WebSphere software

IBM

e business software

WebSphere Application Server 5.0 Security – Advanced Topics

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 24, 2003
IBM Software Group

Why are We Here

- This presentation will
  - Describe several security features that are new WAS 5.0
    - These descriptions will be fairly brief assuming some familiarity
    - The focus is WAS specifics and the “whys”
  - Define and solve several common security problems – sort of a mini-FAQ
  - Describe how to integrate WAS with other security systems

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
  - WAS/J2EE programming skills, including some knowledge of the J2EE security features in J2EE 1.3
- 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

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

Agenda

- Introduction
- Technology/Standards
  - CSIv2
  - Java 2 security
  - JAAS
  - Servlet 2.3 mandates
  - JCE/JSSE
  - J2C
- Hints, Tips, Examples
- Integration
- Futures
- References

CSIv2 – What is it?

- New OMG standard for the distributed security service of the CORBA based system. Part of J2EE 1.3 requirements.
- Distinguishes between network level and app level authentication
  - Network level is SSL and PKI authentication (including client certificates)
  - App level is for exchange of security attributes
    - LTPA, basic auth, asserted identities, and Kerberos (future)
- More listener ports
  - TCP/IP
  - SAS SSL Port for compatibility
  - CSIv2 SSL w/o Client Certificate Authentication
  - CSIv2 SSL w/Client Certificate Authentication

SAS vs. CSIv2

- SAS Features
  - SSL required
  - BasicAuth Client Login required
  - Stateful required
- CSIv2 Features
  - SSL/TCPIP Choice
  - Client auth choices
    - SSL Client Authentication
    - BasicAuth (validated), LTPA, Kerberos Client Login (we don't yet support Kerberos)
    - Identity Assertion
  - Stateful/Stateless Choice
  - Better error handling
    - Use of CORBA minor codes and messages
    - Auto retry for errors which can be corrected
  - More flexible configuration (claim/perform, required/supported)

CSIv2 in WAS

- Support for CSIv2 security protocol
  - Conformance level 0
  - Provides **(in theory**) interoperability between different vendor's application servers. To my knowledge this has not yet been tested.
  - SAS protocol supported for backward compatibility (4.0.x and before)
    - SAS and CSIv2 both supported simultaneously
- WAS5 CSIv2 Features
  - Conformance Level 0
  - Most new features developed in CSIv2 only
    - Client certificate authentication, Kerberos (future), etc
  - Used by WAS for IIOP communication. Irrelevant for Web Services/SOAP.
  - Error messages seem improved

CSIv2 - Identity Assertion

- Delegation without the need for a common authentication infrastructure
  - no need to re-authenticate or revalidate the originating client credentials
- When enabled, invocation credential is asserted to the downstream server
  - Normal delegation - user's identity token passed directly to the downstream server. Must share a common trust domain/infrastructure.
  - With assertion - server's identity token is sent along with the client's identity information, **without** proof of client identity.
- Four formats of identities can be present in an identity token
  - A principal name, a distinguished name, a certificate chain, and an anonymous identity.
  - type depends on the original client authentication, E.g.,
    - if the client uses SSL client authentication then the identity token to the downstream server will contain the certificate chain

CSIv2 – Why do I Care?

- CSIv2 lays the groundwork for secure interoperability
- Identity assertion allows for looser secure coupling from server to server
  - Receiving server can perform its own mapping of the identity information
  - This should enable more interoperability with other vendors and platforms
  - I believe this will eventually make secure interoperability with “foreign” systems and other EJB servers a reality
- For most applications and environments, this matters little
  - CSIv2 is just “magic” plumbing
  - Only very special situations require thinking about this

Agenda

- Introduction
- Technology/Standards
  - CSIv2
  - Java 2 security
  - JAAS
  - Servlet 2.3 mandates
  - JCE/JSSE
  - J2C
- Hints, Tips, Examples
- Integration
- Futures
- References

Java 2 Security – What is it?

- Java standard for securing class/method level operations
  - Not for distributed security
  - “Independent” of J2EE security
- By default, WAS enforces J2EE recommended restrictions, e.g.,
  - No thread access
  - EJBs have no file system access
  - Servlets have file system access, but
    - servlets can only read/write files in WAR
  - Can’t even call getUserPrincipal()
  - Lots of restrictions on other APIs
    - No using WAS internal APIs
  - WAS gives itself the permissions it needs

Java 2 Example

- This pieces of code in an EJB will fail (by default) with an access control exception when Java 2 security is enabled
  - String user = getCallerPrincipal();
  - FileOutputStream f = new FileOutputStream(“somefilename”);
- By using Java 2 security, you can precisely define the access rights of code

Policy Files

- Pretty much standard java policy files with a few minor extensions

grant codeBase “url” {

permission <permission class> “<permission option>“, “<sub options>”;

…repeat permissions as many times as needed …

};

…repeat grants as many times as needed…

Example:

```
grant codeBase "file:Utilities.jar" {
        /* grant read permission on file /tmp/foo.txt to Utilities.jar */
         permission java.io.FilePermission "/tmp/foo.txt", "read";
};
```

