Update on the Vulnerability Assessment Effort

Elisa Heymann

Computer Architecture and
Operating Systems Department
Universitat Autònoma de Barcelona

Elisa.Heymann@uab.es

EMI: Software to Assess

- gLite
  - Argus 1.2
  - gLexec 0.8
  - VOMS Core
  - CREAM: Computing Resource Execution And Management
  - WMS: Workload Management System

EMI: Software to Assess

- Unicore
  - TSI: The Target System Interface
  - Gateway

Current Status

- VOMS Admin 2.0.18: done!

  - Java, JSP y javascript
  - 38 KLOC
  - 2.5 months of work

Current Status

- VOMS Admin 2.0.18: done
  - 2 Critical vulnerabilities
    - Cross Site Request Forgery attacks => an attacker will be able to execute arbitrary VOMS Admin actions.
    - Persistent Cross Site Scripting vulnerabilities => non-privileged users can store javascript code in the database, which will be executed by other users.
  - 2 Non-critical vulnerabilities
    - Fields of the web interface that display information about the users certificates are vulnerable to Persistent Cross Site Scripting vulnerabilities.
    - Reflected Cross Site Scripting vulnerabilities => non-privileged users can submit javascript code to VOMS Admin, and this code is reflected back to the user browser.

Current Status

- gLexec 0.8: done!
  - C
  - 13 KLOC
  - 5 months of work

Current Status

- gLexec 0.8: done!
  - 3 Critical vulnerabilities
    - Lack of cleanup of jobs => allows a prior user of the uid to attack the current user's jobs and files.
    - Reuse of local uids => attack another job running later.
  - 1 Non-critical vulnerabilities
    - A job can prevent the job completed log record from being written to the glexec log.

Current Status

- Argus 1.2: done!
  - Java, C, scripting languages
  - 42 KLOC
  - 5 months of work

- No vulnerabilities found.
- Report on what was assessed and why Argus worked fine.

Were are we now

- VOMS Core 2.0.2: Just started
  - Vincenzo Ciaschini & Valerio Venturi
  - C: 30753
  - C++: 10138
  - exp: 7955
  - java: 7754
- Started: May 2011
- Expected duration: 6 months

Vulnerability Assessment@EMI

Apply FPVA to relevant EMI Middleware

- Assess the SW
- Generate vulnerability reports

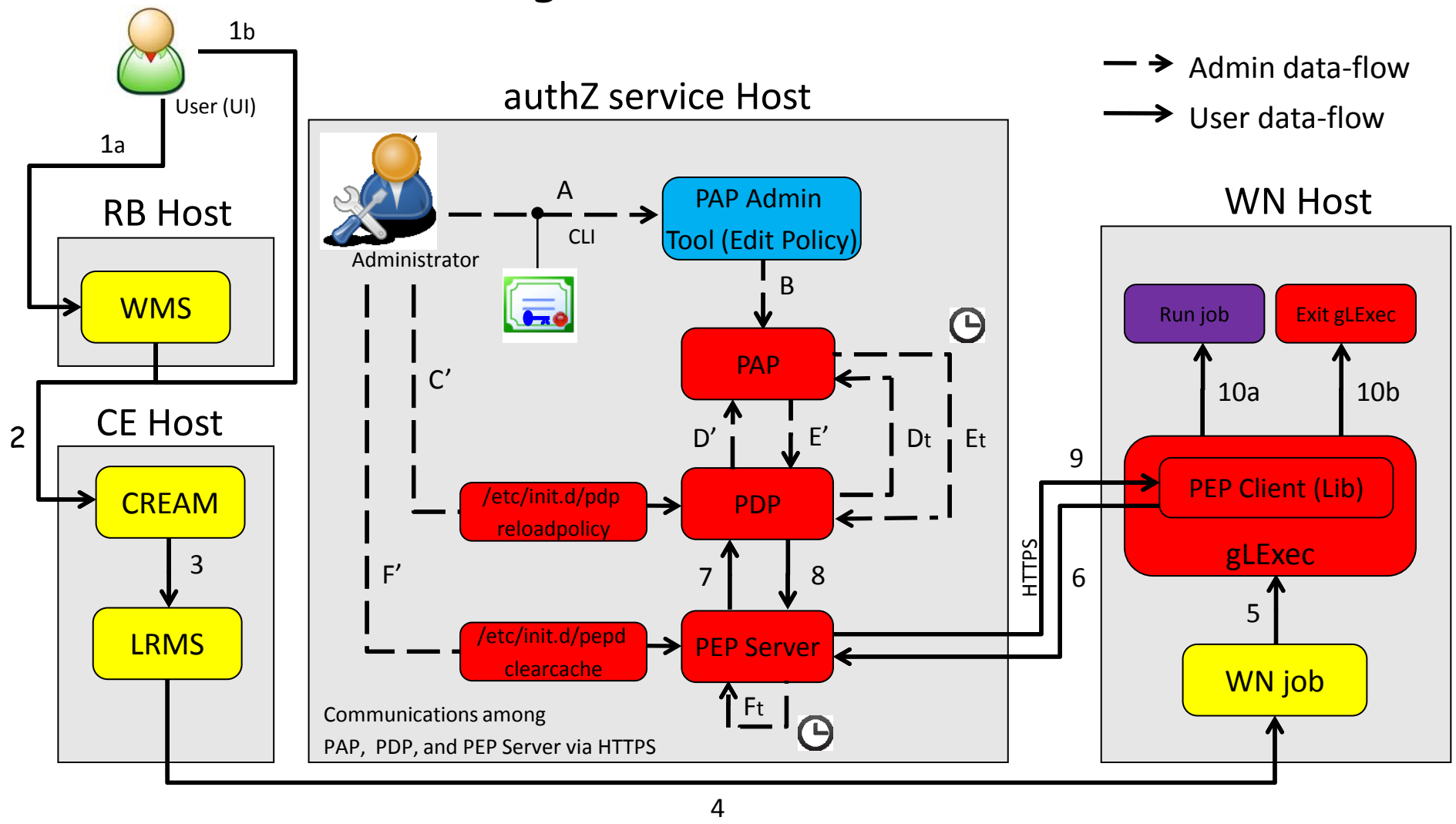
  A vulnerability is considered only when we produce an exploit for it.

