Architecture of Enterprise Applications 6 Security III

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents

REliable, INtelligent & Scalable Systems

- Single Sign-On
  - Overview
  - Kerberos protocol
  - SAML
  - CAS
- Securing Web Application

Single Sign-On - wikipedia

- **Single sign-on** (**SSO**) is a property of access control of multiple related, but independent software systems.
  - With this property a user logs in once and gains access to all systems without being prompted to log in again at each of them.
  - Conversely, **Single sign-off** is the property whereby a single action of signing out terminates access to multiple software systems.

- As different applications and resources support different authentication mechanisms,
  - single sign-on has to internally translate to and store different credentials compared to what is used for initial authentication.

Single Sign-On - wikipedia

REliable, INtelligent & Scalable Systems

- **Benefits** include:
  - Reduces phishing success, because users are not trained to enter password everywhere without thinking.
  - Reducing password fatigue from different user name and password combinations
  - Reducing time spent re-entering passwords for the same identity
  - Reducing IT costs due to lower number of IT help desk calls about passwords
  - Security on all levels of entry/exit/access to systems without the inconvenience of re-prompting users
  - Centralized reporting for compliance adherence.

Common Single Sign-on

REliable, INtelligent & Scalable Systems

- Kerberos based
  - MIT Kerberos protocol
- Smart card based
  - Initial sign-on prompts the user for the smart card.
  - Additional software applications also use the smart card, without prompting the user to re-enter credentials.
  - Smart card-based single sign-on can either use certificates or passwords stored on the smart card.
- OTP token
  - Also referred to as one-time password token.
- Security Assertion Markup Language
  - Security Assertion Markup Language (SAML) is an XML-based solution for exchanging user security information between an enterprise and a service provider.

Kerberos

- Kerberos
  - is a computer network authentication protocol which works on the basis of "tickets" to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner.

- MIT developed Kerberos to protect network services provided by Project Athena.
  - The protocol was named after the character *Kerberos* (or *Cerberus*) from Greek mythology which was a monstrous three-headed guard dog of Hades.
  - Recent release: 08 Aug 2012 - krb5-1.10.3

- **User Client-based Logon**
  - A user enters a username and password on the client machines.
  - The client performs a one-way function (hash usually) on the entered password, and this becomes the secret key of the client/user.

- **Client Authentication**
  - The client sends a clear text message of the user ID to the AS requesting services on behalf of the user. (Note: Neither the secret key nor the password is sent to the AS.)
  - The AS generates the secret key by hashing the password of the user found at the database (e.g. Active Directory in Windows Server).

- The AS checks to see if the client is in its database. If it is, the AS sends back the following two messages to the client:
  - Message A: *Client/TGS Session Key* encrypted using the secret key of the client/user.
  - Message B: *Ticket-Granting-Ticket* (which includes the client ID, client network address, ticket validity period, and the *client/TGS session key*) encrypted using the secret key of the TGS.

- Once the client receives messages A and B, it attempts to decrypt message A with the secret key generated from the password entered by the user.
  - If the user entered password does not match the password in the AS database, the client's secret key will be different and thus unable to decrypt message A.
  - With a valid password and secret key the client decrypts message A to obtain the *Client/TGS Session Key*. This session key is used for further communications with the TGS. (Note: The client cannot decrypt Message B, as it is encrypted using TGS's secret key.)
  - At this point, the client has enough information to authenticate itself to the TGS.

- **Client Service Authorization**
- When requesting services, the client sends the following two messages to the TGS:
  - Message C: Composed of the TGT from message B and the ID of the requested service.
  - Message D: Authenticator (which is composed of the client ID and the timestamp), encrypted using the *Client/TGS Session Key*.