Policy Files Exist at Multiple Levels

- "static" – don't touch, certainly don't increase restrictions
  - java.policy
  - server.policy
- "dynamic" – WAS dynamically computes what this applies to
  - Node Level
    - spi.policy – provider interfaces, such as JDBC, MQ, etc. Have all permissions by default.
    - app.policy – application default (can override in EAR)
    - library.policy – applies to libraries loaded on this node
  - was.policy – in EAR, applies to this application
  - ra.xml – in RAR, applies to this resource adapter
- Most will only edit
  - was.policy
  - Sometimes library.policy & ra.xml

was.policy

- Applies to this EAR's code
  - Can specify restrictions on EAR, WARs, EJBs, etc
  - Standard java policy format with minor extensions
- Location
  - In EAR – META-INF/was.policy
  - Runtime - <root>/config/cells/<cell>/applications/<app>.ear/deployments/<app>/META-INF/was.policy
- Editing
  - No explicit IBM tool support. Add text file to EAR via AAT or WebSphere Studio.
  - Syntax errors can
    - Cause servers or applications to fail to start
    - Cause WAS to discard was.policy file contents at runtime. One error can undermine entire file.
  - Edit carefully - you may want to use java policytool to edit first time

was.policy - Controlling the Level of Enforcement

- Entire EAR
  - grant codeBase "file:${application}“
- All EJBs
  - grant codeBase “file:${ejbComponent}”
- All Utility JARs
  - grant codeBase “file:${jars}”
- All Web Components
  - grant codeBase “file:${webComponent}”
- A specific module/JAR
  - grant codeBase “file:myWebApplication.war”
  - grant codeBase “file:utility.jar”

Java 2 Security – Why do I Care?

- This is the most significant feature in WAS 5.0 security
  - Makes it truly feasible to build a secure shared infrastructure
  - Really needed because of new security architecture - embedded security server
- Only way to protect WAS internal APIs, file system access, and other system access from rogue applications
  - Crucial in a shared infrastructure

Agenda

- Introduction
- Technology/Standards
  - CSIv2
  - Java 2 security
  - JAAS
  - Servlet 2.3 mandates
  - JCE/JSSE
  - J2C
- Hints, Tips, Examples
- Integration
- Futures
- References

JAAS – What is It?

- JAAS = Java Authentication and Authorization Service
- Loosely based on Sun's PAM (Pluggable Authentication Module)
- Java API/Feature for login, credential manipulation, customized authentication, and authorization
  - Login
    - Standard API for authenticating a user using a login configuration
  - Subject
    - Bunch of security info (open-ended structure)
  - Login Modules
    - Each module authenticates user via callbacks to get credential information (userid, password, etc)
    - Adds Principals and Credentials to Subject
  - Login Configuration
    - Ordered list of Login Modules to be called in sequence to update Subject
    - Basically, you have multiple simultaneous identities in multiple authentications systems: E.g., Kerberos and Operating System

JAAS Usage in WAS

- J2EE 1.3 does not mandate JAAS. It only requires a subset as defined by the Java 2 Connector spec (almost nothing required).
- Login
  - To login:

    ```
    LoginContext  lc = new LoginContext(…);
    lc.login();
    WSSubject.doAs(lc.getSubject(), …)
    ```

  - WSSubject.doAs is required to make CORBA thread current security and Subject consistent
  - Doesn't create LTPA cookie so not for web SSO – must use J2EE web authentication (basic, form, certificate)
  - Useful for clients and within server

JAAS Usage in WAS

- Customized Authentication
  - Can define JAAS Application Login Configuration (or edit existing)
    - Specify login modules
    - This configuration must include the existing WAS login module in order to obtain WAS credentials (which are required for security to work)
  - You can define custom login modules with extra credential info, but
    - Until PTF 2, the custom Subject info can't be accessed
      - Will be accessible using WSSubject.getCallerSubject() or getRunAsSubject()
      - Still not transmitted over between servers
    - Has no effect on WAS Security. WAS credentials still required.
    - Doesn't integrate with TAIs (yet – should in PTF 2)
- Authorization – just Java 2, not JAAS extensions
  - Can define Java policy files, but not Principal constraints
  - Can't build your own custom SecurityManager

JAAS LoginModule and ClassLoaders

- JAAS is part of the JRE
  - Ordinarily, classes have to be on lib/ext to be seen by JAAS since it's part of the JVM
  - This is rather unfortunate as it undermines EAR separation and WAS infrastructure
- WAS has special enhancement - LoginModuleProxy
  - Makes it possible to use code in EAR with JAAS
    - This uses the concept of thread based classloaders
  - When specifying a new JAAS Login Module
    - Use class com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy
    - Custom property
      - delegate = {your LoginModule class}

JAAS – Why do I Care?

