VOMS Core Vulnerability Assessment*

Manuel Brugnoli and Elisa Heymann
Universitat Autònoma de Barcelona
February, 2012

Introduction

Virtual Organization Membership Service (VOMS) serves as a central repository for user authorization information, providing support for sorting users into a general group hierarchy, keeping track of their roles, etc.

This report presents the results of the vulnerability assessment of VOMS Core [1], version 2.0.2, as part of a Quality Control process within EMI. For this task we have used the First Principles Vulnerability Assessment (FPVA) methodology [2] proposed by the University of Wisconsin and Universitat Autònoma de Barcelona Middleware Security and Testing Group.

Although VOMS is divided in VOMS Core and VOMS Admin components, in this report only is reported the results of VOMS Core component, given that VOMS Admin was previously evaluated by our group.

In our evaluation of VOMS Core we did not find any serious security issues. Only a Denial of Service vulnerability was found, and was caused by a lack of limits on the number of simultaneous connections from clients that can result in reduced availability of the VOMS server.

This report is structured as follows. The first section presents the architectural, resource and privilege analysis of VOMS Core. The component evaluation of VOMS Core is discussed in the second section, and the final section provides the results of the vulnerability assessment performed.

Architectural, Resource and Privilege Analysis

In this section, we describe the steps of the FPVA methodology as applied to VOMS Core. We also show the resulting diagrams for each step.

Architectural Analysis

Figure 1 shows the generic architecture of VOMS. VOMS is divided in two main components, VOMS Core and VOMS Admin.

* This research was funded in part by NATO grant ICS.MD.CLG 984138 and the European Commission (contract number: RI-261323), through the EGI-InSPIRE (Integrated Sustainable Pan-European Infrastructure for Researchers in Europe) project.

VOMS Core is written mainly in C++. Its main functionality is to listen for user requests and create attribute certificates. VOMS Core follows a client/server architecture, and consists of:

- a server (VOMS daemon) that receives requests from a VOMS client and returns information about the user
- the VOMS client (`voms-proxy-init`) that contacts the standalone *voms daemon* process that queries the authorization database and generates the actual VOMS attribute certificates
- locally-executed utilities:
    - `voms-proxy-info`: Once a VOMS proxy certificate has been created, this command allows the user to retrieve several information from it.
    - `voms-proxy-destroy`: this command erases an existing VOMS proxy certificate from the system.
    - `voms-proxy-list`: this command displays a list of all available attributes from a specified proxy server. This information can also be obtained using the `list` option of the `voms-proxy-init` command (i.e. `voms-proxy-init -voms VO_NAME -list`).
    - `voms-proxy-fake`: this command generates a proxy containing arbitrary attributes without contacting the VOMS server. It is intended as a tool for testing and debugging software.

