

Authentication and Authorization of PRISM users

Proposed Solution

Xavier Le Pasteur
for Fujitsu Systems Europe

october 2004

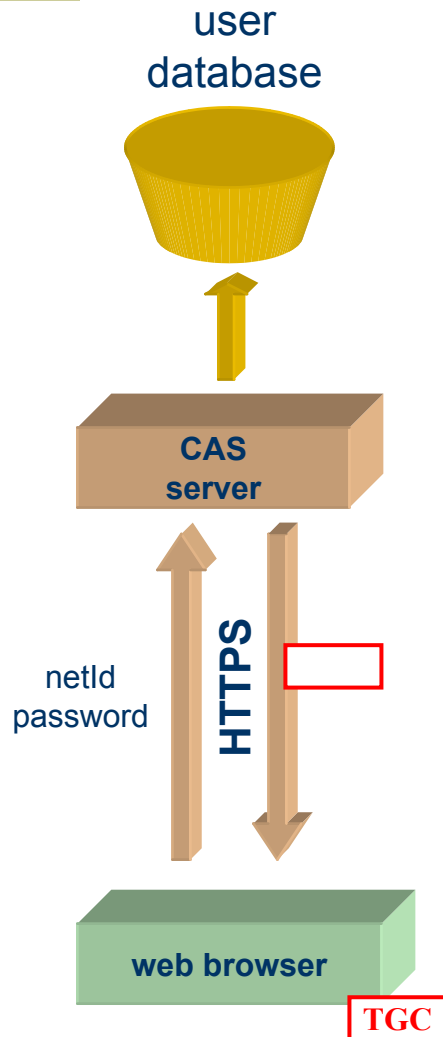
Solution overview

- ◆ SSO Authentication using CAS (Central Authentication Service)
 - Advantages
 - Behavior
 - Limits
 - Setting up CAS at Fecit
- ◆ Authorization using user groups and filters based on url patterns
 - Authorization problematic
 - Proposed solution
 - Pros and Cons

CAS: advantages

- Security
 - Password is never transmitted to applications
 - Opaque tickets are used
- N-tier installations
 - Without transmitting any password
- Portability (client libraries)
 - Java, Perl, JSP, ASP, PHP, PL/SQL, Apache and PAM modules
- Permanence
 - Developed and maintained by Yale University
 - World-wide used (mainly Universities)
- Open source
- Allows “proxy” authentication for Web portals
- Works with existing authentication infrastructures, such as Kerberos