- Does provide
  - Standards based API for authentication
  - Way to perform additional authentication checks during login
  - Way to attach additional attributes to a specific user for later use (as of 5.0.2)
    - E.g., you might want to get extra info from LDAP and attach to user
    - Remember that this isn't shared among servers… yet….
- But it isn't
  - a "standards-based" replacement for a custom registry
  - a way to customize authorization
- You still need WAS security, custom registries, TAIs, and standard J2EE web login methods

Agenda

- Introduction
- Technology/Standards
  - CSIv2
  - Java 2 security
  - JAAS
  - Servlet 2.3 mandates
  - JCE/JSSE
  - J2C
- Hints, Tips, Examples
- Integration
- Futures
- References

Servlet 2.3 – Minor Security Changes

- Servlet 2.3 Introduces Run-As for servlet
  - Used when calling an EJB
  - Overrides the usual delegation model where the client identity propagates from the web client to the servlet to the EJB
  - Configured in the deployment descriptor (web.xml)
  - Very useful for handling requests that need to access a secure EJB under another identity
- WAS Run-As does not affect servlet.init() method
  - Servlet 2.4 spec clarifies that init() should be affected by Run-As

JCE/JSSE – What is It?

- Standard APIs
  - JSSE = Java Secure Sockets API
  - JCE = Java cryptography API
  - Widely requested by customers
- Fully supported in WAS 5.0
  - Customers must use the IBM implementation that is in the JDK/JRE directory
    - It should be compatible with Sun's as its behavior is in the spec
    - Do **not** replace with Sun's jsse.jar/jce.jar
  - Customers can add additional providers for JCE
    - Do not remove existing providers
  - Anecdotal evidence suggests that adding new JSSE providers may not work
- IBM JCE/JSSE includes a number of useful providers
  - Probably sufficient for most customers without specialized requirements

JCE/JSSE – Why do I Care?

- Neither is particularly interesting, but are standard
  - Now you have a portable/easy way of performing encryption and opening HTTPS connections

J2C Authentication Providers

- Maps from one identity to another in external system
  - Takes current user identity and generates new identity (in Subject)
  - E.g., take current Subject and create a Subject suitable for establishing CICS session for same user
- Today
  - Used for datasources and other enterprise systems (CICS, IMS, etc)
    - Trivial mapping – all users are mapped to same userid/password
- Why do I Care?
  - Pretty lame today, but in the future ….
    - New modules might perform more complex mapping
      - I assume we'll see smart CICS, IMS, SAP, etc modules
      - Maybe even a smart datasource module
  - You can write your own providers today **IF** the J2C resource adapter can handle what you are doing
    - E.g., custom mappings are of little value if the adapter can't handle (efficiently) the required authentication
    - Tight coupling between adapter and authentication provider

Agenda

- Introduction
- Technology/Standards
- Hints, Tips, Examples
  - Java 2
  - JAAS
  - Authentication
  - Etc
- Integration
  - Authentication
  - Authorization
  - Access Manager
- Futures
- References

was.policy Example – Read/Write Files

```
grant codeBase "file:${application}" {
    /* all of these work */
    permission java.io.FilePermission "/temp/foo.txt", "read";
    permission java.io.FilePermission "c:/temp/foo3.txt", "read, write";
    permission java.io.FilePermission "c:\\temp\\foo2.txt", "read, write";

    /* this works */
    permission java.io.FilePermission "c:\\temp\\foodir\\-", "read";

    /* This doesn't work */
    permission java.io.FilePermission "c:/temp/foodir2/-", "read";
};
```

- Be aware of Unix vs. NT pathname syntax

was.policy Example – Get User Identity

- This enables the use of WSSubject.getCallerPrincipal(), EJBContext.getCallerPrincipal(), and req.getUserPrincipal()

```
grant codeBase "file:${application}" {
    permission java.security.SecurityPermission "printIdentity";
};
```

was.policy Example - JAAS Login and WSSubject.doAs

```
grant codeBase "file:${application}" {
        permission javax.security.auth.AuthPermission "createLoginContext";
        permission javax.security.auth.AuthPermission "doAs";
       /* notice the quoting. The outer string contains 3 parts. The last part is the
           principal name which must also be quoted.
       TBD: this may be giving away too many permissions. */
        permission javax.security.auth.PrivateCredentialPermission "* * \"*\"", "read";
};
```

Determining Needed Java 2 Permissions

- Unfortunately, the docs are lacking. Fortunately, you can figure it out.