First Principles Vulnerability Assessment
Understanding the System

Step 1: Architectural Analysis

- Functionality and structure of the system, major components (modules, threads, processes), communication channels
- Interactions among components and with users

Argus 1.2 Architecture



PAP (Policy Administration Point) → Manage Policies.

PDP (Policy Decision Point) → Evaluate Authorization Requests.

PEP (Policy Enforcement Point) → Process Client Requests and Responses.

OS privileges

- user
- batch user
- root
- External Component
- Administrator & root

First Principles Vulnerability Assessment
Understanding the System

Step 2: Resource Identification

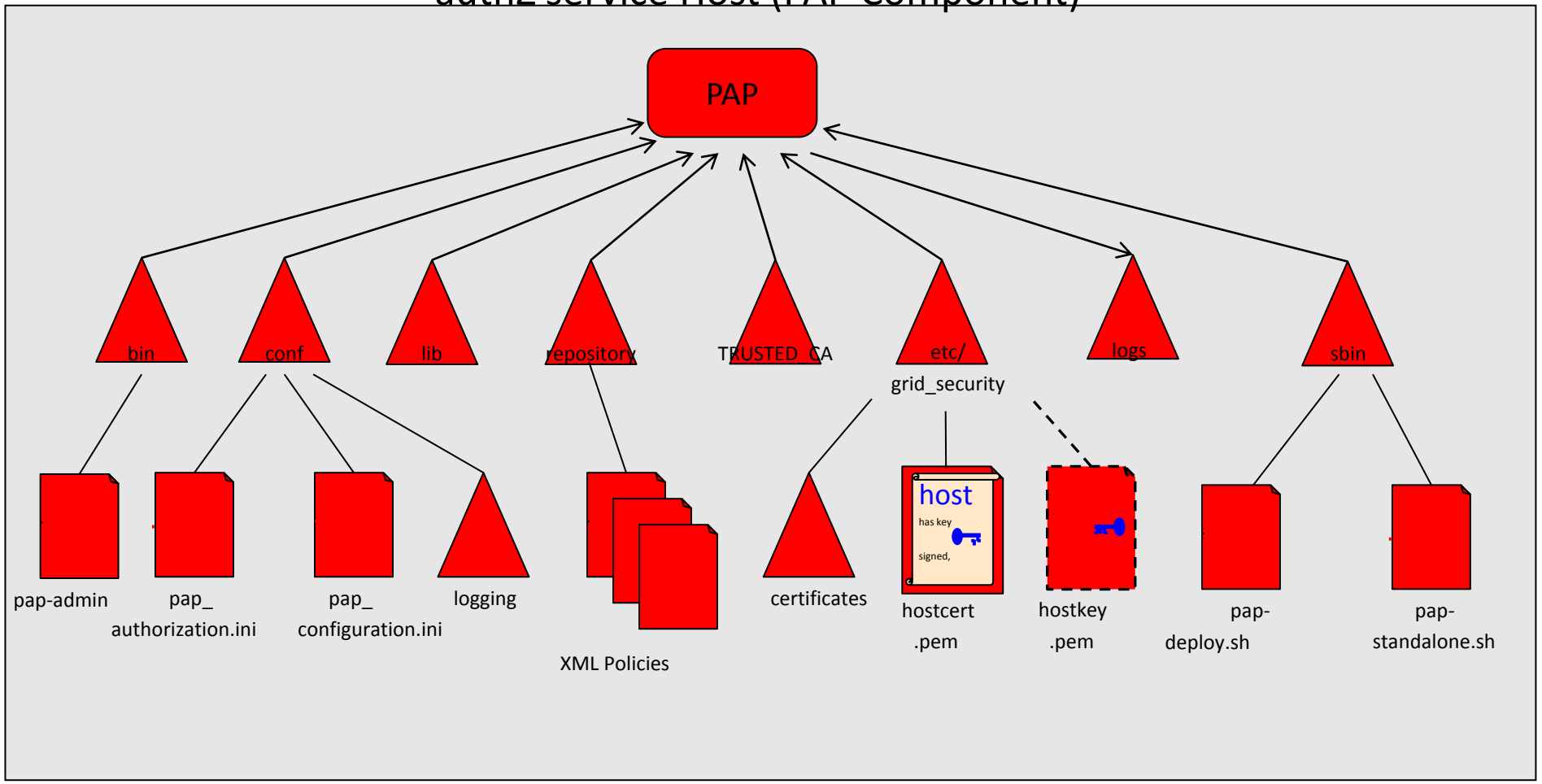
- Key resources accessed by each component
- Operations allowed on those resources

Step 3: Trust & Privilege Analysis

- How components are protected and who can access them
- Privilege level at which each component runs
- Trust delegation

Argus 1.2 Resources

authZ service Host (PAP Component)



