SSL Considerations for CAS: Planning, Management, and Troubleshooting

Marvin Addison
Middleware Services
Virginia Tech
October 13, 2010

Agenda

- Planning and deployment considerations
- Discussion of Java SSL model
- Tools for certificate management
- Client and server SSL configuration points
- Troubleshooting common problems

Planning and Deployment

Planning: SSL or !SSL

- SSL is *strongly* recommended for all environments
- Exceptions
  - Protyping/Testing
  - Small and secure networks (you probably don't have one that is both)
  - You have excellent insurance and PR to cover security breaches
- SSL is worth the investment in all cases

Planning: Choice of PKI

- Commecial vendor (e.g. Thawte, Verisign)
- Institutional PKI signed by commercial vendor
- Institutional PKI *not* signed by commercial vendor
- Test PKI
- Self-signed certificates

Planning: Commercial PKI

- Commercial certs and institutional certs signed by commecial vendor require little to no configuration
- Commecial certs have cost per unit downside
- Initial cost for institutional root certificate signing is high
- Institutional cert ROI is good
- InCommon certificate services – bargain

Planning: Institutional Certs

For certificates issued by an institution with a self-signed root certificate:

- Excellent cost per unit (typically 0)
- Requires trust configuration on services, user browsers, and possibly server
- Browser configuration precludes this option in many cases

Planning: Protyping+Testing

- Avoid self-signed certs
- Consider test PKI
  - As easy as self-signed certs
  - Single certificate to configure for trust
- Use fully-qualified hostnames in cert CN
- DNS infrastructure practically required

Planning: Test PKI

- Generate a self-signed test root cert
  - `openssl genrsa -out test-root.key 2048`
  - `openssl req -x509 -new -days 10000 -out test-root.crt`
- Generate test certs for each service
  - `openssl genrsa -out test-svc-1.key 2048`
  - `openssl req -new -key test-svc-1.key -days 5000 -out test-svc-1.csr`
  - `openssl x509 -days 5000 -req -in test-svc-1.csr -CA test-root.crt -CAkey test-root.key -out test-svc-1.crt -CAcreateserial`
- Configure root cert trust on clients + server

Java SSL Model

SSL Model: Keystore Primitive

- http://download.oracle.com/javase/1.5.0/docs/guide/security/jsse/JSSERefGuide.html
- Keystores and truststores have same structure
- Usage determines function
  - Keystore contains credentials (cert/key pairs) for SSL client authentication
  - Truststore contains trusted remote peer issuers/certificates

SSL Model: Container vs CAS

- Container configuration required to access CAS over SSL
  - Keystore used to hold cert/key pair
  - Tomcat <connector> element, e.g.
- CAS configuration concerns focus exclusively on *system* truststore
  - Clients must trust server for ticket validation
  - Server must trust client for proxy callback authentication to succeed

SSL Model: Container

- Tomcat Java connectors use (abuse) keystore for server certificate/key pair
- Tomcat native connector uses discrete PEM/DER-encoded files ala mod_ssl
- The container keystore has *absolutely nothing* to do with system truststore that provides trust material to HttpsURLConnections used in direct client-server communication

SSL Model: System Truststore

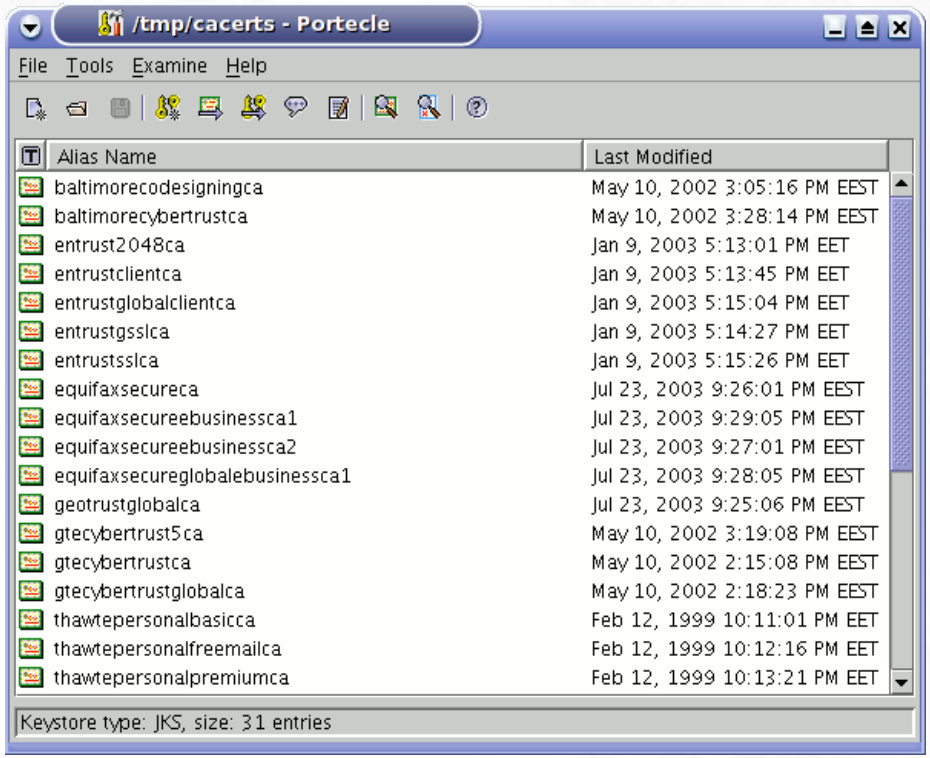
- Default is $JRE_HOME/lib/security/cacerts
- The default doubles as both the system keystore and truststore
- Use custom path by setting javax.net.ssl.trustStore system property (e.g. via $CATALINA_HOME/bin/setenv.sh)
- Be aware of javax.net.ssl.trustStoreType and javax.net.ssl.trustStorePassword