[2/7/03 21:21:38:293 EST] 44c2189b WebGroup E SRVE0026E: [Servlet Error]-[access denied ;java.security.SecurityPermission printIdentity]: java.security.AccessControlException: access denied (**java.security.SecurityPermission printIdentity**) permission

→ java.security.SecurityPermission "printIdentity“

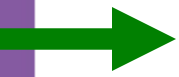
[2/7/03 21:48:26:415 EST] 205ed8ae WebGroup E SRVE0026E: [Servlet Error]-[access denied ;java.io.FilePermission c:\home\.wscprc read]: java.security.AccessControlException: access denied (**java.io.FilePermission c:\home\.wscprc read**)

→ permission java.io.FilePermission “c:\\home\\.wscprc”, “read”

Configuring Java 2 Permissions is Tedious

- It can be quite tedious to determine the permissions needed by an application
  - You get an exception, fix it, then repeat
- com.ibm.websphere.java2secman.norethrow is your friend during development
  - Set on JVM as a system property
  - Java 2 security violations are reported in the log, but not enforced
  - Collect all of the violations and address them all at once

Agenda



- Introduction
- Technology/Standards
- Hints, Tips, Examples
  - Java 2
  - JAAS
  - Authentication
  - Etc
- Integration
  - Authentication
  - Authorization
  - Access Manager
- Futures
- References

JAAS Login (Server Side)

```
public Subject login(PrintWriter out) throws LoginException {
        try {
                LoginContext  lc = new LoginContext( "WSLogin",
                new WSCallbackHandlerImpl(userid, "realm", password));
                lc.login();
                out.println("Login done successfully<br>");
                return lc.getSubject();
        } catch (LoginException e) {
                // insert error processing code
                throw e;
        }
}
```

JAAS doAs()

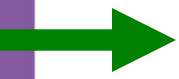
```
….login and do something…
Subject s = login(out);
if (s != null) {
        com.ibm.websphere.security.auth.WSSubject.doAs(s, new Runner(out));
}
logout(s);
….
class Runner implements java.security.PrivilegedAction {
        PrintWriter out;
        public Runner(PrintWriter out) {
                this.out = out;
        }
        public Object run() {
                doSomethingSecure(out));
                return null;
        }
};
```

JAAS Logout

- Beware of credential leaks
  - Here we destroy them
  - May make more sense to create Subject cache and reuse Subjects

```
public void logout(Subject s, PrintWriter out) throws LoginException {
        try {
                LoginContext  lc = new LoginContext("WSLogin", s);
                lc.logout();
                out.println("Logout done.");
        } catch (LoginException le) {
                out.println("Failed to logout");
        }
}
```

Agenda



- Introduction
- Technology/Standards
- Hints, Tips, Examples
  - Java 2
  - JAAS
  - Authentication
  - Etc
- Integration
  - Authentication
  - Authorization
  - Access Manager
- Futures
- References

Why Did Login Fail? – Today/WAS 5.0.1

- There is nothing (yet) in WAS to tell you why a login failed
  - WAS does not provide detailed information on authentication error
  - InfoCenter gives an example that uses ErrorReporter from a JSP – this example is completely wrong and will be removed
  - No one can see the registry error (except trace).
- To determine why login failed
  - Use servlet filters
    - Detect that login failed based on HTTP error code
    - Repeat login to registry to determine error from underlying registry.
    - Beware:
      - May trigger premature account lockouts (extra login attempts)
      - WAS Registries provide no useful error info. You'll have to go directly to "real" registry.
      - Have to write your own HTTPServletResponse

Login Failure Reason – WAS 5.0.2

- Get “last failure” on thread (server or client side)
  - Throwable t = com.ibm.websphere.security.auth.WSSubject.getRootLoginException();
- Step through exception tree to find “cause”
  - Throwable t = com.ibm.websphere.security.auth.WSLoginFailedException.getCause()
  - Throwable t = com.ibm.websphere.security.WSSecurityException.getCause()
- Eventually, you’ll get to the “real” exception. That’s the cause. E.g.,
  - com.ibm.ws.security.registry.nt.NTException
  - javax.naming.AuthenticationException
- Works in client or server. Works in servlet or EJB.
  - Will be disabled by default in client for security reasons

Who Called Me?

- J2EE defines APIs for getting user info
  - Request.getUserPrincipal(), EJBContext.getCallerPrincipal()
- Outside of J2EE constructs, there is no “standard” way to obtain the identity of the current user
- IBM provides an extension – WSSubject with static methods
  - Can be called from anywhere. Identity of current thread.
    - WSSubject.getCallerPrincipal() -- > userid of caller (a string)
  - With PTF 2, WSSubject will be extended
    - You can obtain the JAAS Subject
      - Read only
      - Requires appropriate Java 2 permissions
    - WSSubject.getCallerSubject() – JAAS Subject of Caller
    - WSSubject.getRunAsSubject() – JAAS Subject of RunAs attribute

Custom User Attributes

- Sometimes people want custom user attributes
  - Organization or other registry info
  - Special access rights, etc
- Use a post login servlet filter
  - Add attributes to HTTPSession
  - Might be wrong abstraction: servlet vs EJB needs
- Use JAAS w/ custom module
  - Get the Subject and add attributes post login
  - Will flow transparently and be accessible w/ WAS V5.0.2, but won't flow from server to server
  - May not work with all authentication methods
- Write a simple in memory cache keyed off of user identity
  - Lookup data as needed and cache in memory
  - Might have to perform duplicate lookups in a distributed environment
  - Cache invalidation can be tricky

Anonymous Thread Problem

- A thread needs to access some secured resource (e.g., an EJB)
  - Threads that are created are detached from the WAS runtime
    - No J2EE context
    - No identity, security will fail any request requiring authorization
- WAS 5.0 Enterprise Extensions
  - Async Bean service addresses these issues for arbitrary parallel threads
- Generally less of an issue with base/ND since separate threads forbidden
  - Servlets still have this problem
    - init() method
    - Servlets with anonymous clients

Need an Identity

- Option #1: Run-As – easy, simple, incomplete
  - Set Run-As on servlet methods, EJB methods, etc
  - Doesn't address servlet init()
- Option #2: JAAS Login
  - Use JAAS to obtain an authenticated Subject and use WSSubject.doAs()
  - Be cautious of performance overhead and credential leaks. Caching and reusing Subjects probably warranted.