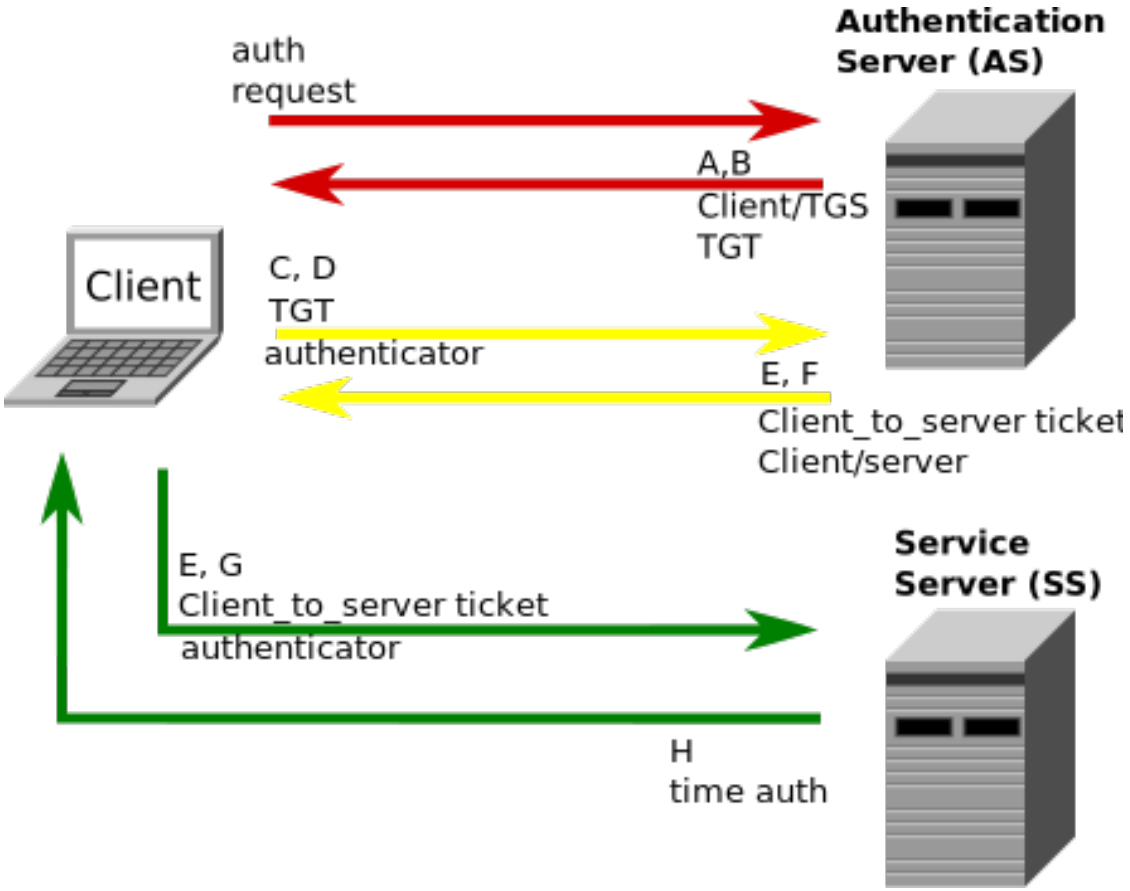
- Upon receiving messages C and D, the TGS retrieves message B out of message C. It decrypts message B using the TGS secret key. This gives it the "client/TGS session key". Using this key, the TGS decrypts message D (Authenticator) and sends the following two messages to the client:
  - Message E: *Client-to-server ticket* (which includes the client ID, client network address, validity period and *Client/Server Session Key*) encrypted using the service's secret key.
  - Message F: *Client/Server Session Key* encrypted with the *Client/TGS Session Key*.

- **Client Service Request**
- Upon receiving messages E and F from TGS, the client has enough information to authenticate itself to the SS. The client connects to the SS and sends the following two messages:
  - Message E from the previous step (the *client-to-server ticket*, encrypted using service's secret key).
  - Message G: a new Authenticator, which includes the client ID, timestamp and is encrypted using *Client/Server Session Key*.

- The SS decrypts the ticket using its own secret key to retrieve the *Client/Server Session Key*. Using the sessions key, SS decrypts the Authenticator and sends the following message to the client to confirm its true identity and willingness to serve the client:
  - Message H: the timestamp found in client's Authenticator plus 1, encrypted using the *Client/Server Session Key*.

- The client decrypts the confirmation using the *Client/Server Session Key* and checks whether the timestamp is correctly updated. If so, then the client can trust the server and can start issuing service requests to the server.
- The server provides the requested services to the client.

Kerberos


Authentication
Server (AS)
auth
request
A,B
Client/TGS
TGT
Client
C, D
TGT
authenticator
E, F
Client_to_server ticket
Client/server
Service
Server (SS)
E, G
Client_to_server ticket
authenticator
H
time auth