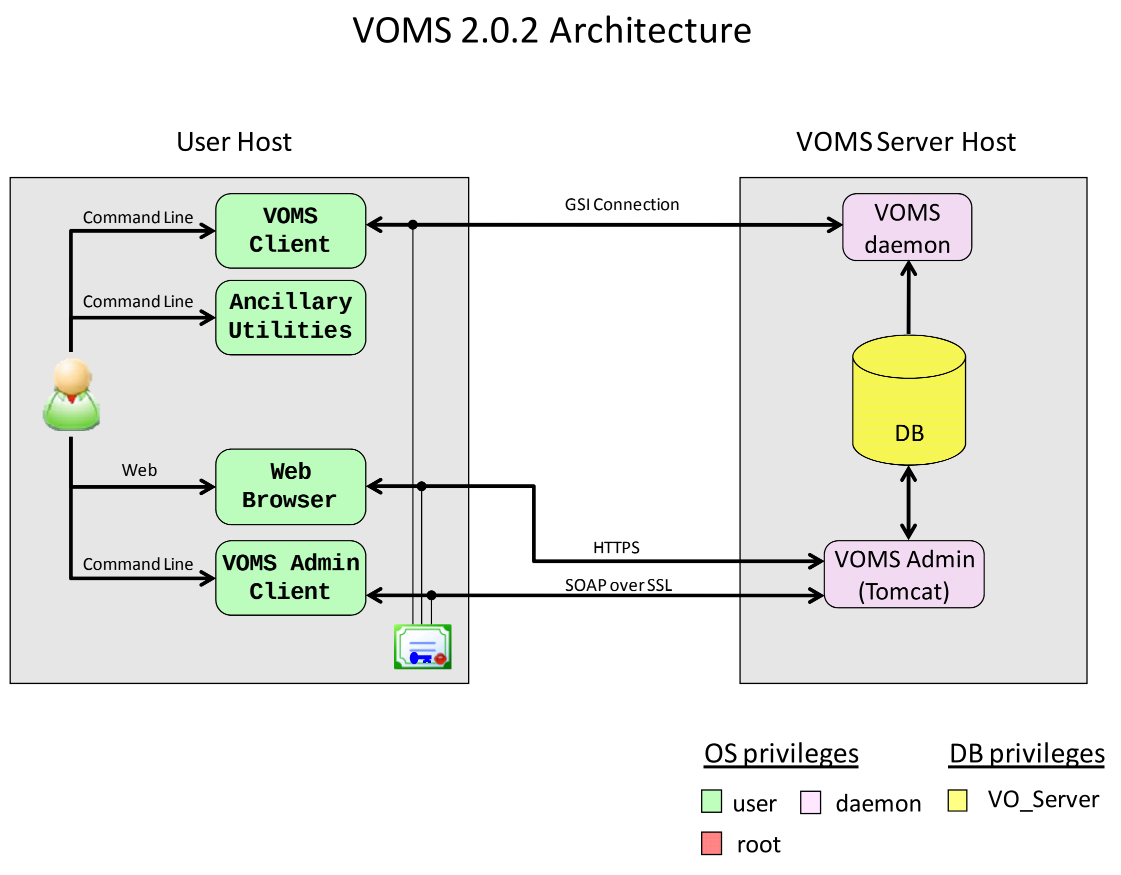

Figure 1. VOMS Architecture Diagram.

As shown in Figure 1, to connect and authenticate to the VOMS daemon, users must have a valid end entity certificate that is signed by a trusted CA, and has not been revoked. This certificate is used to sign a time limited proxy certificate that is used for authentication to the VOMS daemon. To gain access to the VOMS daemon services the client software uses the authentication proxy certificate to negotiate a SSL connection using mutual authentication between the client and the VOMS daemon.

On the other hand, the VOMS Admin component consists of a VOMS Admin server, which is a Java application that runs under Tomcat; a web user interface, which is used by users for daily administration tasks; and a SOAP interface for remote clients, called `voms-admin` that can perform most of the main actions using a SOAP interface to VOMS Admin.

The VOMS database stores the membership of the virtual organization, their roles and attributes, and data only used by VOMS Admin. The VOMS daemon only reads the database, while VOMS Admin component both reads and writes the database.