Agenda

- Introduction
- Technology/Standards
- Hints, Tips, Examples
  - Java 2
  - JAAS
  - Authentication
  - Etc
- Integration
  - Authentication
  - Authorization
  - Access Manager
- Futures
- References

LDAP Directory Authorization

- WAS searches the user registry for users
  - LDAP directories often contain sensitive information
    - Corporate organization information
    - Personal phone numbers / email addresses
  - Some organizations want to restrict access to more than just the passwords
    - Restrict access to the whole registry by disabling anonymous browsing
    - Restrict access to the individual sensitive fields using ACL's
  - TAM also does this. It sets ACLs on the LDAP registry.
- WAS can authenticate to registry if needed (anonymous is used by default)
  - In LDAP user registry configuration, specify bind DN and password
  - Should not be LDAP admin account
  - Ideally account just for WAS, with only read access

Sync Node

- If you make major changes to the security settings and all nodes aren't up and running life becomes fun
- Nodes can't start because they don't know new security settings
  - Get authentication/authorization failures when trying to talk to dmgr
- Have to force a sync using syncNode.bat

Admin Tool Password Prompting

- WAS security runtime needs a password if security enabled
  - Can be stdin, prompt, properties, or specified programmatically
  - WAS tools rely on the implicit prompting rather than prompting themselves
  - Still often use 'prompt' (GUI based) by default
- You want to force to use stdin prompting as is normal with command line tools
  - SOAP layer is incapable of prompting at all
  - Must force to use RMI JMX connectors
    - –conntype RMI –port <bootstrap port>.
      - Bootstrap is usually 9809 on dmgr and 2809 for nodes.
      - Applies to most cmd line tools, including wsadmin. If you leave out port or there is an error, will silently fall back to SOAP.
      - tperfviewer is special, use "./tperfviewer.bat localhost 9809 RMI"
    - You still need to edit sas.client.props to use stdin instead of prompt
  - Beware that RMI is firewall hostile

Security Timeout/Tuning

- LTPA expiration time
  - Controls the lifetime of the cookie sent to the browser
  - Once the timeout occurs, user will have to login again
  - Default: 120 minutes - Shorter values strengthen security
    - LTPA tokens harder to steal
    - Increases user frustration
    - Harms performance
  - Recommendation: leave as is unless customer needs dictate otherwise
- Cache timeout time
  - Controls how long WAS caches information related to security
    - information from the registry: groups, etc
  - Default: 600 seconds - Shorter values improves security
    - System is able to detect registry changes faster
    - Harms performance
  - Recommendation: Increase this unless registry changes frequently/critical
  - Size of caches can also be tuned using various undocumented properties
  - In PTF 2 there will be a way to force a cache flush

Getting Current User Registry Handle

```
import com.ibm.websphere.security.UserRegistry;
{
        Context ic = new InitialContext();
        Object objRef = ic.lookup("UserRegistry");
        UserRegistry userReg = (UserRegistry)PortableRemoteObject.narrow( objRef,
            UserRegistry.class);
        out.println(" registry says display name is: " +
            userReg.getUserDisplayName(req.getUserPrincipal().getName()));
}
```

- You'll need to add wssec.jar to your classpath in order to compile your code.

Small Tips

- WAS_UseRemoteRegistry
  - Set on Global Security as custom property
  - Can be “node” or “cell”
  - Forces registry to run out of process eliminating need for root access for app servers w/ OS registry
  - Introduces dependency on Node manager
  - → I suspect use of this will be rare
- Logging users out
  - ibm_security_logout URL
- WebSphere Studio does not seem to enforce Java 2 security in spite of having a checkbox for it

Agenda

- Introduction
- Technology/Standards
- Hints, Tips, Examples
  - Java 2
  - JAAS
  - Authentication
  - Etc
- Integration
  - Authentication
  - Authorization
  - Access Manager
- Futures
- References

On Integration

- Definition
  - Integration is about using WAS security in the real world where there are other security products and secure systems. Fortunately, WAS 4.0 is specifically designed to address some, but not all, of the integration issues.
- Motivation
  - WAS is not an island
  - WAS security must integrate into real a company with existing systems
    - user registries
    - authentication systems