Kerberos

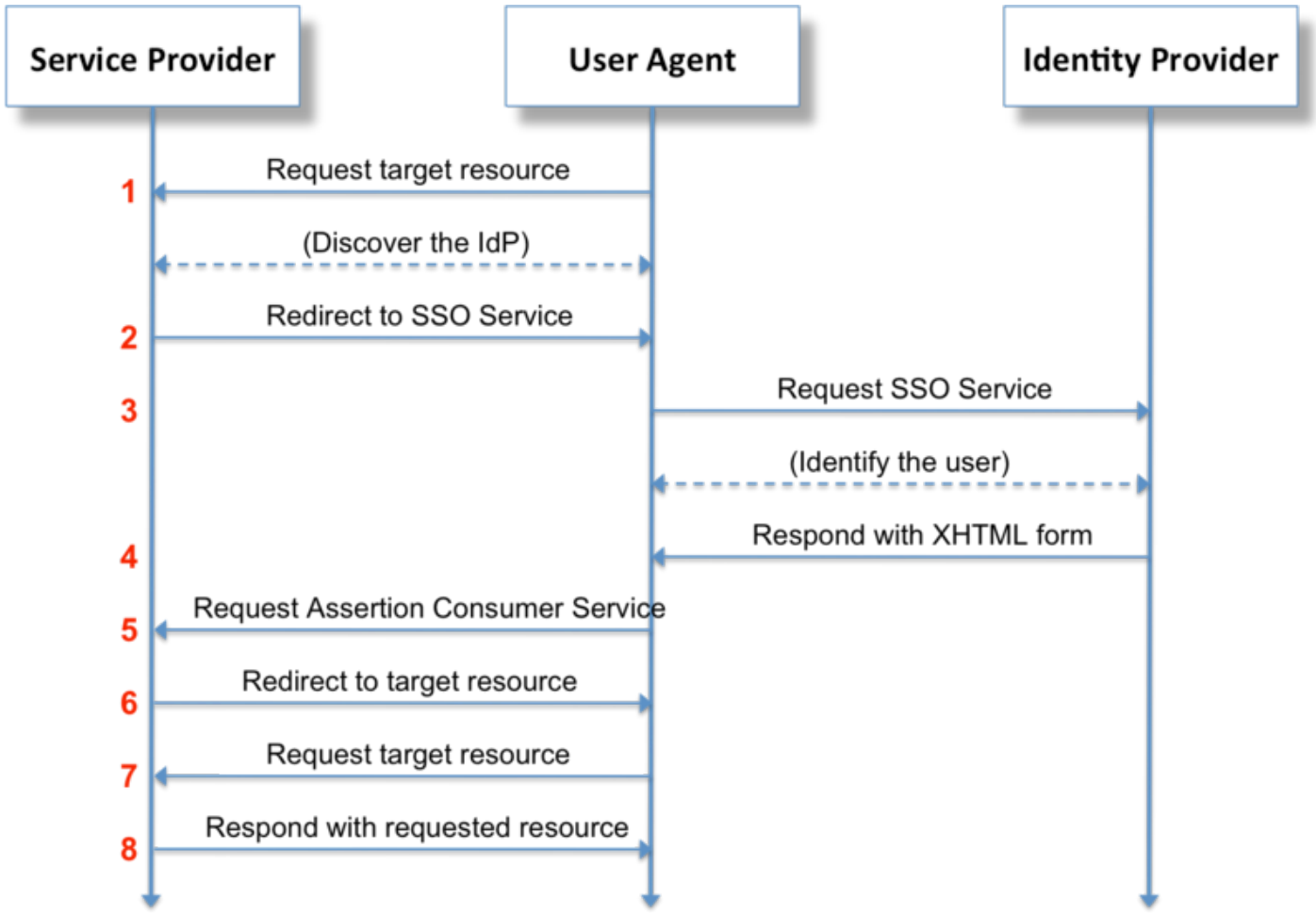
- Drawbacks and Limitations
  - Single point of failure.
  - Kerberos has strict time requirements, which means the clocks of the involved hosts must be synchronized within configured limits.
  - The administration protocol is not standardized and differs between server implementations.
  - Since all authentication is controlled by a centralized KDC, compromise of this authentication infrastructure will allow an attacker to impersonate any user.
  - Each network service which requires a different host name will need its own set of Kerberos keys. This complicates virtual hosting and clusters.

SAML

- Security Assertion Markup Language is an XML-based open standard data format
  - for exchanging authentication and authorization data between parties,
  - in particular, between an identity provider and a service provider.

- SAML is a product of the OASIS Security Services Technical Committee.
  - SAML dates from 2001.

- The primary SAML use case is called *Web Browser Single Sign-On (SSO)*.

