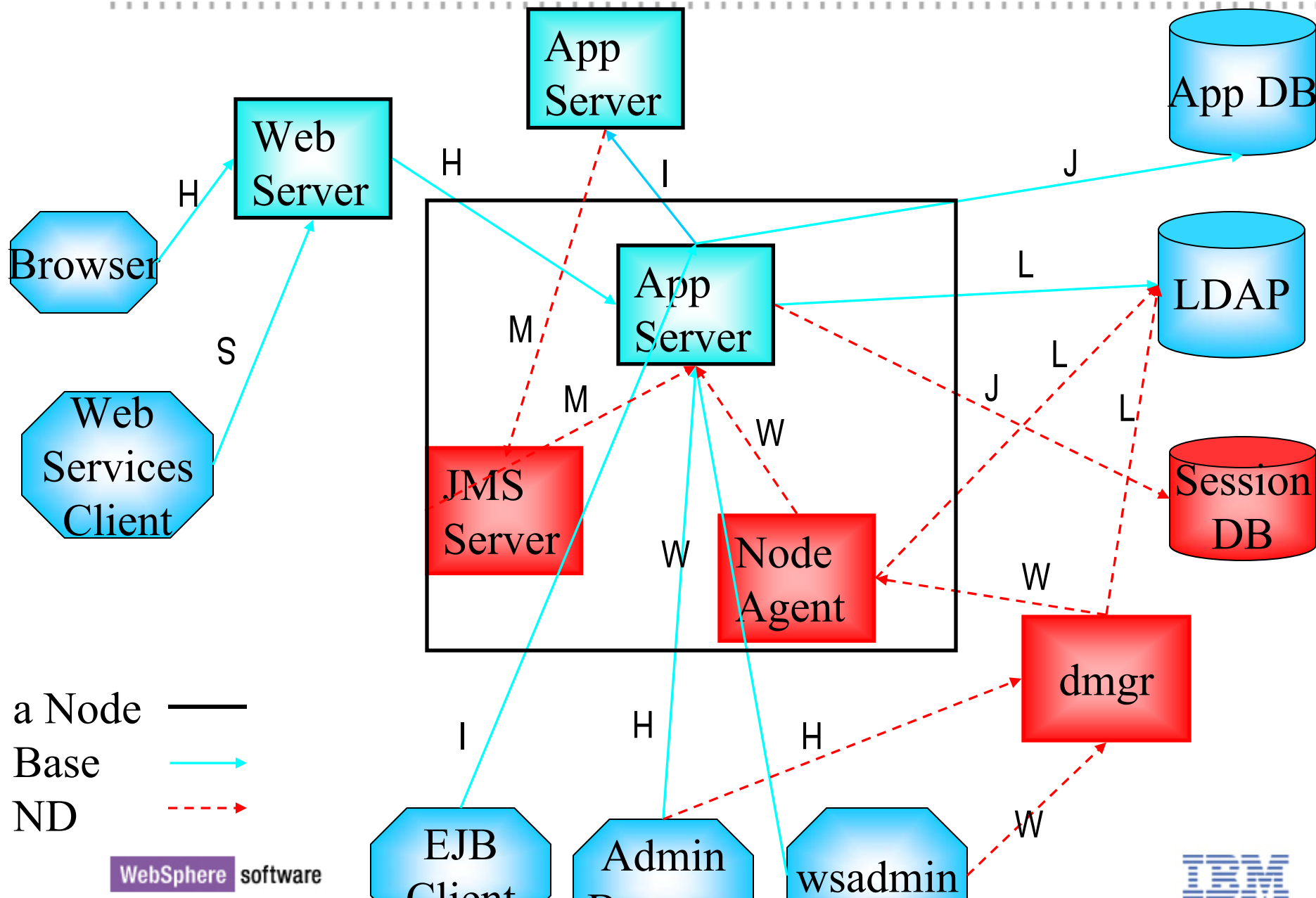
- WAS Infrastructure
 - Protocol overview
 - Firewalls
 - SSL review
 - Network Link Security
 - WAS Infrastructure Access Control
- Machine/Operating System Protection
- Application Security

- Performance
- Operational Issues
- Wrap-up

Major Security Changes in 5.0

- All app servers contain an integrated security server
 - Access registry (typically LDAP) directly
 - Critical part of security trust domain
 - The admin server is gone
- Java 2 Security is integral (and critical) to the product
- All app servers read configuration information from local XML configuration files

Basic Topology



Protocols Used

- H = HTTP traffic
 - Usage: browser to web server, web server to app server, and admin web client
 - Firewall friendly
- W = WAS internal communication
 - Usage: admin clients and WAS internal server admin traffic
 - Protocol:
 - RMI/IIOP or SOAP/HTTP. Client protocol is configurable.
 - File transfer service (dmgr to node agent) uses HTTP(S)
 - DRS (memory to memory replication) uses private protocol
 - Firewall friendly if using SOAP/HTTP
- I = RMI/IIOP communication
 - Usage: EJB clients (standalone and web container)
 - Firewall hostile
- M = MQ protocol
 - Usage: MQ clients (true clients and application servers)
 - Protocol: Proprietary
 - Firewall feasible (there are a number of ports to consider). Refer to MQ support pac MA86.