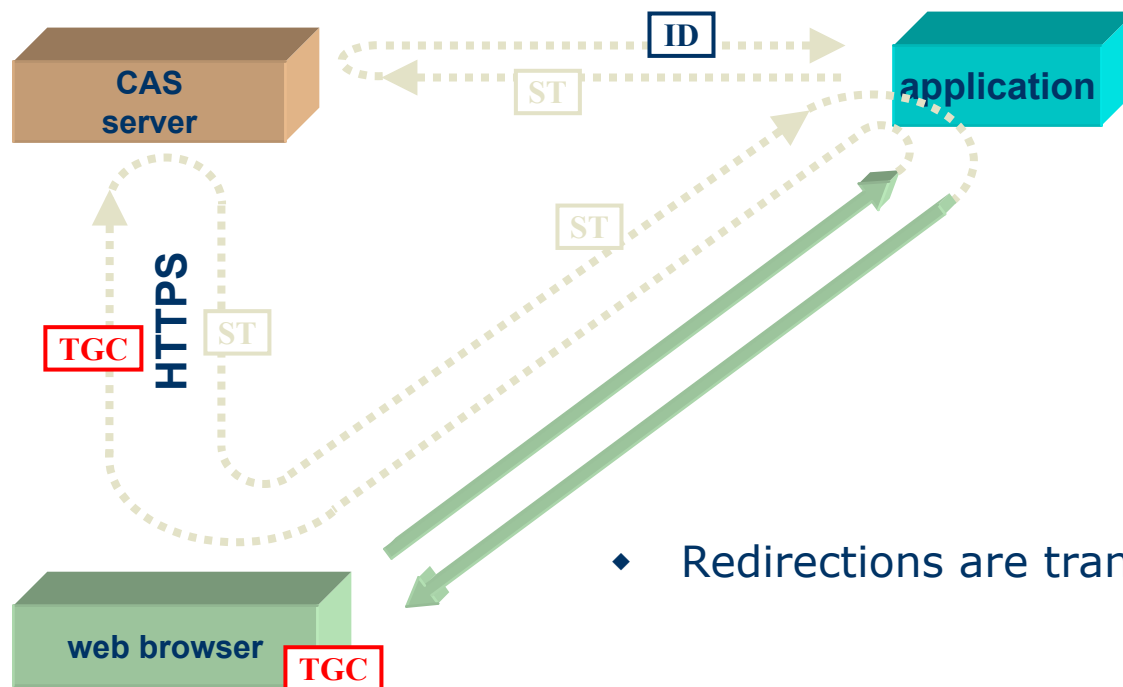
User authentication



- TGC: Ticket Granting Cookie
 - User's passport to the CAS server
 - Private and protected cookie
 - Opaque re-playable ticket

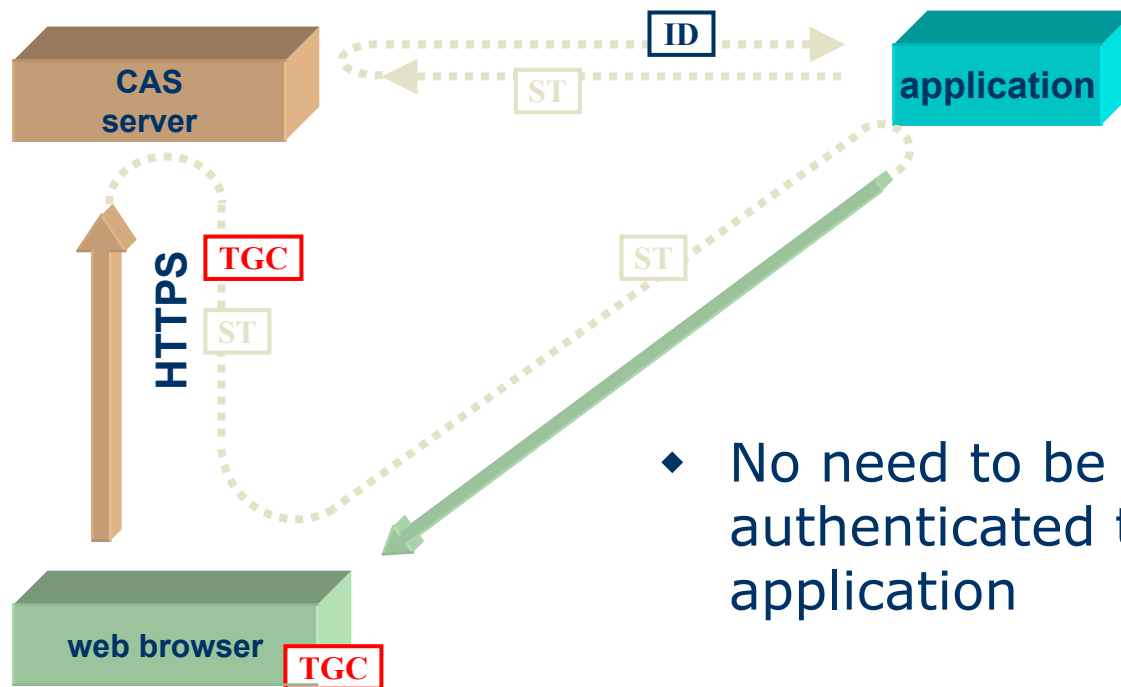
Accessing an application after authentication