Tools for Certificate Management

Tools: Keytool

- Sun keytool utility is included with JDK
- Basic functionality is adequate in most cases
- Adequate command-line documentation
- Shortcomings
  - Cannot import cert/key pairs
  - Cannot export keys

Tools: Portcle



- GUI tool
- Supports all import, export operations
- Supports any keystore format supported by JRE (e.g. JKS, PKCS12, BKS)

Tools: vt-crypt

- Includes keystore command-line utility
- keystore == keytool++
- Supports any keystore format supported by JRE (e.g. JKS, PKCS12, BKS)
- Embedded documentation and examples
- Excellent replacement for keytool

Tools: OpenSSL

- OpenSSL + PKCS12 = Sweet
- Leverage JDK 1.5+ support for PKCS12 keystores, e.g. javax.net.ssl.trustStoreType=PKCS12
- Standard-based certificate/key container format
- Leverage existing knowledge, excellent documentation, and power of OpenSSL

Client and Server SSL Configuration Points

Server Config: SSO Cookie

- CASTGC cookie only delivered to client over HTTPS by default
- No SSL means no SSO by default
- **DANGER**: Set secure property of CookieRetrievingCookieGenerator to false to send over plain HTTP

Server Config: Proxy Auth

- Requirements for proxy authentication
  - Callback is HTTPS URL
  - Response is 200, 202, 301, 302, 304 (can be customized via HttpClient class)
- Server must trust client SSL certificate to make HTTPS connection
- May require system truststore configuration on server

Client Config: Ticket Validation

- Ticket validation involves a direct connection from client (CAS service) to server
- Available configuration points
  - Trust everything (difficult in Java, **AVOID**)
  - Trusted issuers (e.g. CA certificates)
  - Trusted certificates (**AVOID**)
  - Hostname verification (e.g. hostname mismatch, wildcard with subdomains)

Client Config: Trust Everything

- Java: Inject promiscuous TrustManager into SSLContext#init() method, then use context to obtain a SSLSocketFactory to create SSLSockets for connections. WHEW!
- phpCAS::setNoCasServerValidation();
- .NET: Attach custom RemoteCertificateValidationCallback to ServicePointManager.ServerCertificateValidationCallback

Client Config: Trusted Issuers

- Java: Add CA certs to system truststore
- phpCAS::setCasServerCACert($path); where $path is path to PEM-encoded CA cert that issued CAS server cert
- .NET: Add CA certs to Local Machine store using Certificate Management MMC snap-in
- mod_auth_cas: CASCertificatePath directive (same meaning as phpCAS above)

Client Config: Host Verification

- Java: TicketValidator#setHostnameVerifier()
  - AnyHostnameVerifier
  - RegexHostnameVerifier
  - WhitelistHostnameVerifier
- .NET: ServicePointManager.ServerCertificateValidationCallback

Troubleshooting

Troubleshooting: Reference

- Common problems
  - Remote certificate not trusted
  - Hostname mismatch
- Answers: https://wiki.jasig.org/display/CASUM/SSL+Troubleshooting+and+Reference+Guide

Troubleshooting: Top 3

- PKIX path building failed – add issuer of remote certificate to system truststore
- No subject alternative names present – ticket validation URL should not contain IP address
- HTTPS hostname wrong – hostname mismatch (e.g. wildcard with subdomain)

Troubleshooting: SSL Trace

- An SSL trace is the *best* way to diagnose any SSL problem
- Set system property: javax.net.debug=ssl
- Output goes to STDOUT (e.g. catalina.out)
- See https://wiki.jasig.org/display/CASUM/SSL+Troubleshooting+and+Reference+Guide for Tomcat setup example

Troubleshooting: Other Debugging Options

- Complete listing at http://download.oracle.com/javase/1.5.0/docs/guide/security/jsse/JSSERefGuide.html#Debug
- java.security.debug=certpath is a good secret

Questions