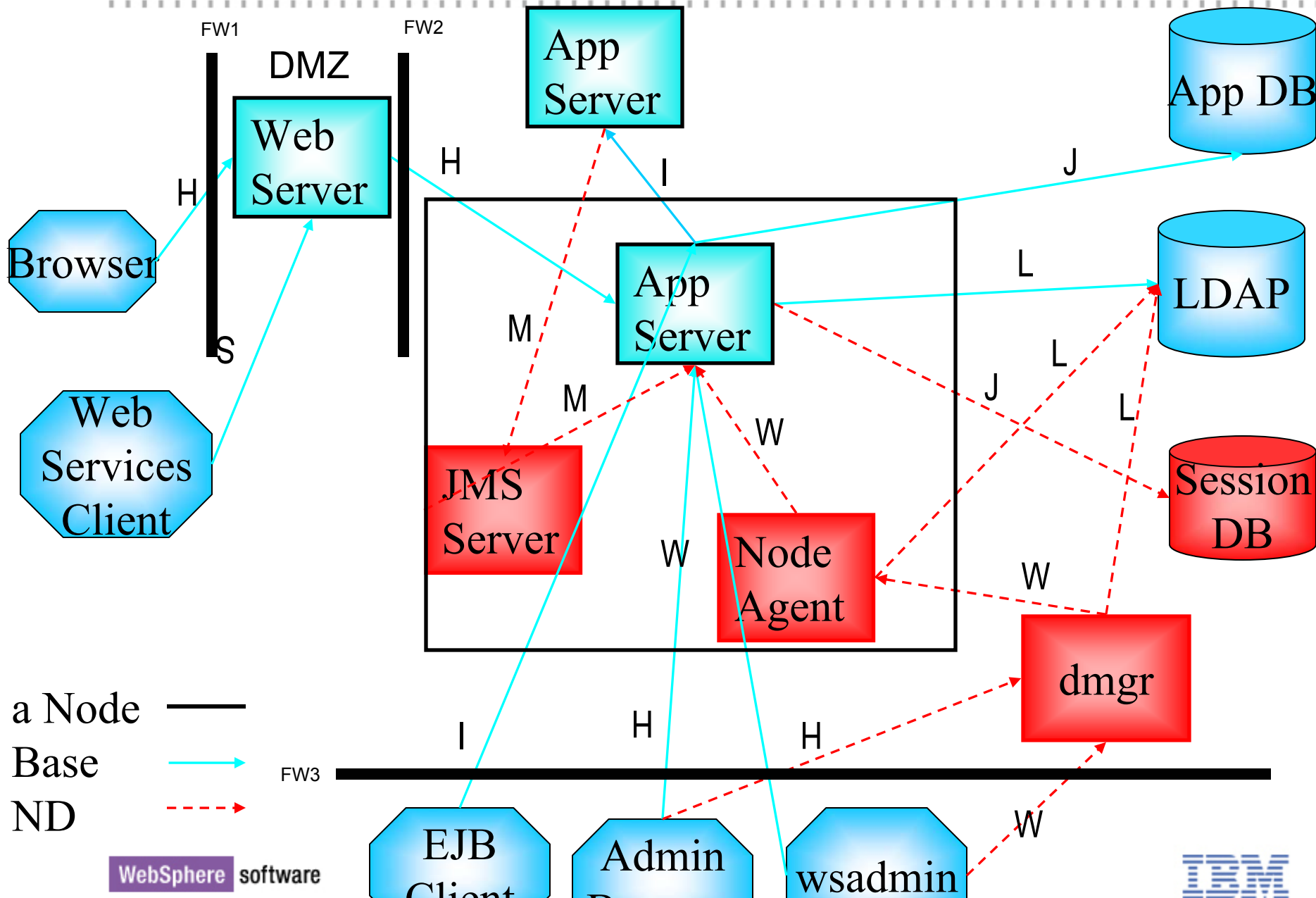
Protocols Used

- L = LDAP communication
 - Usage: WAS verification of user information in registry
 - Protocol: TCP stream formatted as defined in LDAP RFC
 - Firewall friendly
- J = JDBC database communication via vendor JDBC drivers
 - Usage: application JDBC access and WAS session DB access
 - Protocol: Network protocol is proprietary to each DB.
 - Firewall aspects depend on database (generally firewall friendly)
- S = SOAP
 - Usage: SOAP clients
 - Protocol: generally SOAP/HTTP
 - Firewall friendly

Agenda

- Introduction
- Areas of Security Concern
 - WAS Infrastructure
 - Protocol overview
 - ➔ – Firewalls
 - SSL review
 - Network Link Security
 - WAS Infrastructure Access Control
 - Machine/Operating System Protection
 - Application Security
- Performance
- Operational Issues
- Wrap-up

The "Standard" Configuration With Firewalls



Firewall comments

- Notice the DMZ
 - Nothing in the DMZ but the web server. Hopefully on a stripped web server box.
 - Hostile place with hardened OS and few tools. WAS doesn't belong here.
- The app server (WAS & DB) goes behind second firewall on a "production" subnet protected by a third firewall
- Both DMZ firewalls configured to allow only HTTP(S) traffic on specific ports
- Notice the third firewall between production subnet(s) and the corporate intranet
 - Web admin console will go through firewall if correct ports opened
 - wsadmin may not work if using IIOP. SOAP should go through okay.
 - Can configure firewalls to allow IIOP, just tricky

Additional Firewalls

- Every component (WAS, DB, Web server, LDAP) could get its own subnet (vlans - easily achieved with a switch)
 - Firewall between every subnet ("virtual" firewalls)
 - Consider DNS, internal client connection and manageability issues
 - Complexity increases rapidly with diminishing value
 - Any firewalls that separate EJB clients from the WAS services need to be IIOP friendly
 - I prefer to focus firewalls on traffic from “outside” of cell rather than trying to block WAS internal traffic

Agenda

- Introduction
- Areas of Security Concern
 - WAS Infrastructure
 - Protocol overview
 - Firewalls
 - ➔ – SSL review
 - Network Link Security
 - WAS Infrastructure Access Control
 - Machine/Operating System Protection
 - Application Security
- Performance
- Operational Issues
- Wrap-up

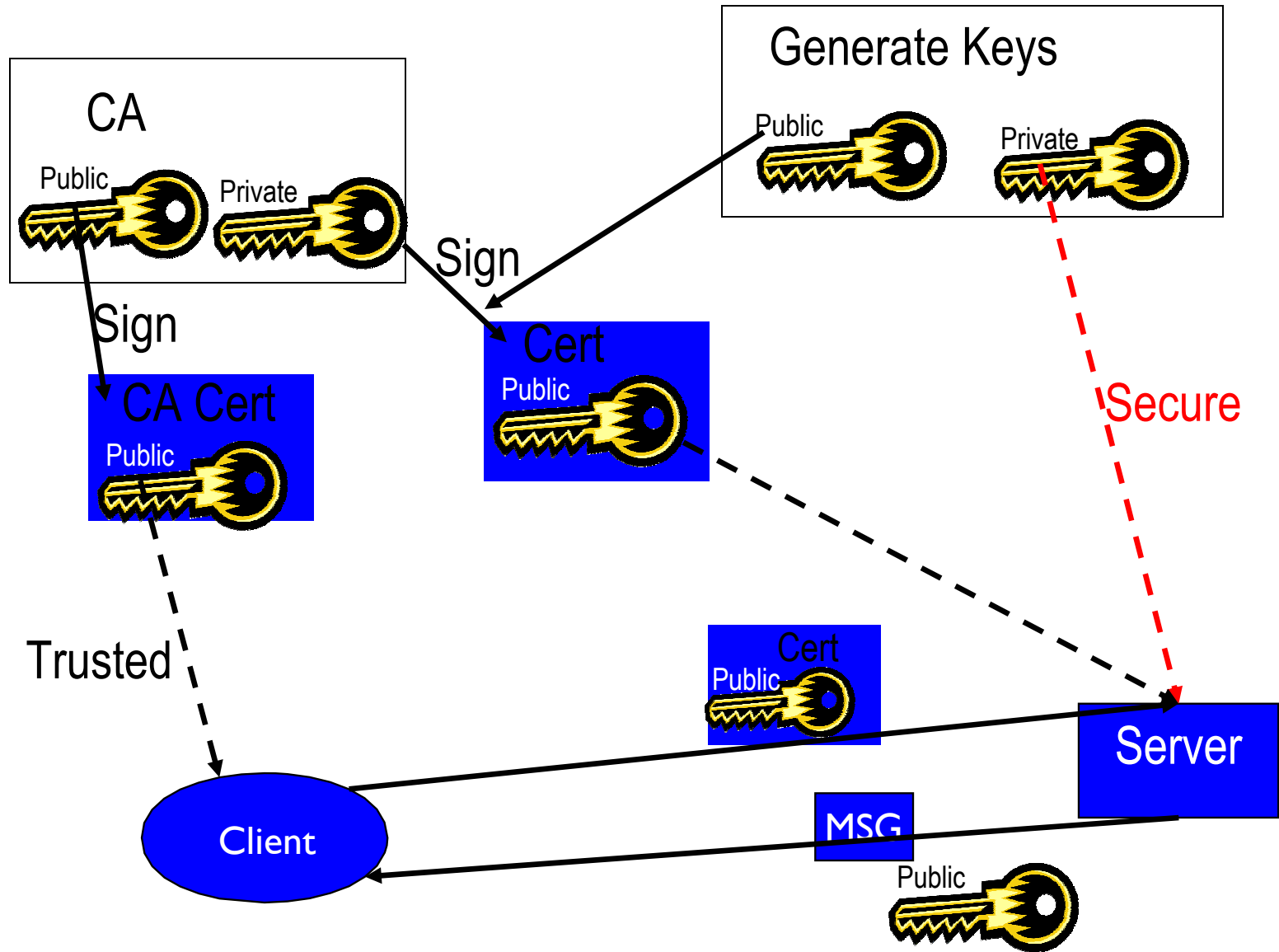
SSL Nuts and Bolts

- In order to establish trust, each side of SSL connection must possess appropriate keys in a key file
- Private Key
 - My private information that I use to prove identity (private half of public/private key pair)
- Certificate
 - A public key signed by someone using a signer certificate
- Signer certificate – two kinds
 - A Certificate Authority's (CA) certificate (e.g., Verisign), or
 - A self signed certificate - in this case, the certificate subject and signer are the same entity. Note: these are **not** weaker. They are just harder to manage. Key distribution is more difficult.