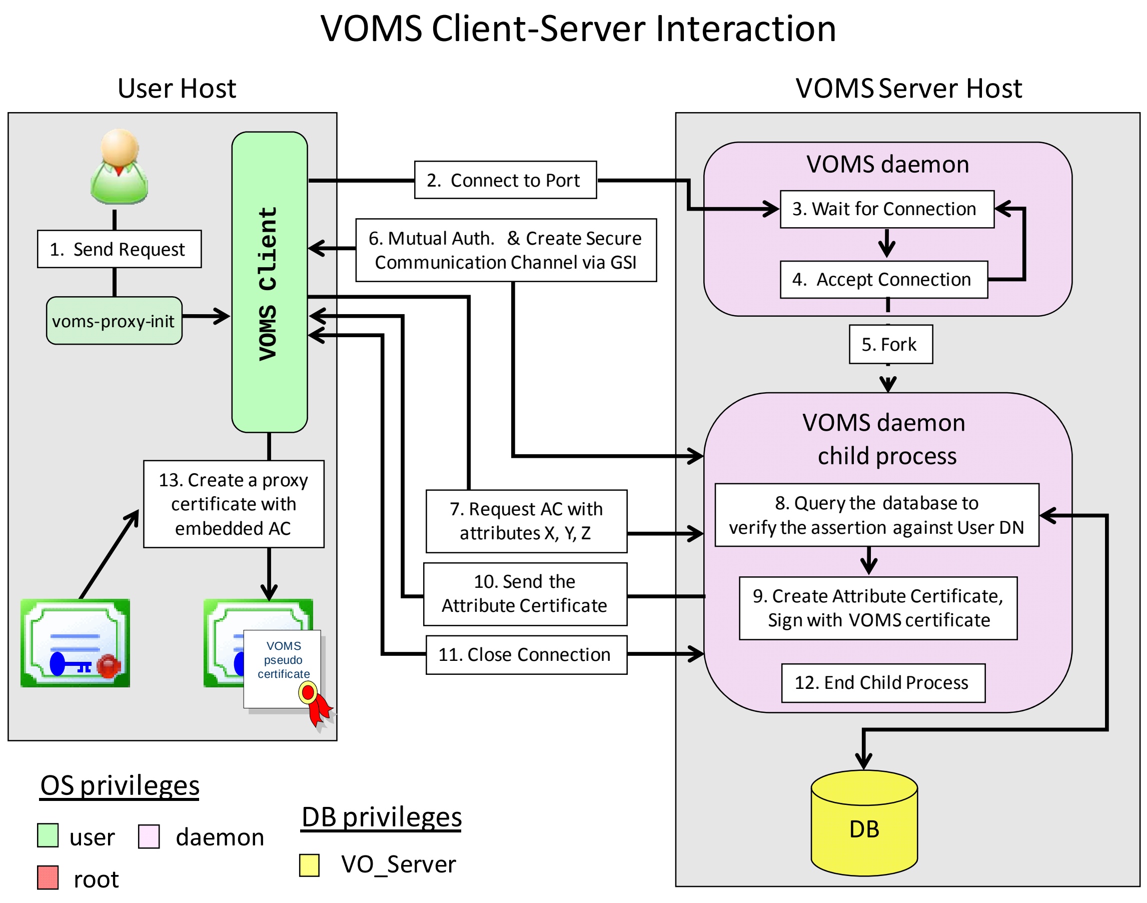
VOMS Client-Server Interaction
User Host
VOMS Server Host
VOMS daemon
2. Connect to Port
3. Wait for Connection
1. Send Request
6. Mutual Auth. & Create Secure Communication Channel via GSI
4. Accept Connection
voms-proxy-init
VOMS Client
5. Fork
VOMS daemon child process
13. Create a proxy certificate with embedded AC
7. Request AC with attributes X, Y, Z
8. Query the database to verify the assertion against User DN
10. Send the Attribute Certificate
9. Create Attribute Certificate, Sign with VOMS certificate
VOMS pseudo certificate
11. Close Connection
12. End Child Process
OS privileges
🟩 user
⬜ daemon
🟥 root
DB privileges
🟨 VO_Server
DB

Figure 2. VOMS Client-Server Interaction Diagram.

All interactions between the VOMS server and a VOMS client follow a pattern shown in Figure 2. To begin the client-server interaction, the client process makes a connection to the VOMS daemon. When the connection is made, the VOMS server uses a forking server model to handle all requests, so a forked copy of the VOMS daemon is created that handles the interaction with this connection. Next, the VOMS daemon checks the user's authentication certificate DN for authorization, and that the requested attributes are valid. If authorized, the VOMS daemon returns an attribute certificate binding the user's DN with the requested attributes, and the client creates a new proxy certificate with the returned attribute certificate embedded. After servicing the request, the forked VOMS daemon exits.

The above sequence of steps may be repeated any number of times. The forked server model allows concurrent execution of request, and isolates requests from each other.

Resource Analysis

The main resource that the VOMS daemon manages is the VOMS database, which contains all the information about users in a VO with their roles and attributes. The database also contains data used by VOMS Admin. VOMS can use an Oracle or MySQL database. In our assessment we used a MySQL database.