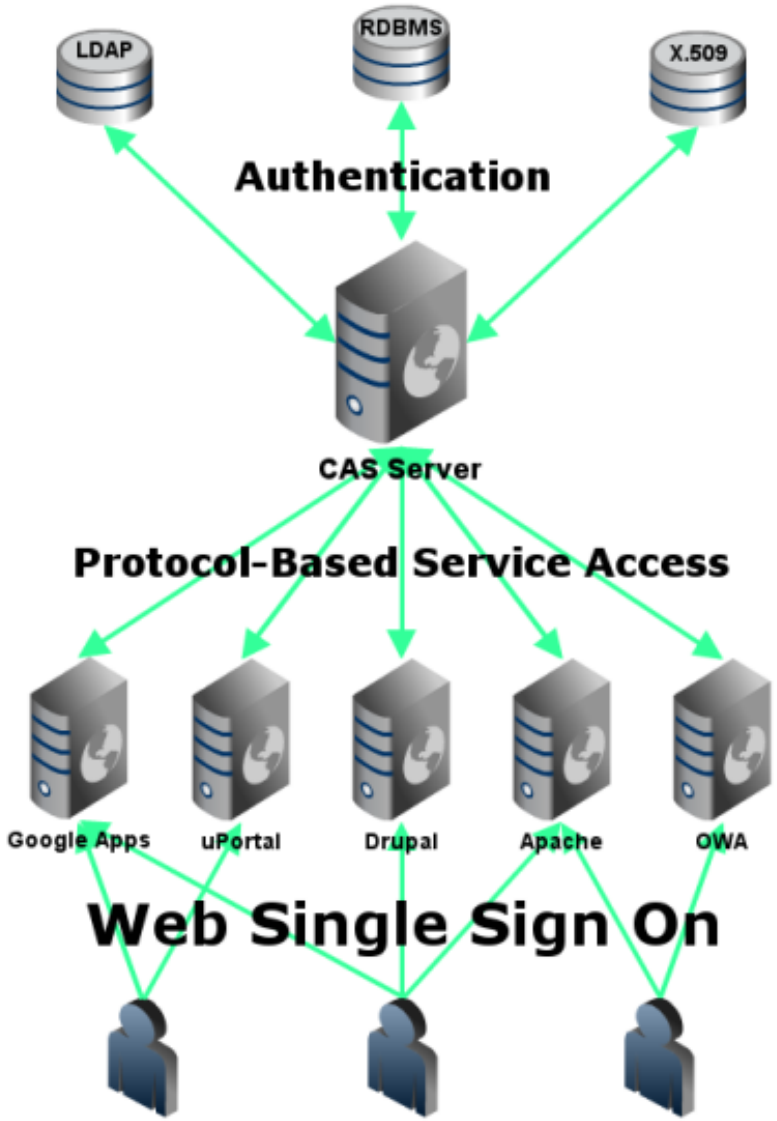


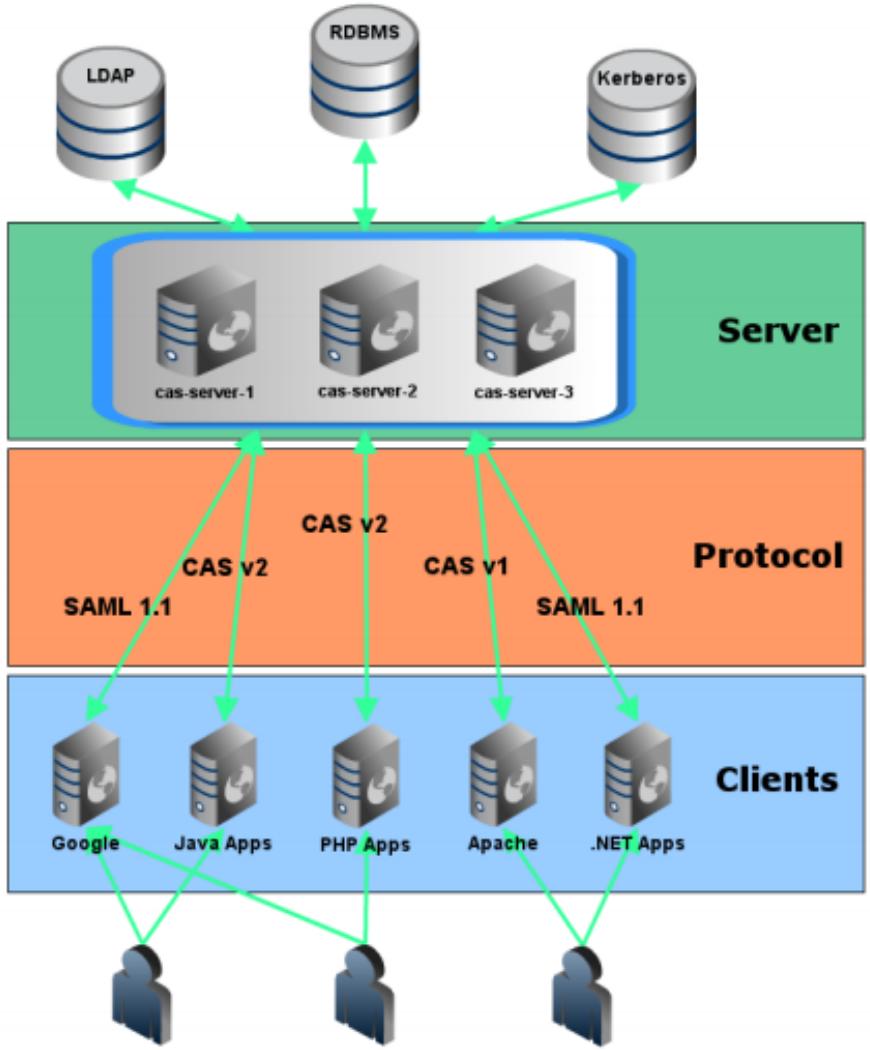
CAS

REliable, INtelligent & Scalable Systems

- Central Authentication Service provides enterprise single sign-on service:
  - An open and well-documented protocol
  - An open-source Java server component
  - A library of clients for Java, .Net, PHP, Perl, Apache, uPortal, and others
  - Integrates with uPortal, BlueSocket, TikiWiki, Mule, Liferay, Moodle and others
  - Community documentation and implementation support
  - An extensive community of adopters

- Latest version: 3.5.0 final


LDAP
RDBMS
X.509
Authentication
CAS Server
Protocol-Based Service Access
Google Apps
uPortal
Drupal
Apache
OWA
Web Single Sign On


LDAP
RDBMS
Kerberos
cas-server-1
cas-server-2
cas-server-3
Server
CAS v2
CAS v2
CAS v1
SAML 1.1
SAML 1.1
Protocol
Google
Java Apps
PHP Apps
Apache
.NET Apps
Clients