SSL Nuts and Bolts (*continued*)

- To support ordinary server authentication (client validates server's identity)
 - Server must have its own private key & certificate
 - Client must have signer certificate for server certificates
- To support client authentication (server validates client's identity)
 - Client must have its own private key and certificate
 - Server must have signer certificate for client certificates

Certificate Usage (Client Identifies Server)



SSL – how do I manage these keys?

- Key files contain certificates and private keys
- Key file versus Trust file
 - A trust file is a kind of key file
 - Trust file is intended to contain signer certificates
 - Key file contains private keys and personal certificates
- Place the keys as described in previous slide in appropriate key files
- IBM provides ikeyman tool for managing key files
 - multiple versions out there, so use the right one
 - One for WAS key files known as JKS files (Java Key Store)
 - One for IBM HTTP and WAS plugin key files known as KDB files.

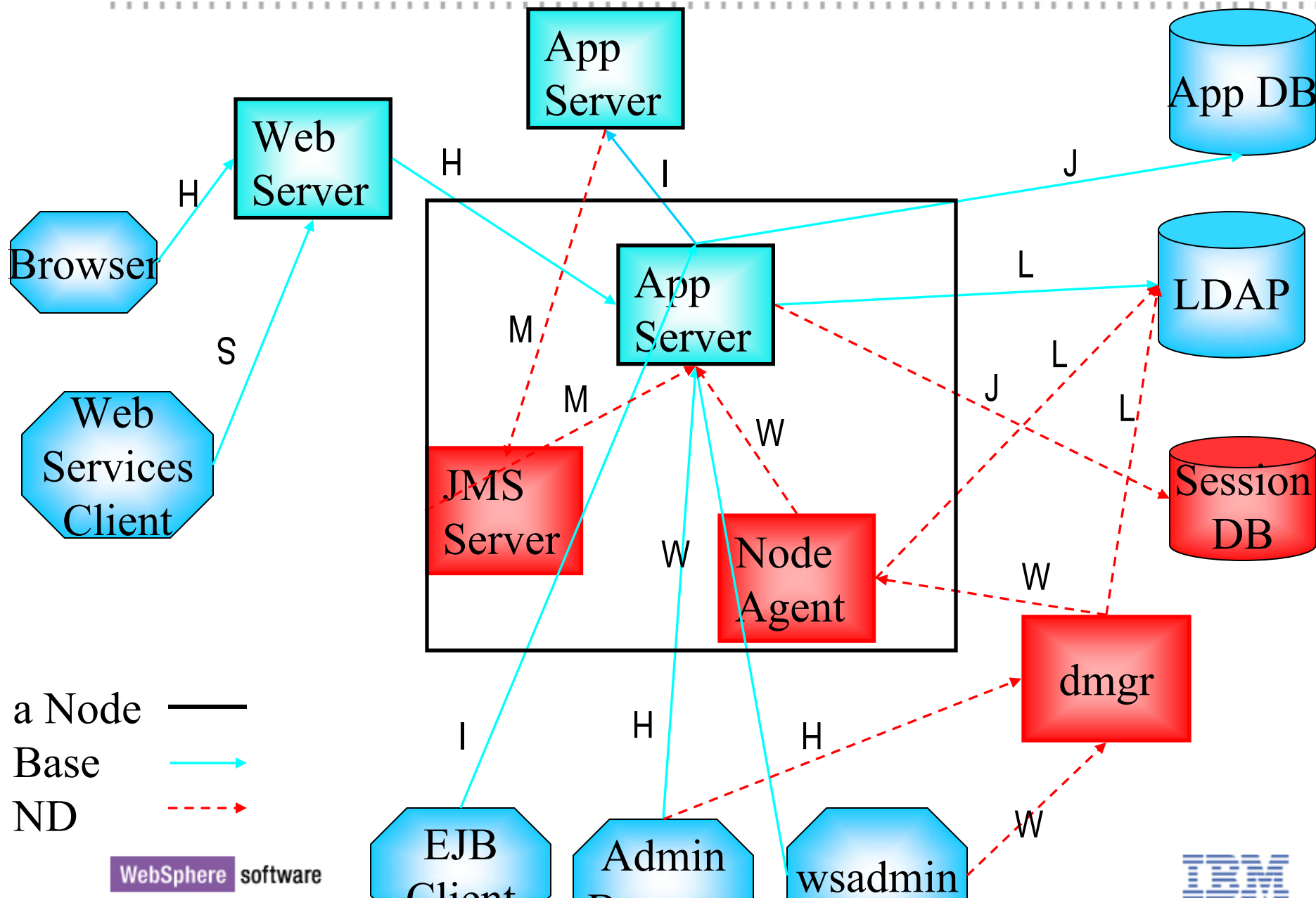
Agenda

- Introduction
- Areas of Security Concern
 - WAS Infrastructure
 - Protocol overview
 - Firewalls
 - SSL review
 - ➔ – Network Link Security
 - WAS Infrastructure Access Control
 - Machine/Operating System Protection
 - Application Security
- Performance
- Operational Issues
- Wrap-up

Beyond Firewalls: Securing the Links

- Firewalls are a valuable component in an overall security plan, but they are **not** sufficient
- Now that we have firewalls in place
 - Let's secure all of the network links within WAS
 - This list is in priority order

Basic Topology



The Links

- So many links, so many protocols
- Just be systematic and patient

(H) Browser to Web Server SSL

- Configure your web server to support HTTPS as defined in the web server documentation
 - Popular web browsers ship with 50 odd CA certificates. You'll want to support them. So, purchase a certificate from a well-known CA.
- If needed, add a virtual host alias for the HTTPS port
- WAS can enforce that HTTPS is used by specifying a data constraint in web.xml
- Applications can get connection information via servlet interface ...
`X509Certificate certChain [] = (X509Certificate [])req.getAttribute("javax.net.ssl.peer_certificates");`
- Always use SSL for any confidential information
- Hardware acceleration is available but support varies based on Hardware, OS and Web server

(W) & (I) WAS Internal and IIOP Communication

- Enable WAS security
 - WAS internal communication (SOAP and RMI/IIOP) runs over SSL. This includes admin clients, EJB clients, and WAS internal communication. Even the WAS web admin console switches to HTTPS.
 - But, DRS still insecure. Must force to use DES encryption via configuration of Internal Replication Domain
- Don't use SWAM Authentication
 - Uses weak mechanism that relies on HTTP Session for tracking user identity
 - LTPA is stronger and is supported in all editions (Express PTF 1) and with all registries
 - If you must use SWAM, always use HTTPS to protect session id. This helps, but is still weaker than LTPA.
- WAS internal security is independent of application security
 - Enabling makes WAS infrastructure secure. Applications must leverage this explicitly to secure themselves.