Readable

----- Owner

—— World

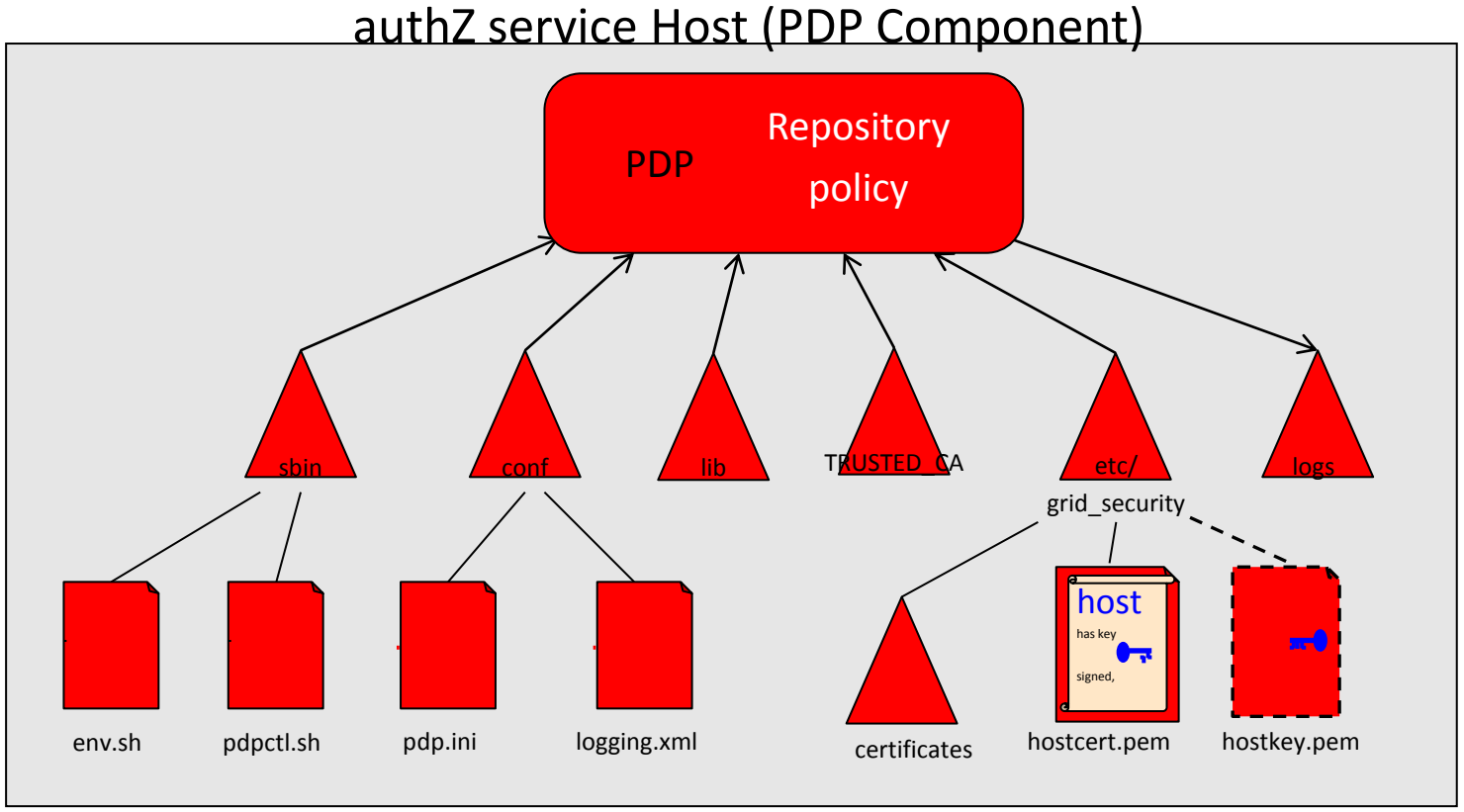
OS privileges

user batch user

root External Component

Administrator & root

Argus 1.2 Resources



Readable

----- Owner

—— World

OS privileges

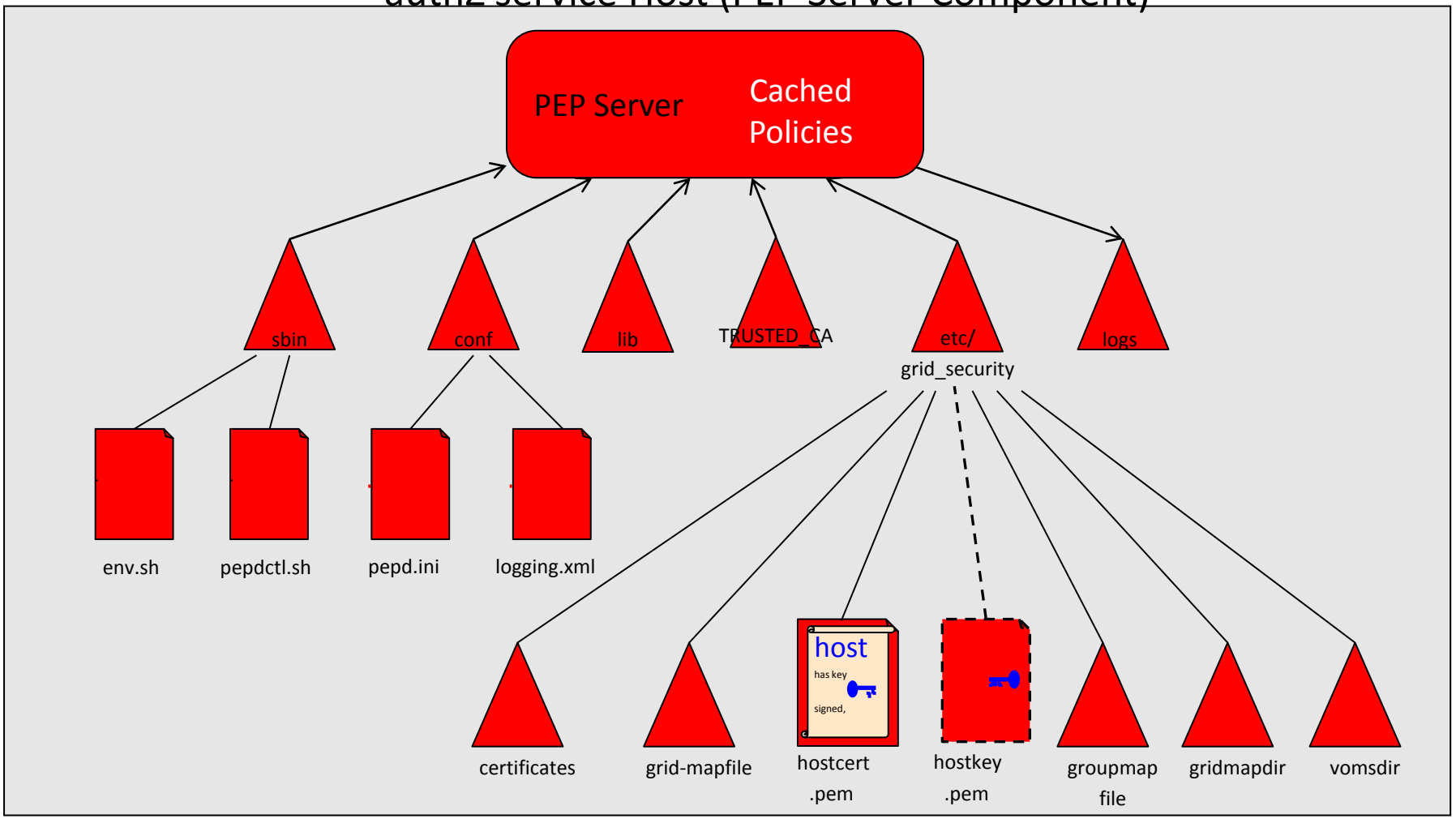
user batch user

root External Component

Administrator & root

Argus 1.2 Resources

authZ service Host (PEP Server Component)



Readable

----- Owner

—— World

OS privileges

user

batch user

root

External Component

Administrator & root

First Principles Vulnerability Assessment
Search for Vulnerabilities

Step 4: Component Evaluation

- Examine critical components in depth
- Guide search using:
  - Diagrams from steps 1-3
  - Knowledge of vulnerabilities
- Helped by Automated scanning tools

First Principles Vulnerability Assessment
Taking Actions

Step 5: Dissemination of Results

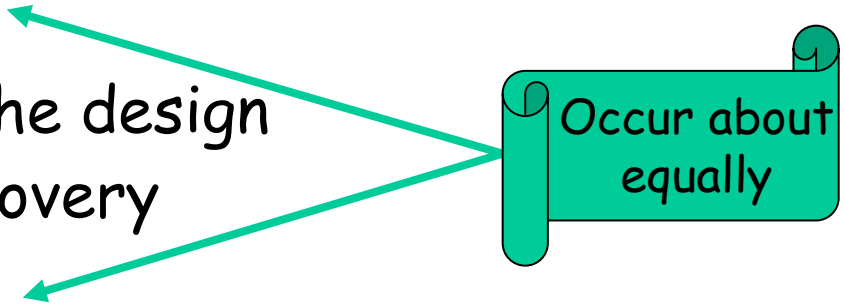
- Report vulnerabilities