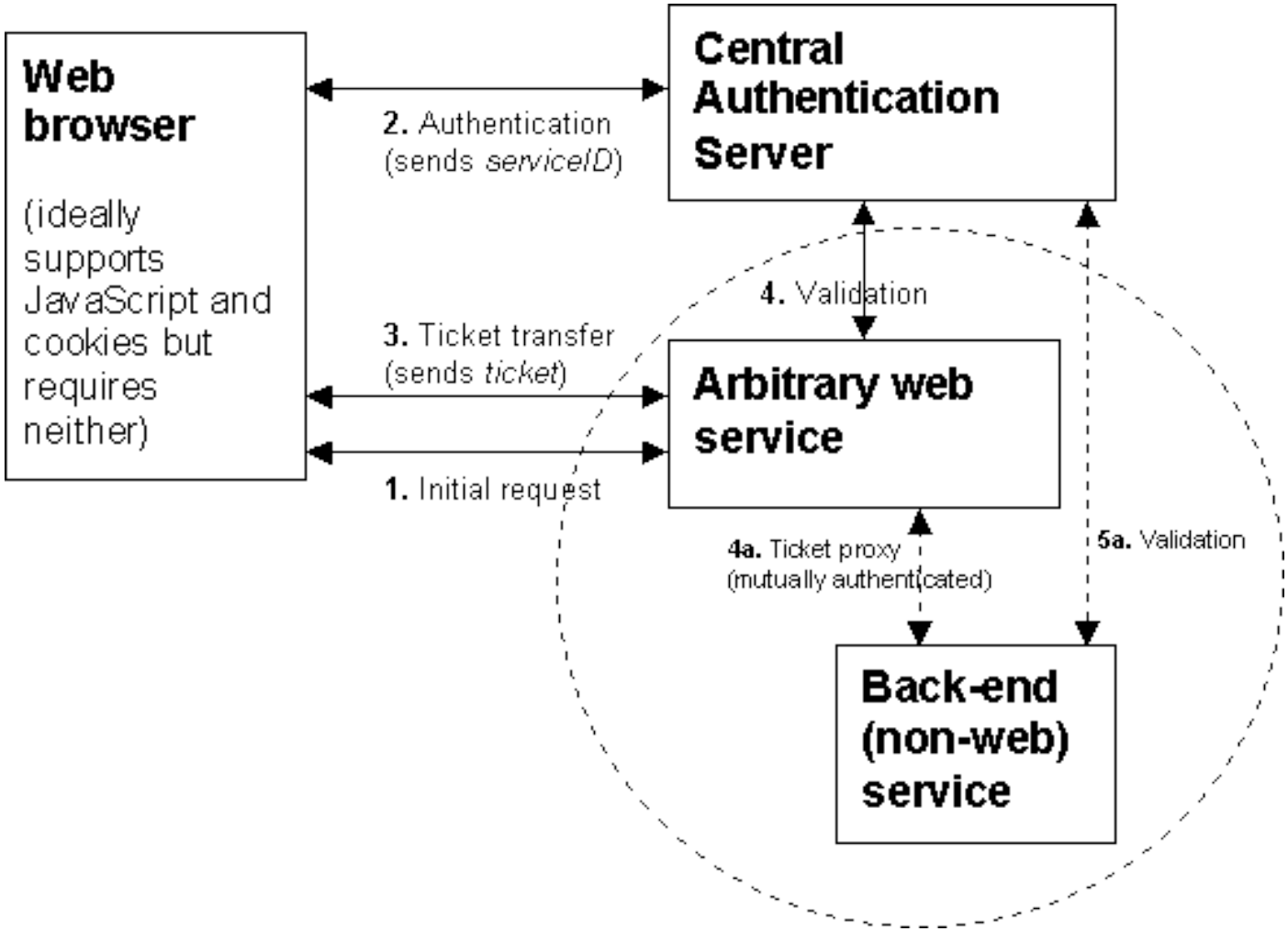
List of URIs to access CAS.

REliable, INtelligent & Scalable Systems

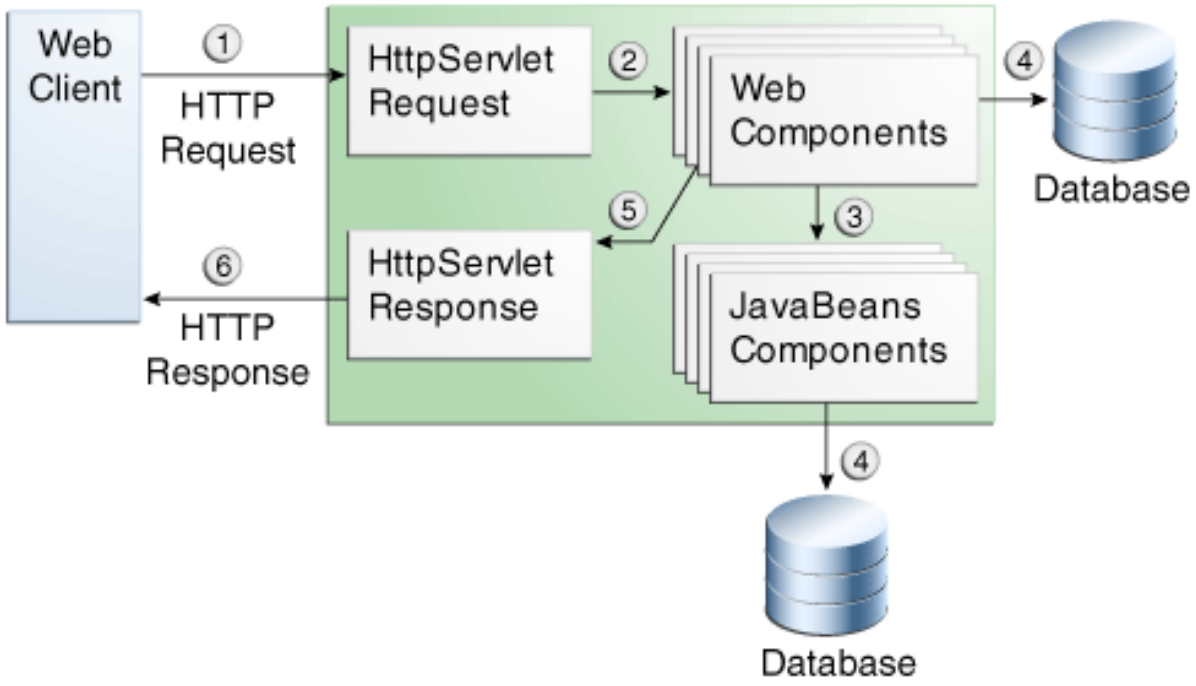
- /login
  - Parameters: service, renew, gateway, warn
- /logout
  - Parameters: url
- /validate
  - Parameters: service, ticket, renew
- /serviceValidate
  - Parameters: service, ticket, pgtUrl, renew
- /proxy
  - Parameters: pgt, targetService
- /proxyValidate
  - Parameters: service, ticket, pgtUrl, renew