(W) & (I) The Default Key Files

- All WAS internal SSL traffic (SOAP and IIOP) is based around the keys in the key files used by WAS. These are controlled by the SSL configurations.
- By default, there is one SSL configuration shared by all of WAS and it is using a DummyKeyFile that shipped with product
 - Every user of WAS gets shipped the same private key. Not very private.
- As appropriate, you need to create new SSL configurations with new key and trust files
- In most cases, sufficient to update the existing default SSL configuration that is shared by the WAS components
- Create new key DB with new private key & certificate
 - procedure requires using WAS ikeyman to request certificate from CA
 - update SSL config (Security Center/Default SSL Config) to use the new key database

(W) & (I) Updating All the Key Files

- Don't forget to update the related key files so everything will work
- All nodes and the deployment manager have these files – each has its own <root>/etc directory, update them everywhere
- The web server plugin needs to recognize the new server certificate. Update plugin-key.kdb. Make sure you update the one actually used by the web server (check plugin-cfg.xml).
- WAS IIOP/SOAP clients
 - WAS admin clients are just IIOP/SOAP clients
 - Update the client trust file with the signing cert
 - Need to update the product *.client.props file to point to updated trust file
 - New Key file names and passwords
 - On *all* nodes and dmgr
 - Don't forget remote clients

(W) & (I) Updating All the Key Files – CA Issues

- How you generate the new private key and certificate affects how you must update in client trust files
 - Well known CA
 - CA cert it is probably already in the standard WAS client trust file
 - No update needed for client trust files (or plugin-key.kdb)
 - Must buy cert from CA and renew yearly
 - Your own CA or a self-signed certificate
 - Create a trust file for the client using ikeyman
 - Add the WAS or CA certificate as a signer
 - Create/update a sas.client.props or soap.client.props file for all of the clients
 - Saves money, but now you must distribute the keys to your clients
- Generally self-signed certificates are best choice unless you have many Java clients

(L) Security Server to LDAP server SSL

- Each WAS server (app, dmgr, and nodemgr) contains an embedded security server
- The security server communicates to LDAP server
 - Queries lots of information which may be sensitive
 - Sends passwords to LDAP for verification
- To protect this link
 - Specify “use SSL” LDAP config page in Security->User Registries->LDAP
 - Can use the DefaultSSLConfig or create new one
 - WAS needs to possess certificate for CA/signer that issued LDAP certificate
 - Using ikeyman, add the LDAP signer certificate to the trust file used by the SSL configuration that LDAP will be using

(H) Web Server (via plugin) to App Server SSL

- Plugin transmits information from the web server to the WAS application server
 - this information may be security sensitive - in particular, authentication information (passwords or user identity from certificates).
- Use HTTPS/SSL to protect traffic
 - SSL and non-SSL is supported by default by the web container
 - No action is required after configuring web server to use HTTPS and updating plugin key file to contain correct signing certificate (the one you created earlier)
 - Not using client authentication – app server is not authenticating web server
- If you want to limit app server web container access to only trusted web servers, you need to do more:
 - Limit the app server to HTTPS transport
 - Update the web server plugin configuration
 - See next 2 slides

(H) Web Server (via plugin) to App Server SSL

- Create three key files - two for the WAS web container and one for web server plugin
 - For web container use WAS ikeyman to create 2 JKS key files: WASWebTrust and WASWebKey
 - For WAS plugin on the web server use GSK5 ikeyman to create KDB key file
 - For all key files, remove all default CA/signer certs
 - Now, can't trust **any** certificate
 - For KDB file and WASWebKey create self-signed cert & private key
 - Each side has its own private key and corresponding self-signed cert
 - Import self-signed cert for plugin into WASWebTrust and import self-signed cert for web trust into plugin KDB file
 - Now, each has ability to verify the other's certificate **and nothing else**
 - If authentication used, no other party can possibly communicate since they don't have the needed keys

(H) Web Server (via plugin) to App Server SSL

- Configure WAS web container to use new JKS files
 - Create a new SSL configuration
 - Select client authentication (will authenticate the web server's cert)
 - Specify the web trust and key files you just created
 - Update the existing SSL transport for the Web Container that serves application requests to use the new SSL configuration.
 - Remove existing HTTP Transport from the Web Container to force use of authenticated SSL
- Replace plugin key file with the one you just created
- Now, only trusted web servers that possess corresponding private key can talk to WAS
 - FYI, typically all of the web servers will share the same key ring
 - Even an ordinary web browser will be unable to connect to the app server. Will get SSL handshake failure.

(H) Web Server (via plugin) to App Server SSL

- Performance Note

- The plugin is smart enough to reuse SSL sessions
- If container configured with HTTP and HTTPS transports
 - if request came in on HTTPS plugin uses HTTPS
 - if request didn't use HTTPS plugin doesn't use it either

(J) WAS Session Manager and App to DB

- Many databases do not provide encrypted links from JDBC clients to the database, but this is changing for the better
 - Oracle Advanced Security supports encryption
 - The DataDirect Sequelink driver supports encryption (w/ SQL Server)
 - DB2 V8.x may add network encryption (x means some fixpack)
- If you can't swing DB encryption, protect this link as best as you can
 - Inside your intranet. NEVER expose DB to internet.
 - Use firewalls to protect production database from non-production networks
 - Use clever network routing and/or firewalls to limit access to DB to "trusted" client machines
 - Use VPN technology (such as IPSEC) to encrypt links between DB and WAS

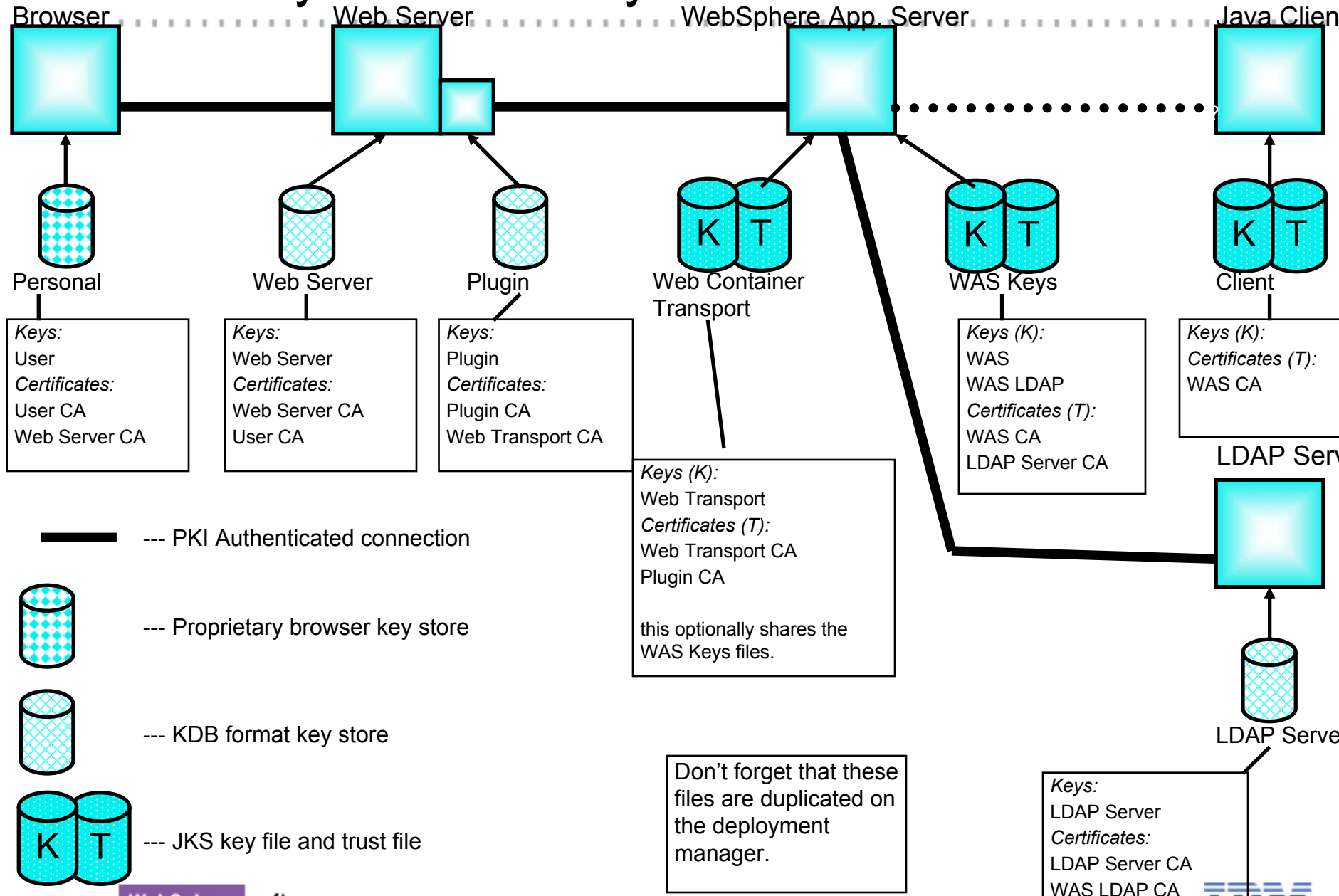
(S) SOAP Client Access

- Of course, SSL/HTTPS is leveraged from the client to the web server
- Need to authenticate SOAP clients by
 - Using HTTP Basic auth
 - Very preliminary WS-SECURITY
- Don't forget to authorize
 - Best bet: map your SOAP requests to EJBs. Then, standard J2EE authorization rules apply.