- ST: Service Ticket
 - Browser's passport to the CAS client (application)
 - Opaque and non re-playable ticket
 - Very limited validity (a few seconds)



- ◆ Redirections are transparent to users

Accessing an application without authentication

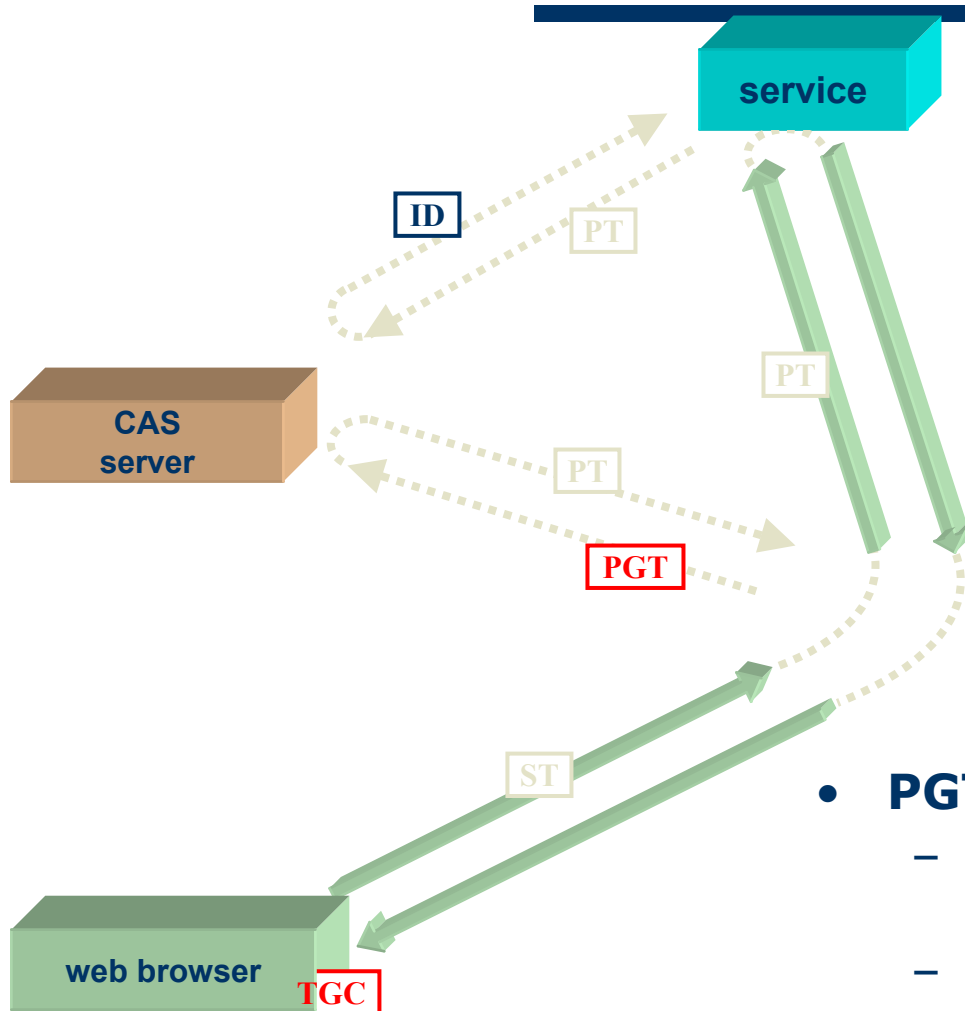


- ◆ No need to be previously authenticated to access an application

Authenticating users with CAS

- ◆ No authentication implementation in CAS
 - Left to administrators, developers...
- ◆ ESUP-Portail CAS Generic Handler:
 - Set of authentication implementations for CAS:
 - LDAP directory
 - Database
 - NIS domain
 - X509 certificates
 - Kerberos domain
 - Windows NT domain
 - Flat files (currently in use)
 - Maintained by ESUP-Portail, active
 - Open-source