- Interaction with developers
- Disclosure of vulnerabilities

Vulnerability Report

- One report per vulnerability
- Provide enough information for developers to reproduce and suggest mitigations
- Written so that a few sections can be removed and the abstracted report is still useful to users without revealing too much information to easily create an attack.

Categories of Vulnerabilities

- Design Flaws
  - Problems inherent in the design
  - Hard to automate discovery
- Implementation Bugs
  - Improper use of the programming language, or of a library API
  - Localized in the code
- Operational vulnerabilities
  - Configuration or environment
- Social Engineering
  - Valid users tricked into attacking



How do You Respond?

- Identify a team member to handle vulnerability reports.
- Develop a remediation strategy:
  - Study the vulnerability report.
  - Use your knowledge of the system to try to identify other places in the code where this might exist.
  - Study the suggested remdiation and formulate your response.
  - Get feedback from the assessment team on your fix – very important for the first few vulnerabilities.
- Develop a security patch release mechanism.
  - This mechanism must be separate from your release feature/upgrade releases.
  - You may have to target patches for more than one version.

How do You Respond?

Develop a notification strategy:

- What will you tell and when?
- Users are nervous during the first reports, but then become your biggest fans.
- Often a staged process:
  1. Announce the vulnerability, without details at the time you release the patch.
  2. Release full details after the user community has had a chance to update, perhaps 6-12 months later.
- Open source makes this more complicated!

  The first release of the patch reveals the details of the vulnerability.