(M) – App Servers to MQ

- Embedded Messaging (based on MQ) does not support SSL connections from clients
- Full MQ supports SSL between MQ managers and from clients when using MQ client mode
- Recommendation: Utilize the full MQ for production environments with network hops, but beware of the need to develop security exits for authentication and authorization (more on this later)

SSL Key File Summary



Agenda

- Introduction
- Areas of Security Concern
 - WAS Infrastructure
 - Protocol overview
 - Firewalls
 - SSL review
 - Network Link Security
 - ➔ – WAS Infrastructure Access Control
 - Machine/Operating System Protection
 - Application Security
- Performance
- Operational Issues
- Wrap-up

WAS Infrastructure Security

- Enable Global Security
 - All admin access now requires authentication and is authorized and uses secure links
- Enable Java 2 Security
 - Prevents application code from doing dangerous things
 - Critical for protecting applications from each other
 - Critical for protecting WAS internals from applications

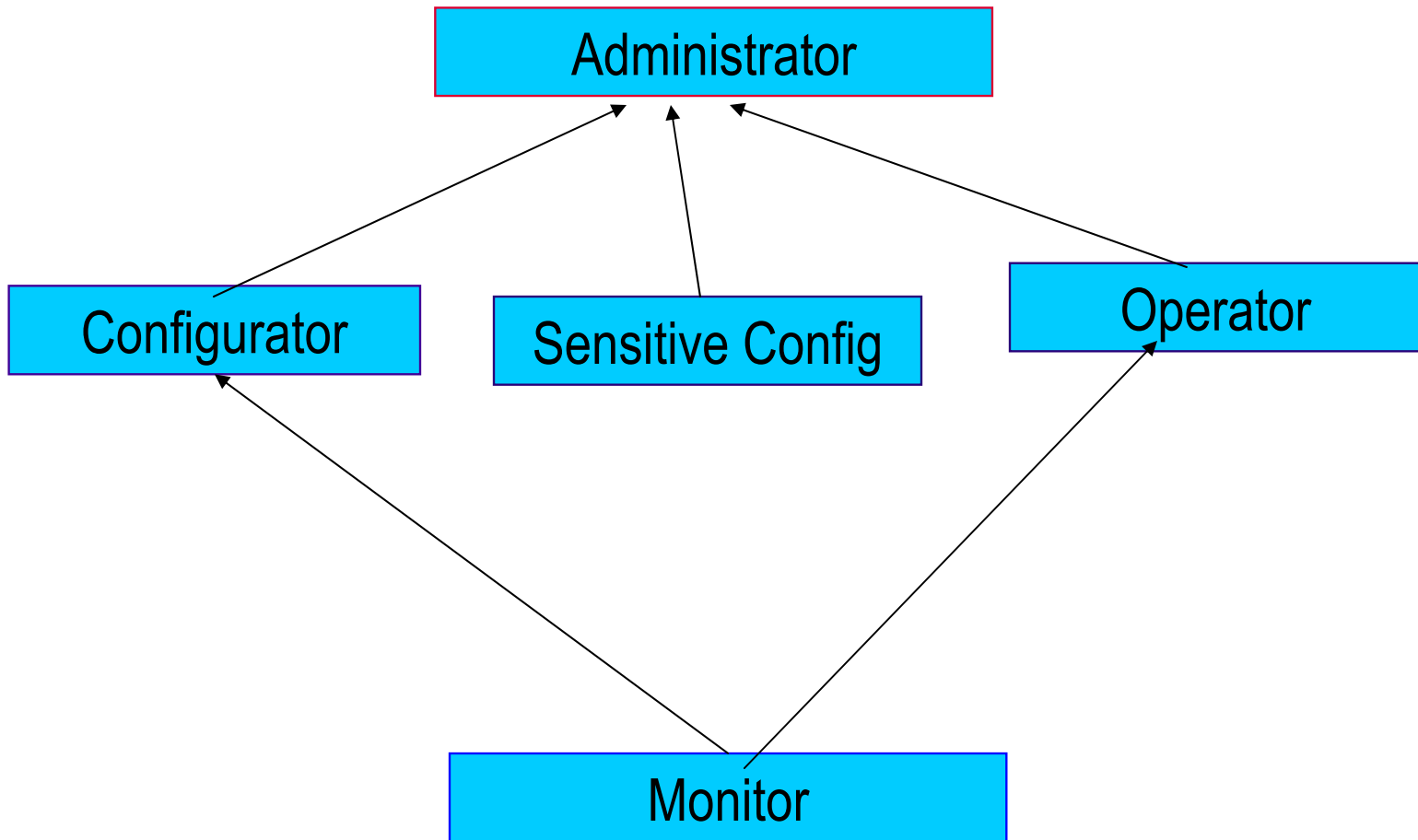
WAS Administrators

- Need a “root/Administrator” **WAS user id** in the WAS user registry (Server User ID)
 - Does not need administrative privileges in the registry. Certainly doesn't need to be root.
 - Used by WAS servers when authenticating internally
 - Should not be used by WAS administrators when authenticating
- Create a **WAS user id** account for each person that will administer the WAS domain
 - Create in your registry, then
 - Using the admin console System Administration->Console Users/Groups, specify additional administrators. These are users/groups from the underlying WAS registry.

WAS Administrators

- WAS admin authority has 4 levels:
 - **Monitor** – look, but not touch
 - **Operator** – start/stop, but not change
 - **Configurator** – configure, but not start/stop. Can't edit some “sensitive” data.
 - **Administrator** - everything
- Now, you can limit administrative access based on need. This is valuable, for example:
 - During development you can give all developers the ability to start/stop app servers, but not mess with the repository
 - During production, you can give people permissions based on job role
- Access is cell wide. Split into multiple WAS cells to restrict access.

WAS Admin Roles



Name Space (a.k.a. CORBA Naming) is Open

- Too many have too much access by default
- Tighten this up
 - By default the Naming permissions are scary
 - Everyone – read
 - All Authenticated – read, write, create, delete
 - ➔ Anyone in your registry can pretty much destroy a cell
 - This is much safer
 - Everyone – nothing
 - Unauthenticated threads (e.g, anonymous servlets) can't read global JNDI. This might break apps that don't use security.
 - MDBs will need RunAs in order to access JNDI
 - Local refs (e.g., java:comp/env) still work fine
 - All Authenticated - read
 - WAS ensures that its own components always have read/write access so core function will continue to work
- Permission are set in the admin console via Environment->Naming

(H) Web Admin Console

- Don't forget that the web admin console uses HTTP(S) to connect to the admin web application. As with any web client, it connects to the web container.
- The admin console connects on a special port for this purpose in the web container (9043 by default)
 - As described earlier, you need to update the default SSL keys, but there is more to worry about
 - The web admin app uses the server key from the server key file (aka the "default" keyring)
 - Get into the usual certificate issues...next slide

Admin Web Browser Certificate Warnings

Security Alert [X]

 Information you exchange with this site cannot be viewed or changed by others. However, there is a problem with the site's security certificate.

-  The security certificate was issued by a company you have not chosen to trust. View the certificate to determine whether you want to trust the certifying authority.
-  The security certificate date is valid.
-  The name on the security certificate is invalid or does not match the name of the site

Do you want to proceed?