CAS Architecture

REliable, INtelligent & Scalable Systems



- Declarative security
- Programmatic security
- Message Security



Specifying Security Constraints

- A security constraint is used to define the access privileges to a collection of resources using their URL mapping.

- The following subelements can be part of a security-constraint:
  - **Web resource collection** (web-resource-collection): A list of URL patterns (the part of a URL *after* the host name and port you want to constrain) and HTTP operations (the methods within the files that match the URL pattern you want to constrain) that describe a set of resources to be protected.
  - **Authorization constraint** (auth-constraint): Specifies whether authentication is to be used and names the roles authorized to perform the constrained requests..
  - **User data constraint** (user-data-constraint): Specifies how data is protected when transported between a client and a server.

Specifying a Web Resource Collection

- A web resource collection consists of the following subelements:
- web-resource-name is the name you use for this resource. Its use is optional.
- url-pattern is used to list the request URI to be protected
  - If you set up the paths for your web application so that the pattern /cart/* is protected but nothing else is protected.
  - Assuming that the application is installed at context path /myapp, the following are true:
    - http://localhost:8080/myapp/index.xhtml is *not* protected.
    - http://localhost:8080/myapp/cart/index.xhtml *is* protected.

Specifying a Web Resource Collection

- A web resource collection consists of the following subelements:

- http-method or http-method-omission is used to specify which methods should be protected or which methods should be omitted from protection. An HTTP method is protected by a web-resource-collection under any of the following circumstances:
  - If no HTTP methods are named in the collection (which means that all are protected)
  - If the collection specifically names the HTTP method in an http-method subelement
  - If the collection contains one or more http-method-omission elements, none of which names the HTTP method

Specifying Authentication Mechanisms

REliable, INtelligent & Scalable Systems

```
<!-- SECURITY CONSTRAINT #1 -->
<security-constraint>
   <web-resource-collection>
        <web-resource-name>wholesale</web-resource-name>
        <url-pattern>/acme/wholesale/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
        <role-name>PARTNER</role-name>
   </auth-constraint>
   <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
   </user-data-constraint>
</security-constraint>
```

Specifying a Web Resource Collection

REliable, INtelligent & Scalable Systems

```
<!-- SECURITY CONSTRAINT #2 -->
<security-constraint>
   <web-resource-collection>
        <web-resource-name>retail</web-resource-name>
        <url-pattern>/acme/retail/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
        <role-name>CLIENT</role-name>
   </auth-constraint>
   <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
   </user-data-constraint>
</security-constraint>
```