Direct LTPA Integration

- Domino has support for recognizing LTPA tokens
- Multiple WAS cells
  - if each WAS cell shares LTPA encryption keys and registry, then they can communicate securely
  - export LTPA keys using Security Center and import into peer domains
- See InfoCenter for details on Domino integration and multiple WAS cells

Trusted Associations

- WAS can be told to "trust" web proxy authentication systems via Trusted Associations
  - Write custom plug-in Java code that WAS calls when identifying user web request. WAS will trust your code to identify user.
  - WAS will still obtain some information from registry (LDAP or custom)
  - Only works with web proxies. Won't help with other SSO systems. For example, existing windows authentication (without some magic).
    - Note that both Access Manager and a forthcoming WAS enhancement will address Windows SSO

Writing a Trusted Interceptor

- extend TrustAssociationInterceptor interface
- **isTargetInterceptor(HttpServletRequest req)**
  - examine incoming request to see if request applies to this interceptor (there can be several)
  - probably look for some expected cookie or header
- **validateEstablishedTrust(HttpServletRequest req)**
  - examine request to determine if request can be trusted
  - take the time to find a secure way to validate information
  - for example, with some systems, the request includes a secret password. If using some form of token, verify digital signature and extract user information
  - verifying IP address is **not** sufficient. Header can be forged.
  - if can't verify, rely on network security (definitely use plugin HTTPS)
- **getAuthenticatedUsername(HttpServletRequest req)**
  - return the user's name as a string
  - WAS will lookup name and group information in registry and create credentials

Writing a Trusted Interceptor

- Be careful, you are extending WAS trust domain!!
  - WAS is trusting the interceptor to validate identity. If the interceptor can be tricked, WAS is now open to attack
  - Code should be reviewed by at least one other senior technical person
  - Never do this without careful analysis and design
- See InfoCenter section for details
- Effort:
  - Probably 1-4 weeks of development effort if detailed information about proxy available at start, but highly skilled and extremely security conscious person required.
  - Realistically involves weeks of meetings to get needed information
- Sample TAI on my web site
  - Shows issues to be addressed
  - Not suitable for reuse

Custom Registry

- Normally, WAS supports either OS registry or one LDAP registry.
- Can create custom registry which provides access to arbitrary registries
  - implement UserRegistry interface
  - configure into WAS
- Registries may not use WAS services
  - bootstrap issue during initialization
  - Issue in node manager (no J2EE infrastructure)

Writing a Custom Registry

- Extend UserRegistry interface
- The interface has been improved over 4.0. Here's a summary of the key functions:
- Result getUsers(String pattern, int max)
  - Return all users matching the pattern (e.g, "a*")
  - In LDAP: (&(objectclass=person)(uid="a*"))
  - The max limits the size of the result set
- Result checkPassword(String userId, String password)
  - Verify that user's password is valid. Done via ldap_bind in LDAP.
- String getUniqueUserId(String userName)
  - Convert username to registry specific unique value (DN in LDAP)
  - In LDAP, search and return DN.

Writing a Custom Registry

- String getUserSecurityName(String uniqueUserId)
  - Convert registry specific unique value to unique username.
  - In LDAP, lookup given DN and return uid
- List getUniqueGroupIds(String userName)
  - Return groups for this user
  - In LDAP, search for user's DN, then search for groups with member=this user's DN
- Similar for groups

Writing a Custom Registry

- Caveats
  - uniqueness is important. If your registries contain duplicate ids, life can get fun and ambiguous.
  - fault tolerance is critical. If there is an error, WAS expects you to recover automatically after the error.
  - watch out for multithreading issues
- Effort:
  - About 2-4 weeks of development effort, but highly skilled person required
  - Realistically this is preceded by weeks of meetings to get the needed information on the existing registry
- Sample quality custom registry on my web site
  - Supports combining multiple registries together
  - Replication and failover
  - Can be reused, sometimes with little change

Warnings About Interceptors and Registries

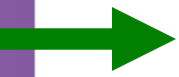
- Writing security code is difficult, be careful:
  - typically much harder than initially appears
  - will it scale?
    - fast enough?
    - handle large numbers of users and groups?
  - is it really secure?
    - consider points of attack
    - can people forge credentials?
  - is it robust?
    - what about failure scenarios?
  - is it manageable?
    - how will you debug this? (hint: use WAS JRAS)
  - testing can be very difficult and time consuming

TAM Authentication Integration with WAS

- Web SSO integration
  - WebSEAL/TAM authenticates web browser user and passes that information on to WAS
    - Leverages standard WAS TAI function
    - WAS ships with WebSEAL/TAM TAI
      - Configure via admin console, under LTPA section
  - Access Manager can directly issue LTPA tokens - **deprecated**
  - Both work great

Agenda

- Changes
- Technology/Standards
- Hints, Tips, Examples
  - Java 2
  - JAAS
  - Authentication
  - Etc
- Integration
  - Authentication
  - Authorization
  - Access Manager
- Futures
- References



