Securing REST APIs with SSL/TLS

Youssef Oujiangaa

OWASP Netherlands
2016-04-21
Personal Introduction

- Youssef Ouajamaa
  - Software Engineer
    - Full-stack Java SE/EE, JavaScript, AngularJS
    - Software system design
  - Computer Security Enthusiast
    - Secure code analysis
    - Web, Linux, OpenBSD
Contents

• REST APIs
• SSL/TLS
• HTTPS
• Use Case
• Cons and Pros
• Configuring Your Application Server
• Certificate Management
• Continues Integration & Delivery
• Hardening Apache Tomcat
REST APIs

- Client-Server
- HTTP
  - GET, POST, DELETE, PUT
  - http://securityevents.com/api/resource
- Stateless
- JSON

```
{
  "ticketId": 12,
  "eventName": "Security Conference 2020",
  "price": 41.95,
  "presentations": [
    "0days",
    "buffers"
  ]
}
```
SSL/TLS

- Secure Sockets Layer (SSL)
- Transport Layer Security (TLS)
- OSI Model

<table>
<thead>
<tr>
<th>Application Layer</th>
<th>HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentation Layer</td>
<td>SSL/TLS</td>
</tr>
<tr>
<td>Transport Layer</td>
<td>TCP</td>
</tr>
<tr>
<td>Network Layer</td>
<td>IP</td>
</tr>
</tbody>
</table>
SSL/TLS

- Symmetric Cryptography
- Key Exchange
  - RSA
  - Diffie-Hellman
  - ECDH
- Cipher
  - AES
- Certificate Authority
  - Commercial solutions
  - Self-signing
HTTPS

• HTTP over TLS
• Securely transfers
  – URL, Headers, Cookies and Body
• Insecurely transfers
  – Hostname and Port Number (TCP/IP)
• One and two way TLS authentication
Use Case

- Cross network communication
- Transactions between organizations

Untrusted Network
Let’s do it!

We know there is a short road, so let’s pave it!
Cons and pros

• Proper Key Infrastructure
  – Investment
• Network Infrastructure
  – Proxies, Firewalls, Load Balancers
• Distribution of server and client certificates
• The initial handshake is still slow
• Possible single point of failure
Java EE 6 Working Example

https://github.com/youjamaa
Mutual Authentication

Client
- Truststore
- Keystore

Server
- Keystore
- Truststore

Realm

Secure Line
- Request Resource
- Send Server Certificate
- Send Client Certificate
- Lookup
- Verify Certificate

Verify Certificate
Configuring Mutual Authentication

- server.xml

```xml
<Connector port="7777" protocol="HTTP/1.1" SSLEnabled="true"
    maxThreads="150" scheme="https" secure="true" clientAuth="false"
    sslProtocol="TLS" />
```

- web.xml

```xml
<login-config>
    <auth-method>CLIENT-CERT</auth-method>
    <realm-name>owasprealm</realm-name>
</login-config>
```
Configuring Mutual Authentication

• Realm configuration
  `<Realm className="org.apache.catalina.realm.MemoryRealm" />

• tomcat-users.xml
  `<tomcat-users>
    <role rolename="owasprealm" />
    <user username="CN=CertName,OU=Marketing,O=Company,L=Osaka,ST,S=Osaka,C=JP"
      password="null"
      roles="owasprealm" />
  </tomcat-users>

• DataSourceRealm to use a database
• JNDIRealm to use an LDAP server
Securing Resource Paths

<security-constraint>

  <web-resource-collection>
    <web-resource-name>someName</web-resource-name>
    <url-pattern>/api/marketing/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>

  <auth-constraint>
    <role-name>owasprealm</role-name>
  </auth-constraint>

  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>

</security-constraint>
Certificate Management

• Enrollment
  – Creation and signing
• Provisioning
  – Uploading keystore and trustores
• Monitoring
  – Logging events and tracking certificates
• Revocation
  – Preparing for the worst

- Manual work and it’s prone to errors
  - *i.e.* risk increases as each step requires human interaction
  - Possible mitigation by applying the four-eyes principle
We need to automate this!
Continues Integration & Delivery

• Ideal world – Zero Touch
• Automatic generation of certificates
  – Keys, certificates, stores and their passwords generated
  – Data is stored in a vault
• Signing
  – Uploading and signing
• Provisioning
  – Configuration based provisioning
• Revoking
  – Single source for revocation
Hardening Apache Tomcat

- sslEnabledProtocols (same as the sslProtocol option)
  - SSLv2 and SSLv3 are not secure
  - Use TLSv1.2, TLSv1.1, TLSv1

- Ciphers
  - Really old ciphers like Triple DES are enabled by default!
  - Explicitly specify secure ciphers and key exchange methods

- Configure a secure realm
  - MemoryRealm based updates require a restart
  - Use the LockOutRealm
Hardening Apache Tomcat

- Plain text password mess
  - truststorePass, keystorePass, keyPass all visible in server.xml
  - Mitigation
    - Password provisioning during application deployment
    - Secure access on operating/file system level

- Deciding between OpenSSL vs. JSSE
  - OpenSSL seems to be haunted with security issues
  - At first sight JSSE seems more secure but it could be obscurity
  - Performance wise; Java 8 supports hardware acceleration for cryptographic operations
    - JEP 164: Leverage CPU Instructions for AES Cryptography
Hardening Apache Tomcat

- Disable client-initiated renegotiation
  - Java 8 features a new option
    jdk.tls.rejectClientInitiatedRenegotiation
That’s all for now! Questions? :)

Open Web Application Security Project