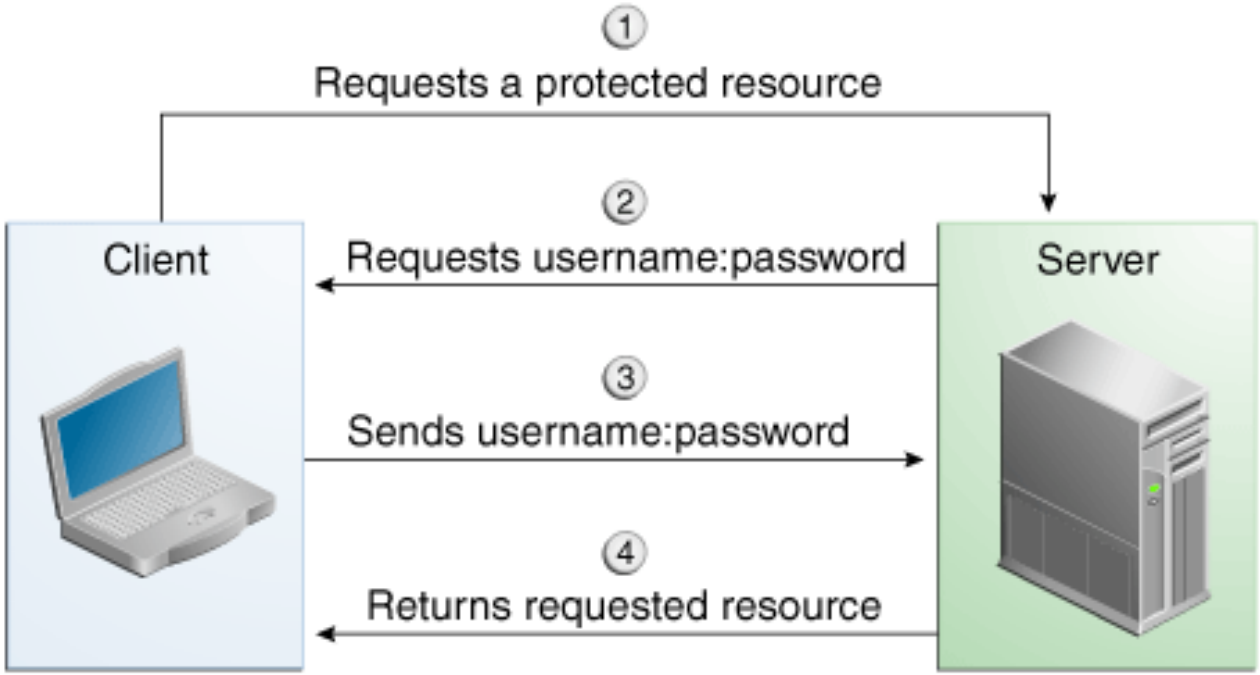
Specifying Authentication Mechanisms

- A user authentication mechanism specifies
  - The way a user gains access to web content
  - With basic authentication, the realm in which the user will be authenticated
  - With form-based authentication, additional attributes

- The Java EE platform supports the following authentication mechanisms:
  - Basic authentication
  - Form-based authentication
  - Digest authentication
  - Client authentication
  - Mutual authentication

Authentication

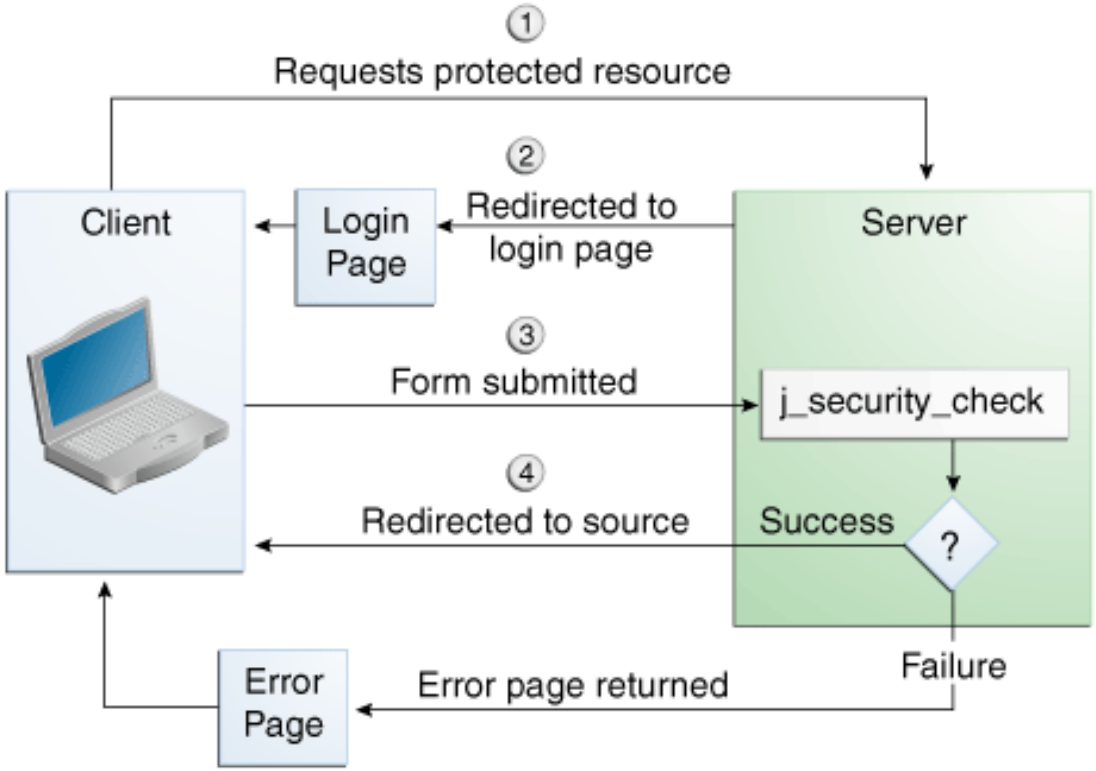
- Basic authentication



Authentication

- Form-based authentication

```
<form method="POST" action="j_security_check">
 <input type="text" name="j_username">
 <input type="password" name="j_password">
</form>
```