Authorization

- WAS needs to authorize access to resources
- Ordinarily, WAS follows normal J2EE defined rules
- This authorization decision can be externalized
  - JSR 115: Java Authorization Service Provider Contract for Containers
    - WAS will support in the future
    - I expect many products will implement
  - Limited use of IBM proprietary authorization interface (TAM uses)
- Be aware, some products claim integration that does not really exist
  - May require turning off WAS security
  - May only authorize URLs
  - May require special code generation

TAM Authorization Integration with WAS

- WAS delegates authorization decisions to TAM
  - Asks TAM to determine user mapping to J2EE roles on the fly
  - WAS then applies J2EE specified constraints based on role
- This may be more complicated than it appears
  - Domain conflicts between WAS and TAM.
    - Somewhat difficult to express and manage versus native product tools.
    - E.g., it's easier to specify a WPS security constraint in the WPS than in TAM
  - Some evidence of performance issues, particularly with WPS
- Applications can also directly use TAM APIs to obtain credentials and make authorization decisions, but I do not endorse this

Database Integration

- 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 Integration

- 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.
    - not going to work with CMP beans
    - Can be made to work with WAS datasource using 5.0 features (we have this working with DB2)
    - not all databases support this

Agenda

- Introduction
- Technology/Standards
- Hints, Tips, Examples
  - Java 2
  - JAAS
  - Authentication
  - Etc
- Integration
  - Authentication
  - Authorization
  - Database
  - Access Manager
- Futures
- References

Tivoli Access Manager (TAM) and WAS

- Good products that are leaders in their areas
  - WAS is a Java application server (J2EE, etc)
  - TAM is a web SSO/advanced authentication and enterprise authorization tool
- Both compliment the other **when appropriate**
  - If your customer uses WebLogic and TAM, why buy WAS?
  - If your customer uses WebSphere, does not always benefit from TAM
- WAS is a high quality secure product in its own right
  - Identify specific needs and then determine what additional value TAM provides
  - TAM adds powerful security **features**, it doesn't make WAS significantly more secure

Tivoli Access Manager (TAM) and WAS

- Beware of these facts
  - Proxy servers are hard to make work with existing apps
    - Rewriting all embedded URLs in arbitrary app is impossible - think about embedded JavaScript and applets
    - Often requires code changes.
    - WebSEAL in web server plugin mode addresses this – unclear what functional issues this may raise
  - TAM is a big product in its own right
    - Additional products add complexity and cost
    - If the function is needed, this is worth it. If not, ….

When TAM is a Good Fit

- Things TAM does, that WAS doesn't address
  - Web SSO across multiple products (WAS and WebLogic for example)
  - Cross DNS domain SSO
  - Advanced authentication: SecureID, password strength testing, password expiration, login rules, RACF authentication, etc
    - By design, WAS doesn't do this and neither does IBM Directory Server
    - Note: other LDAP directory products can enforce some rules. But, still no UI component for handling problems from web interface.
  - Classic build vs. buy situation. Need to understand how many features you need and evaluate the value of buying versus building.

When TAM is a Good Fit

- Already looking into Web SSO or authorization tools (e.g., TAM competitors)
  - TAM simply works better with WAS and WPS then the competition
    - TAIs ship with WAS
    - WAS and WPS authorization can call out to TAM
    - Integration is real, not smoke and mirrors – although there may be some issues

When to Consider TAM – Other Reasons

- Your like the idea of a central authorization rules server
  - Only TAM (today) can integrate with WAS and WPS in this way
- You believe in “defense in depth”
  - WebSEAL can be placed in DMZ in front of web server. Only authenticated traffic enters enterprise intranet.
  - Be aware:
    - Now a high function product is in the DMZ which requires lots of infrastructure in a place that is supposed to be minimal
    - Proxy server configuration raises URL issues that must be addressed early in development
- These are reasonable choices, but I’m not comfortable recommending them. It’s up to you to decide.

Agenda

- Introduction
- Technology/Standards
- Hints, Tips, Examples
  - Java 2
  - JAAS
  - Authentication
  - Etc
- Integration
  - Authentication
  - Authorization
  - Database
  - Access Manager
- Futures
- References

Possible Future Features

- End to End Security
  - Improved J2C
    - I'm optimistic this will someday provide a way to transmit identity information from WAS to other systems: CICS, database, etc
  - Kerberos
  - JSR 115: Java Authorization Service Provider Contract for Containers
- Authorization
  - Embedded Tivoli Access Manager components
  - Instanced based admin authorization
  - Fine-grained Namespace ACLs
- Run-As for servlet.init() method (required by spec)
- Better JAAS Integration
  - Roles mapping defined by JAAS module
  - The custom Subject isn't transmitted to other servers as of today. It will be in the future.