Admin Web Browser Certificate Warnings - Addressing

- Your web browser may issue warnings when using the web admin tool
- To address the site name not matching hostname problem
 - Obtain a certificate with the dmgr's hostname as the subject to avoid browser warnings about the name mismatch on every access
 - Since the dmgr hostname can't change this should not be a problem
- To address the certificate trust problem
 - Option 1: use a well-known CA to issue WAS' certificates
 - Expensive, must renew yearly
 - Option 2: simply accept the certificate in the browser on the first use as trusted
 - Make sure it's the right cert
 - Train your admins that if this message ever comes up again there is a problem!

Corner Cases: Weaknesses in WAS Security

- Certificate Authentication Limitations
 - Doesn't check Certificate Revocation Lists (CRLs)
 - Doesn't check that destination of request is in fact intended party
 - Would require that caller identify hostname or identity expected of server being contacted
 - Subject to man in the middle attacks
- If you compromise one server you have compromised the entire cell
 - Kerberos and DCE have very limited set of trusted servers
 - A WAS cell should not span trust boundaries. If you can't trust someone else totally, don't let them run a server in your cell.
- Does not perform secure chained delegation
 - If I call A, nothing prevents A from calling B and acting as me
 - B can't tell that call came through A

Agenda

- Introduction
- Areas of Security Concern
 - WAS Infrastructure
 - Protocol overview
 - Firewalls
 - SSL review
 - Network Link Security
 - WAS Infrastructure Access Control
 - Machine/Operating System Protection
 - Application Security
- Performance
- Operational Issues
- Wrap-up



Machine/Operating System Issues

- You've secured the network links, but now somebody has access to the WAS machine
- Maybe their access is legitimate, how do you reduce risk?
- Think about operating system level control
 - File permissions
 - Operating system users
 - Auditing/monitoring (not covered here)
 - Hardened operating systems (not covered here)

Protect Your Configuration Files

- There are numerous files in a typical WAS install that need to be protected because of what they contain
 - The configuration repository XML files in config – contain topology information and many embedded passwords (LDAP, WAS root, admin DB)
 - etc/DummyServerKeyFile.jks (you will probably change the name of the file) - a JKS keystore containing WAS' private key
 - etc/plugin-key.kdb - web server's private key
 - etc/plugin-key.sth - the password for access to the plugin-key.kdb.
 - ▶ sas.client.props or soap.client.props - Client connection config file may contain a user UID and password
 - installedApps – files for applications that have been installed. User's other than WAS shouldn't be able to modify. Might contain sensitive information.

Protect Your Configuration Files

- Access required
 - Every application server needs to read these files. Write access is needed if WAS base.
 - ND: Only nodemgr needs write access to config files. Dmgr has its own config directory that it needs read/write access to.
 - The web server (if on same machine) needs some access as well: plugin key file, log directory, and plugin-cfg.xml
 - Beware: many shared files (e.g., logs) that everybody writes to

User Identities for WAS

- Two kinds of identities to consider
 - WAS "User Registry" principals: called "**WAS users**".
 - defined in the WAS user registry which could be: LDAP, Operating System (OS), or Custom
 - Operating System (OS) accounts used for file ownership and as the identity of running processes: called "**OS users**".
 - Defined in the OS on the machine running WAS
- No relationship between the two
- Both types of identities have security implications. We've talked about WAS users, now let's discuss OS users.
- When configuring WAS you have to choose the operating system identities it should use for its processes
 - many issues to consider

Choosing OS Users - Everything as Root/Privileged User

- Default, out of the box configuration
- Can use local OS as user registry
- All WAS processes have read/write access to all WAS related files (and everything else)
- WAS administrators have implicit root authority
- Can configure so that nothing else (other than root) has any access to WAS files
- Very easy to configure










OS User Accounts – Everything as Normal User

- All app servers must run as same **OS user** as the Node Manager (e.g. wasuser)
- All WAS processes have equal read/write access to WAS directories
- The **OS user** (wasuser) needs these file system permissions
 - Read privileges on all WAS files
 - Write privileges on (all) application content (installedApps), config, etc.
 - Write privileges on log, temp, config, tranlog, and properties directories
- WAS administrators don't have root access
- Don't forget about Embedded Messaging. You need to add the wasrun user to the both the **mqm** and **mqbrkrs** groups.
- Fairly easy to configure
 - Simple chmod/chown of the WAS files and you are on your way

OS User Accounts – Node Manager as root (Unix only)

- Assign **OS user** accounts for app servers to limit access
 - “wasrun” - is the **OS user** that the app server “runs as”
 - member of *wasgroup*
 - Can limit access to files not owned by that application (doesn’t address shared application servers)
 - “wasgroup”
 - Write privileges on temp, log, tranlog directories (and maybe more)
 - Read privileges on all WAS files
- Node manager runs as the root (or root-like) **OS user**
 - Same permissions as above, plus has write privileges on the WAS config, installedApps, and properties directories.
 - WAS administrators have implicit root authority
- Very difficult to configure

Options Summary

	All as root user	Node as root	All as non-root
WAS admins have implicit root authority			
Some WAS admin tasks may require root access			
Apps in different servers theoretically have access to WAS' and other apps files and resources*			
Can't use Operating System Registry			
May be complex file ownership/permission issues			

*Java 2 Security can address this, even for shared application servers

Agenda

- Introduction
- Areas of Security Concern
 - WAS Infrastructure
 - Machine/Operating System Protection
 - Application Security
 - Resource Security
 - J2EE Security is good
 - Various Dangers
- Performance
- Operational Issues
- Wrap-up



Application Security

- Those pesky applications and users have legitimate access. The challenge is to limit it appropriately.

J2C Resource Security

- Most J2EE resources are J2C resources: JDBC datasources, JMS, etc.
- Resources
 - Are visible via JNDI
 - Have associated authentication aliases
 - Can be bound into the local namespace for an application via resource refs
 - Provide ability to authenticate to underlying resources. E.g., can provide userid & password to a database.
- The JNDI access is a potential security hole
 - If a resource can be accessed by an attacker that does not specify authentication data this is a potential security hole
 - Anyone (within WAS cell or *external* to WAS cell) with JNDI access may be able to access your resources depending on your configuration
- Think carefully about resource security – this is a common source of serious security holes

J2C Resource Security – Authentication Modes

- Mode defined several ways
 - On a resource-ref definition
 - On CMP factory connection binding
 - Implicitly when global JNDI name is looked up
- Two modes: application and container
- Application (called `per_connection_factory` for CMP beans)
 - Authentication provided via
 - J2C **component**-managed authentication alias
 - Explicit `userid/password` on `getConnection()` calls
 - This is the mode if the global JNDI name is looked up directly in code
- Container
 - Authentication provided via **container**-managed authentication alias
 - Can also write custom J2C security module, but we are not discussing
 - `getConnection()` with `userid` & `password` is ignored
 - This can only be chosen via a local resource-ref binding or CMP binding

J2C Resource Security – Resource Definition

- J2C Resource Authentication Aliases
 - **Component**-managed
 - Resource accessible via JNDI access from anywhere (with IIOP connectivity)
 - **Container**-managed
 - Allows access to resource without password, but only when enabled via a local-ref and binding created by the administrator
 - Administrator is in control of resource access
- The net, to be highly secure
 - Define J2C resources with container managed aliases, define local resource references, and administratively bind these together during deployment, or
 - Define J2C resources with *no* alias and then use application authentication by specifying userid and password on getConnection()

Reality Check: Datasource and JMS Usage

- DataSources
 - Define only container alias and require that applications use resource-refs
 - Applications have to use resource-refs or provide userid/password
 - **Warning:** JTA/XA two-phase commit requires that a J2C alias be specified on the datasource. You **must** define container aliases when using JTA/XA.
- JMS
 - Similar approach, but there is a catch
 - WAS 5.0.0 and 5.0.1 MDBs require component aliases