Authentication

REliable, INtelligent & Scalable Systems

```
<security-constraint>
  <web-resource-collection>
      <web-resource-name>Protected Area</web-resource-name>
      <url-pattern>/security/protected/*</url-pattern>
      <http-method>PUT</http-method>
      <http-method>DELETE</http-method>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
      <role-name>manager</role-name>
  </auth-constraint>
</security-constraint>

<!-- Security roles used by this web application -->
<security-role>
      <role-name>manager</role-name>
</security-role>
<security-role>
      <role-name>employee</role-name>
</security-role>
```

Authenticating Users Programmatically

REliable, INtelligent & Scalable Systems

```
@WebServlet(name="TutorialServlet", urlPatterns={"/TutorialServlet"})
public class TutorialServlet extends HttpServlet {
 @EJB private ConverterBean converterBean
 protected void processRequest(HttpServletRequest request,
                               HttpServletResponse response)
                               throws ServletException, IOException {
     response.setContentType("text/html;charset=UTF-8");
     PrintWriter out = response.getWriter();
     try {
         out.println("<html>");
         out.println("<head>");
         out.println("<title>Servlet TutorialServlet</title>");
         out.println("</head>");
         out.println("<body>");
         request.login("TutorialUser", "TutorialUser");
         BigDecimal result = converterBean.dollarToYen(new BigDecimal("1.0"));
         out.println("<h1>Servlet TutorialServlet result of dollarToYen= " +
                        result + "</h1>");
         out.println("</body>");
         out.println("</html>");
     } catch (Exception e) { throw new ServletException(e); }
     finally { request.logout(); out.close(); }
 }
}
```

Authenticating Users Programmatically


```
public class TestServlet extends HttpServlet {

protected void processRequest(HttpServletRequest request,
     HttpServletResponse response)
     throws ServletException, IOException
{
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
          request.authenticate(response);
          out.println("Authenticate Successful");
      } finally {
          out.close();
       }
}
```

Authenticating Users Programmatically


```
package enterprise.programmatic_login;

import java.io.*;
import java.net.*;
import javax.annotation.security.DeclareRoles;
import javax.servlet.*;
import javax.servlet.http.*;

@DeclareRoles("javaee6user")
public class LoginServlet extends HttpServlet {

 /** * Processes requests for both HTTP GET and POST methods.
  * @param request servlet request
  * @param response servlet response
  */
  protected void processRequest(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException
  {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
```

Authenticating Users Programmatically


```
    try {
         String userName = request.getParameter("txtUserName");
         String password = request.getParameter("txtPassword");
         out.println("Before Login" + "<br><br>");
         out.println("IsUserInRole?.." +
                   request.isUserInRole("javaee6user")+"<br>");
         out.println("getRemoteUser?.." + request.getRemoteUser()+"<br>");
         out.println("getUserPrincipal?.." +
                   request.getUserPrincipal()+"<br>");
         out.println("getAuthType?.." + request.getAuthType()+"<br><br>");

         try {
             request.login(userName, password);
         } catch(ServletException ex) {
             out.println("Login Failed with a ServletException.." + ex.getMessage());
             return;
         }

         out.println("After Login..."+"<br><br>");
         out.println("IsUserInRole?.." + request.isUserInRole("javaee6user")+"<br>");
         out.println("getRemoteUser?.." + request.getRemoteUser()+"<br>");
         out.println("getUserPrincipal?.." + request.getUserPrincipal()+"<br>");
         out.println("getAuthType?.." + request.getAuthType()+"<br><br>");
         request.logout();
         out.println("After Logout..."+"<br><br>");
         out.println("IsUserInRole?.." + request.isUserInRole("javaee6user")+"<br>");
         out.println("getRemoteUser?.." + request.getRemoteUser()+"<br>");
         out.println("getUserPrincipal?.." + request.getUserPrincipal()+"<br>");
         out.println("getAuthType?.." + request.getAuthType()+"<br>");
    } finally {
         out.close();
    }
   }
...
}
```

Declaring and Linking Role References

REliable, INtelligent & Scalable Systems

```
<servlet>
  ...
    <security-role-ref>
       <role-name>cust</role-name>
       <role-link>bankCustomer</role-link>
    </security-role-ref>
  ...
</servlet>

<security-role>
    <role-name>bankCustomer</role-name>
</security-role>
```

Project

REliable, INtelligent & Scalable Systems

- To design your security system of your project
  - Develop the SimpleLoginModule in Courseware V, and use it as the authentication method in your project
  - Develop the billing function in a security way (Encrypt financial information)
  - Develop a customized permission to view categories of books. The categories are modeled as a tree. You can grant permission to codebase(or principal) to describe which code(or who) can view which categories of book.

References

- Single Sign On, http://en.wikipedia.org/wiki/Single_sign-on
- Kerberos: The Network Authentication Protocol, http://web.mit.edu/kerberos/
- Kerberos(protocol), http://en.wikipedia.org/wiki/Kerberos_(protocol)
- SAML, http://en.wikipedia.org/wiki/Security_Assertion_Markup_Language
- CAS, http://www.jasig.org/cas
- Jasig CAS Documentation, http://www.unicon.net/files/cas-server-3-4-11-snapshot-manual.pdf
- CAS ppt, http://www.jusfortechies.com/java/cas/ppt.php

Thank You!