N-tier installations



◆ **PT : Proxy Ticket**

- Application's passport for a user to a tier service
- Opaque and non re-playable ticket
- Very limited validity

● **PGT: Proxy Granting Ticket**

- Application's passport for a user to the CAS server
- Opaque and re-playable ticket

CAS: limits

- ◆ CAS deals with authentication, not authorization
- ◆ No redundancy
 - No native load-balancing (but low load)
 - No fault-tolerance (but very good reliability)
- ◆ No Single Sign-Off
- ◆ Very poor documentation (if any...)

Our CAS setup

- ◆ Hard to make it work
 - Poor documentation
 - Tricky issues (at least for me)
 - Self-signed certificates rejected by CAS
 - Didn't manage to get Tomcat working correctly with CAS on Apache
- ◆ Two hosts at Toulouse:
 - Host for CAS authentication: `prismaus.fecit.fr`
 - Host for Web services: `prism.fecit.fr`
 - Each runs Tomcat 5 as a standalone web-server
- ◆ File based authentication

Authorization: background

- ◆ CAS doesn't support Authorization nor carries user attributes
- ◆ CAS provides APIs for user authentication
- ◆ No clue about the structure of authorization data of current or future PRISM application

Authorization: problematic

- ◆ What a user is allowed to do with a specific application is strongly application dependant
 - Options:
 - ◆ Application stores its own authorization data
 - ◆ Authorization system stores authorization data for the application
 - In all cases, only the application can interpret its own authorization data
- ◆ What static resources a specific user has access to
 - Resources: files identified by url
 - Authorized resources: url patterns

Authorization: problematic (cont)

- ◆ Group of users have **roughly** the same authorization data
- ◆ Granularity of authorization data
 - Raw means simple but rigid
 - Fine means complex but flexible
 - User granularity: groups vs individuals
 - Authorization granularity: application access vs specific action with application

Authorization: solution

- ◆ Fine authorization
 - Too application dependant
 - Authorization system handles **raw** authorization
 - Applications handles **fine** authorization based on their own user data
 - Applications have to stores their own authorization data:
 - ◆ Storing authorization data for applications can help but can't work in every cases
- ◆ Fine user authorization administration:
 - Too much administration load
 - Intuitive Idea: authorization is based on the **role** of a user in an organisation.

Authorization solution (cont)

- ◆ Tomcat solution: realms
 - Authorization data: each user has a set of **roles**
 - Tomcat authorization based on Tomcat authentication
 - Idea: replace Tomcat authentication by CAS authentication, reuse Tomcat authorization and administration tools
 - Didn't manage to make it work (currently)

Authorization solution (cont)

- ◆ If only CAS could carry user attributes...
 - Then CAS could carry user's roles
- ◆ We can fake it by extending user name semantic
 - Group = set of user's roles
 - An user belongs to a single group (like unix)
 - Formated user name:
 - [group]'.'[username]
 - Ex: admin.jean, devel.xavier, ghest.claes...

Authorization: responsibilities

- ◆ Account administration:
 - Admins manages user database on authentication host
 - formatted user name and password for each user:
 - ◆ Ex: line in a file based user database:
`devel.xavier:passwd`
- ◆ Authorization data:
 - Developers manages authorization data for groups and applications
 - Special Filter: `PRISMAuthorizationFilter`
 - Authorization data:
 - For each group and for each web application, pattern of authorized URLs.

Authorization solution: pros and cons

◆ PROS:

- Convenient administration:
 - Simple
 - Authentication and authorization data on the same site
- Simple authorization data based on url patterns
- Straightforward to know users belonging
- Close to widely accepted (but limited) authorization solutions (UNIX groups and Tomcat authorization)

◆ CONS:

- « Groups » lacks of flexibility, to counterweight this:
 - Some PRISM user may need more than one account
 - Developers may need to define a lot of groups
- Users belonging is public