Security Fixes/Enhancements

- Get the enhancements. Get them now. They close several security holes.
- MDBs support use of container aliases (eliminating component exposure)
 - PTF 2, or
 - WAS_Messaging_06-16-2003_5.0.1_cumulative_Fix which requires WAS_Naming_05-20-2003_5.0.1_cumulative_Fix
 - Note: there is also a Container Messaging (Enterprise) fix: PQ75257_Fix (prereqs MDB fix)
- Prevent external clients from accessing J2C resources with component aliases
 - APAR PQ76478 or PTF 3
 - With this fix, component aliases are not subject to external attacks, but can still be attacked from applications within the cell
 - You don't need this fix if you don't use component aliases
- WAS 4.0 datasources (in 4.0 or 5.0)
 - If you define a default userid/password, you are subject to external attack
 - Beware that 2-phase commit recovery requires a default userid/password. You can protect by specifying new datasource property secureXACredential=true
 - Makes default userid/password available only to WAS internals
 - For WAS 4.0, you'll need connection manager fix dated 05-30-2003 or later

Database Authentication and Authorization

- Issue: Conflicts between Database and Application Security Models
 - WAS knows the user's identity. The DB does not.
 - All DB connections use a connection pool with a common identity
 - Database audit and database authorization functions are useless (this is bad)
 - Non-options
 - separate connection for each user (won't scale)
 - new connection for each request (won't scale)

Database Authentication and Authorization

- Options:
 - Get over it, and (best choice)
 - write application audit functions in app layer
 - write application authorization functions in app layer
 - Rely on DB specific and proprietary SQL for changing connection identity
 - Some customers have done this with WAS by writing some tricky code
 - Not all databases support this
 - Probably not going to work with CMP beans

JMS/MQ Authentication

- Embedded
 - Userid/password that is sent to JMS Server is verified against WAS registry
- Full MQ in bindings mode
 - No built in fine grained authentication, relies on process level authentication
 - Userid/password info specified on JMS resource is ignored
 - All that matters is that the process id that WAS runs as has access to MQ – should be in the appropriate mqm group, etc.
- Full MQ in client/server mode
 - Relies on process level authentication by default
 - Userid/password info specified on JMS resource is ignored by default
 - You can develop custom security exits for client authentication. These exits can access the userid/password information on the connection.
- Embedded JMS superficially appears to have many security advantages, but it isn't:
 - Embedded doesn't support SSL
 - Lacks many other production quality features unrelated to security

JMS/MQ Authorization

- J2EE security
 - Not relevant to queue access
 - E.g., MDB's access to queue is based on queue authentication, not any J2EE identity. Ditto for direct JMS queue access.
 - MDB J2EE identity is anonymous by default
 - Can use RunAs on an MDB to affect the J2EE resources (JNDI, EJBs, J2C, etc) that the MDB accesses
 - Identity of message enqueuer is irrelevant
- Embedded
 - Authorization based on integral-JMS-authorizations.xml
- Full MQ
 - Can use setmqaut tool to set queue level authorization

Beware of Native Authentication

- Many databases (e.g., DB2) and MQ support native authentication based on process id
 - If the WAS operating system process has resource access, you have given it to all applications in the cell. They can use direct JDBC/JMS access.
- Database Solution
 - WAS process id doesn't need database access
 - Running database remotely also addresses (assuming no cross operating system trust)
- Messaging Solution
 - Embedded – no problem, requires userid & password on connection
 - Full MQ
 - Uses process level authentication unless you custom develop security exits and enable client/server mode
 - In a shared infrastructure, this is a potential problem – need those security exits

Agenda

- Introduction
- Areas of Security Concern
 - WAS Infrastructure
 - Machine/Operating System Protection
 - Application Security
 - Resource Security
 - J2EE Security is good
 - Various Dangers
- Performance
- Operational Issues
- Wrap-up



User Access: J2EE Security

- Applications or components that call each other using J2EE APIs can (and should) leverage standard J2EE security authorization
- To be secure, you **must** have strong authentication and authorization! Think!!
- “Users” can also be other remote applications. They need authentication and authorization too.
- Refer to J2EE documentation, WAS Redbooks, and my presentations for more

Think About and Leverage J2EE Security

- I'm amazed by how often I see
 - Applications with non-existent or laughable "authentication." E.g., remote servers to simply pass in a userid without proof of identity. Use WAS authentication. If you wrote your own, there is a good chance it isn't really secure.
 - Applications that authenticate users but then don't perform any meaningful authorization
 - Applications that authenticate and authorize access, but leave open back doors. E.g., authenticate and authorize servlet access but leave EJBs wide open!
 - Incredibly, I even see applications that have strong authorization, but weak or non-existent authentication
 - It's not easy to write a strong security token and manage authentication

Application Restrictions: Java 2 security

- OS permissions provide limited protection
 - Can limit file system access based on operating system id, but
 - Multiple applications might share an application server
 - All WAS application servers might share one id if WAS node manager isn't running as root
 - Doesn't limit access to in-process resources or APIs
- Rely on Java 2 security
 - Default permissions prevent dangerous access to file system and Java resources. E.g., servlets have only file system access to their own WAR.
 - Access to WAS APIs (such as security APIs), threads, sockets, file system, etc is greatly restricted. Excellent for a shared environment.
 - We have prototyped application code that can break WAS security when Java 2 security is not enabled.
 - Access to other applications APIs is prevented
 - **Net:** you need Java 2 security in a shared environment and you must enforce it carefully

Effectively Using Java 2 Security

- Some applications need additional access
 - If so, **carefully** give them just the access they need by editing was.policy in EAR. Limit access to just the resources that are appropriate.
 - When deploying applications that need additional Java 2 security permissions, take the time to read the warnings from WAS
 - Generally, you want Java 2 security
- Beware of native code
 - It is not checked by Java 2 security
 - It can do anything that isn't prevented by the operating system

Agenda

- Introduction
- Areas of Security Concern
 - WAS Infrastructure
 - Machine/Operating System Protection
 - Application Security
 - Resource Security
 - J2EE Security is good
 - Various Dangers
- Performance
- Operational Issues
- Wrap-up



HTTP Sessions

- **Pretty Weak Security**
 - Web applications that don't restrict access to their resources that use sessions (without SSL) are open to session stealing attack
 - If you decide to write your own authorization code, don't rely solely on the session ID for authentication between requests. Not such a risk when using SSL exclusively.
 - Note: WAS 4.0 and above uses JCE SecureRandom() which provides cryptographically random session IDs
 - HttpSession objects may be persisted to a database. If those sessions contain sensitive information, is this secure?
- **Use enable “session security” in Session Manager when feasible**
 - WAS will protect session access to enforce that correct authenticated user accesses the HTTP session – limits the risk of session stealing attacks
 - Sometimes causes problems with access to session from unsecured URLs

Web Application Security Dangers

- Don't set web server and app server doc roots to the same. Otherwise web server will serve up raw WAR content.
- Place only served content (HTML, GIF, and JSPs) in WAR
 - Don't place config files, secrets, etc in WAR otherwise app server will serve (if file serving enabled on)
 - You can safely place things in WEB-INF as this is hidden per spec requirements
- Define a defaultErrorPage in ibm-web-ext.xmi
 - Unhandled errors display a friendly error message instead of stack trace of code

Web Application Security Dangers

- Consider settings these to false in `ibm-web-ext.xml` :
 - File serving (`fileServingEnabled`)
 - Allows remote clients to see specific files in WAR (except WEB-INF)
 - Directory browsing (`directoryBrowsingEnabled`)
 - Allows remote clients to see directory listing of WAR (except WEB-INF)
- Always set this to false in `ibm-web-ext.xml` :
 - Serve servlets by classname (`serveServletsByClassnameEnabled`)
 - Allows remote clients to execute arbitrary classes on classpath
 - Undermines authorization as servlet might have multiple names (classname and URL)

Log and Trace files

- Where are your log files being written to?
- Who can read them?
- Do they contain "sensitive" trace information?
- Using a configurable trace mechanism is a security issue - need to be able to turn off trace after debugging without recompiling the application (else it may be left on to save effort)
 - Application `System.out.println` output ends up in the application server "standard out" log file. Generally, `System.out/err` should be avoided.
 - Better to use a trace system like JRAS or LOG4J.
- By leveraging Java 2 security you can control where application log/trace files get created. This makes it easier to verify that applications follow corporate log/trace guidelines.