References/Acknowledgments

- This isn't my work alone:
  - This presentation based on presentation by Tony Cowan and myself from Developer Works 2002
- Redbooks
  - WebSphere V5.0 Security SG24-6573 available from http://www.redbooks.ibm.com
- This presentation and related tidbits can be found at http://w3.pittsburgh.ibm.com/~keys/internal
  - WAS Security Hardening paper (will be updated for 5.0 soon)
  - WAS Security Hardening Presentation
- Access Manager Info
  - Internal http://integration.cruz.ibm.com/
  - Enterprise Business Portal with Tivoli Access Manager Redbook (SG24-6556)

Appendix

Login Failure Examples

- 5.0.2 example using IBM extensions
  - Login Error Servlet
  - Preferred choice by far
- 5.0/5.0.1 example using servlet filters and retry logic
  - Filter that Determines Login Error Reason

Login Error Servlet Listing

```
package person.botzum.servlet;

import java.io.IOException;
import java.io.PrintStream;
import java.io.PrintWriter;

import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl;

public class LoginTestServlet extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

        PrintWriter out = resp.getWriter();

        String userid = req.getParameter("userid");
        String password = req.getParameter("password");
```

Login Error Servlet Listing

```
out.println("<br>userid = " + userid);
        out.println("<br>password = " + password);
        try {
            LoginContext ctx =
                new LoginContext(
                    "WSLogin",
                    new WSCallbackHandlerImpl(userid, password));
            ctx.login();
            out.println("login succeeded");
            out.println(
                "you are "
                    + ctx.getSubject().getPrincipals().iterator().next());
        } catch (LoginException e) {
            out.println("<br>Login failed w/ Loginexception:" + e);

            Throwable t = e;
            while (true) {
                t = determineCause(t, out);
                if (t == null)
                    break;
            }
        } catch (Exception e) {
            out.println("<br>Login failed w/ exception:" + e);
            e.printStackTrace(out);
        }
    }
```

Login Error Servlet Listing

```
static public Throwable determineCause(Throwable e, PrintWriter out) {
        out.println("<br>*****************************************<br>");
        e.printStackTrace(out);

        Throwable t = null;
        out.flush();

        try {
            if (e    instanceof com.ibm.websphere.security.auth.WSLoginFailedException) {
                t =    ((com .ibm..websphere .security .auth .WSLoginFailedException) e)      .getCause();
            }

            if (e instanceof com.ibm.websphere.security.WSSecurityException) {
                t =   ((com.ibm.websphere.security.WSSecurityException) e) .getCause();
            }
        } catch (IndexOutOfBoundsException e2) {
            out.println("Got that Index error again. Ignoring...");
        }

        if (t != null) {
            out.println("<br>this was caused by this exception = " + t);
            return t;
        } else {
            out.println("No root cause!!");
            printFinalExceptionInfo( e, out);
            return null;
        }
    }
```

Login Error Servlet Listing

```
static private void printFinalExceptionInfo(Throwable e, PrintWriter out) {
        if (e instanceof com.ibm.ws.security.registry.nt.NTException) {
                out.println("<br>This is an NT exception. Error code = " + ((com.ibm.ws.security.registry.nt.NTException)
        e).getErrorCode());
                return;
        }

        if (e instanceof javax.naming.AuthenticationException) {
                javax.naming.AuthenticationException e2 = (javax.naming.AuthenticationException) e;
                out.println("<br>This is a Java naming exception: " + e2);
                out.println(", message: " + e2.getExplanation());
                out.println("<br> root cause: " + e2.getRootCause());
                return;
        }

        out.println("Final Exception: " + e);
   }

public void doPost(HttpServletRequest req, HttpServletResponse resp)
       throws ServletException, IOException {

   }

public void init() throws ServletException {

       super.init();

   }
}
```

Filter That Determines Login Failure Reason

```
public class LoginFilter implements Filter {
public void doFilter( ServletRequest req,ServletResponse resp,FilterChain chain) throws
      ServletException, IOException {
        String userid = req.getParameter("j_username");
        String password = req.getParameter("j_password");
        MyResponse newres = new MyResponse((HttpServletResponse) resp);
        chain.doFilter(req, newres);
        if (newres.getStatus() != newres.SC_OK) {
                HttpSession session = ((HttpServletRequest) req).getSession();
                try {
                /* Repeat authentication here.     Note: can't use WAS registry as it hides the underlying
                     error from you. Have to go direct to LDAP…sigh…
                */
                } catch (Exception e) {
                        session.setAttribute("exception", e);
                }
        }
        newres.completeRedirect();
}
```

Custom HTTPServletResponse

```
public class MyResponse
    implements
    HttpServletResponse {
  HttpServletResponse res;
  int status = SC_OK;
  String desiredRedirect = null;
  public
      MyResponse(HttpServletRespons
      e res) {
        this.res = res;
        res.setBufferSize(32000);
  }
  public void completeRedirect()
      throws IOException {
        if (desiredRedirect != null)
        res.sendRedirect(desiredRedire
            ct);
  }
```

```
public int getStatus() {
      return status;
}
public void sendRedirect(String
    arg0) throws IOException {
      desiredRedirect = arg0;
      //res.sendRedirect(arg0);
}
….
public void addCookie(Cookie
    arg0) {
      res.addCookie(arg0);
}
…..repeat for every method in
    HttpServletResponse…
```