Miscellaneous

- **Stateful session EJBs**
 - Used to hold session information (often sensitive) are passivated (written) to disk under certain circumstances
 - WAS “operating system” account should have exclusive access to that directory
 - Snoopers could conceivably reconstitute and read them
- **Don't deploy the sample applications to production**
 - Don't run your application in the default server (if using just base, delete the samples)
- **Don't use the password remembering functions of browsers when you access the Web server, LDAP, WAS admin screens**
- **Consider enforcing CSlv2 transport SSL use (as opposed to optional). Otherwise**
 - will accept non-SSL connections silently
 - Dangerous. You might be running without SSL and not even know it!!

Don't Forget: Consider Whole Environment

- Operating system
- Databases
- LDAP directories
- Application code
- Enterprise systems: CICS, IMS, etc
- SAP, PeopleSoft, etc.

Agenda

- Introduction
- Areas of Security Concern
 - WAS Infrastructure
 - Machine/Operating System Protection
 - Application Security
- Performance
- Operational Issues
- Wrap-up

Performance

■ HTTPS

- Web browser to web server – negligible throughput drop, but large CPU increase on web server.
- Above, plus web server to app server
 - About 10% throughput drop
 - Big jump in CPU use on web server (nearly 100%). If web server is a bottleneck, this will be a problem.
- Different ciphers have slightly different performance – +/- 10%

Performance

- HTTPS – reducing the cost
 - Encrypt only the pages that need it
 - Those with sensitive data. Trouble is, lots of stuff (other than images) is sensitive.
 - Requires that your authentication information be encrypted. Sessions that store security information are then not acceptable.
 - The WAS plugin is smart enough to only use HTTPS if the inbound request used it
 - Hardware acceleration

Performance

- J2EE Authentication & Authorization (implies global security)
 - About 15-20% overhead for a realistic application
- Java 2 Security (without J2EE security)
 - About 7-20% overhead for a realistic application
- Total cost of Java 2 Security + J2EE Security
 - 25-40% ... Ouch!
 - iSeries had best results, while Win2000 had worst results
 - We have not tested AIX or other Unix platforms (yet)
 - WAS development and performance is looking at this to improve the numbers
 - There are already plans for some changes in PTF 2 and more changes later

Performance

- Less security for better performance ??
 - Do you really want to leave the door wide open to make it faster? What about the cost of a break-in or an outage?
 - It is also faster for me to enter my home if I don't lock the front door, but
- Java 2 security is less critical if
 - Infrastructure is not shared. Only one application or closely related set of applications in entire cell.
 - Applications can only compromise its own integrity (e.g., WAS). Risk is reduced.
 - But, beware of rogue applications that want to do harm
- Disable security for the application server if
 - Your application truly has no security needs
 - No logins, no userids. Do not write your own security to address. I mean **no** security.
 - Does not affect WAS admin infrastructure security
 - Results in essentially no performance cost (less than 1%)
 - But, rogue applications remain an issue without Java 2 security

Agenda

- Introduction
- Areas of Security Concern
 - WAS Infrastructure
 - Machine/Operating System Protection
 - Application Security
- Performance
- Operational Issues
- Wrap-up



Operational Duties

Now that you have everything configured, you have to keep it running.
There are a number of tasks that have to be performed regularly.

Key Files and Admin Passwords

- Change your keys/passwords periodically
 - All the keys we created on previous pages
 - All the "admin" passwords (WAS, Web server, LDAP)
 - All the OS passwords
 - Password for the WAS to DB connection account(s)
 - Password for the WAS to LDAP bind account

Managing Certificates/Encryption Keys

- Certificates
 - These must be updated before expiration and should be changed "just for safety"
 - Plugin's and web container's certificates - note that you can control the expiration date when you create the key (one year is the default)
 - WAS global security certificate for IIOP/SSL and SOAP/SSL communication
 - LDAP server's CA certificate for LDAP communication - should not be a big issue since CA certs rarely expire
- LTPA encryption keys used to sign WAS LTPA cookies should be updated occasionally- update using admin console

Monitoring

- You need to monitor your system to ensure that security is maintained
 - Watch the WAS logs for security events
 - Monitor your operating system
 - Monitor your web server
 - Monitor your network for intrusions

Audit, audit, audit ...

- You spent months in meetings to come up with a security policy. Fine. How do you know if anybody read it. Make sure a regular audit is part of your policy.
 - Monitor employees for non-compliance



Stay up to date

- Stay current with your applications
 - Your security policy MUST include frequent checks to various "security warning sites"
 - Assign an administrator the task of checking and keeping each application current
 - WAS - <http://www.ibm.com/software/webservers/appserv>
 - IHS - <http://www.ibm.com/software/webservers/httpservers>
 - LDAP - <http://www.ibm.com/software/network/directory/>
 - DB2 - <http://www.ibm.com/software/data>
 - OS - check with vendor
 - JVM - <http://www.ibm.com/java/jdk>, <http://java.sun.com>

Agenda

- Introduction
- Areas of Security Concern
 - WAS Infrastructure
 - Machine/Operating System Protection
 - Application Security
- Performance
- Operational Issues
- Wrap-up, Q&A

FAQ: What about Tivoli Access Manager?

- Does TAM makes this any easier? – no
 - TAM adds other features (web SSO, auditing, security monitoring, etc), but they are unrelated to the issues I covered here
- Does TAM make this any worse? – no
 - In fact, in some ways, it will make it better with another layer of security
- Does TAM make this any harder?
 - Yes. You still have to do everything here, plus TAM configuration and management.
 - However, if you need the function, the extra effort is justified
- What about TAM competitors?
 - They address similar issues in similar ways, but they generally don't integrate as well with WAS (today).
 - Just remember, you need WAS security even with one of the various access management security products. Trusted Association Interceptors are your friend.

FAQ: Operating System Hardening

- IBM WebSphere development does not test or recommend any hardened operating system configurations
- We require and test for the “standard” operating system packaging from your vendor
- Anecdotal evidence from several customers suggests that WAS can run on a “hardened” operating system

Future Work

- Implications of JMX to security
 - Custom JMX MBeans worry me – do they weaken WAS security?
 - Custom MBean are shared process wide – how does this affect shared infrastructure?
- We need to better define the JMS issues

References/Acknowledgments

- This isn't my work alone:
 - This presentation based on presentation by Tony Cowan and myself from Developer Works 2002
 - Tom Alcott, Sree Ratnasinghe, Benedict Fernandes, and Vinod Jessani provided valuable assistance
- Redbooks
 - WebSphere V5.0 Security SG24-6573 available from <http://www.redbooks.ibm.com>
- This presentation and related tidbits can be found at <http://www.keysbotzum.com> (public) or <http://w3.pittsburgh.ibm.com/~keys/internal> (IBM internal)
 - WAS Security Hardening paper by Keys Botzum (will be updated for 5.0 soon)
 - WAS Security Advanced Topics presentation by Keys Botzum
- Access Manager Info
 - Internal <http://integration.cruz.ibm.com/> (IBM Internal)
 - Enterprise Business Portal with Tivoli Access Manager Redbook (SG24-6556)

© Copyright IBM Corporation 2003. All rights reserved.

IBM, the IBM logo, the e-business logo and other IBM products and services are trademarks or registered trademarks of the International Business Machines Corporation, in the United States, other countries or both. References in this publication to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

Product release dates and/or capabilities referenced in this publication may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

All other trademarks, company, products or service names may be trademarks, registered trademarks or service marks of others.