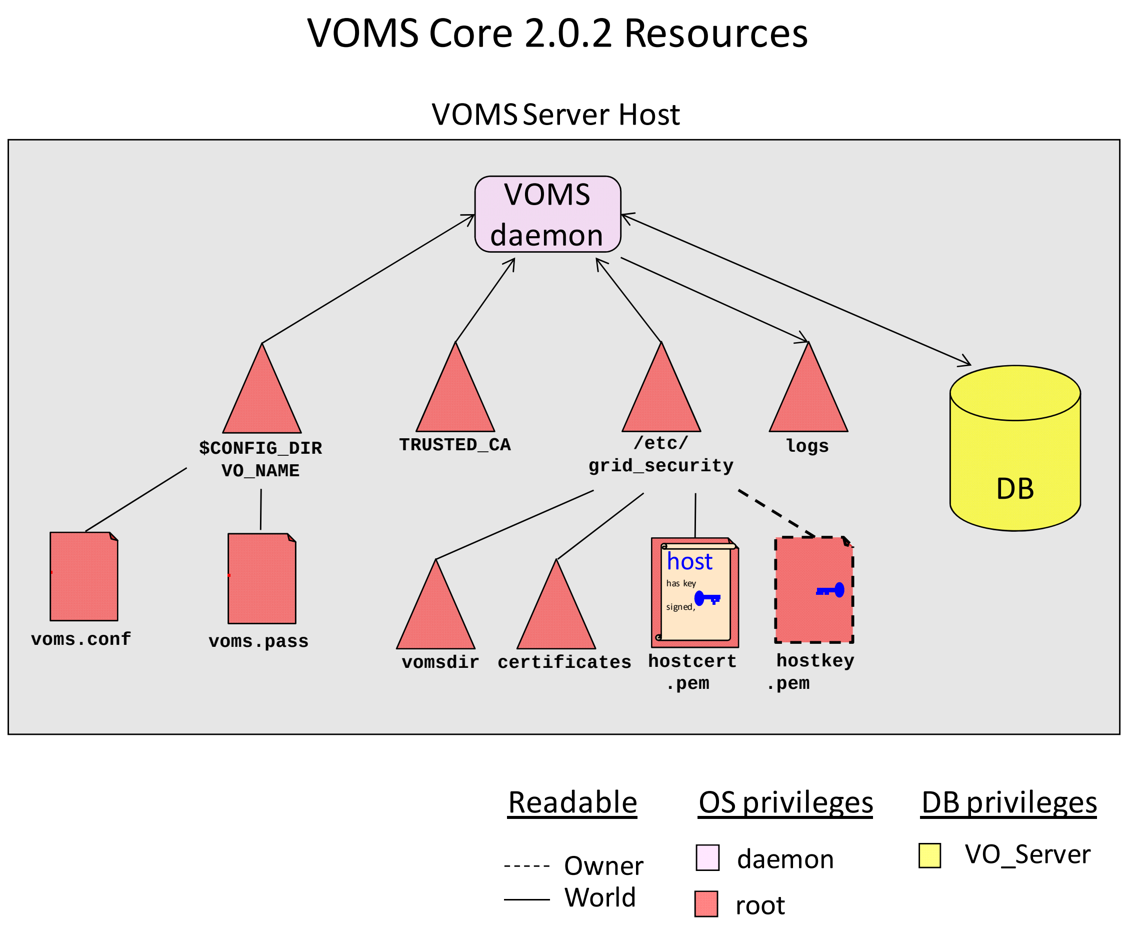

Figure 3. VOMS Core Resources Diagram (VOMS daemon).

As shown in Figure 3, the VOMS daemon also uses other files such as the TRUSTED_CA directory that contains files describing all the trusted Certificate Authorities (CA) including the CA certificates and signing policy files of the CAs trusted by the Grid Security Infrastructure (GSI). The */etc/grid-security* directory contains the X.509 host certificate and private key. The CONFIG_DIR VO_NAME directory contains the configuration files of VOMS, including a file with the database user and password. Finally, the log files are used to keep track of the activity performed by VOMS Core.

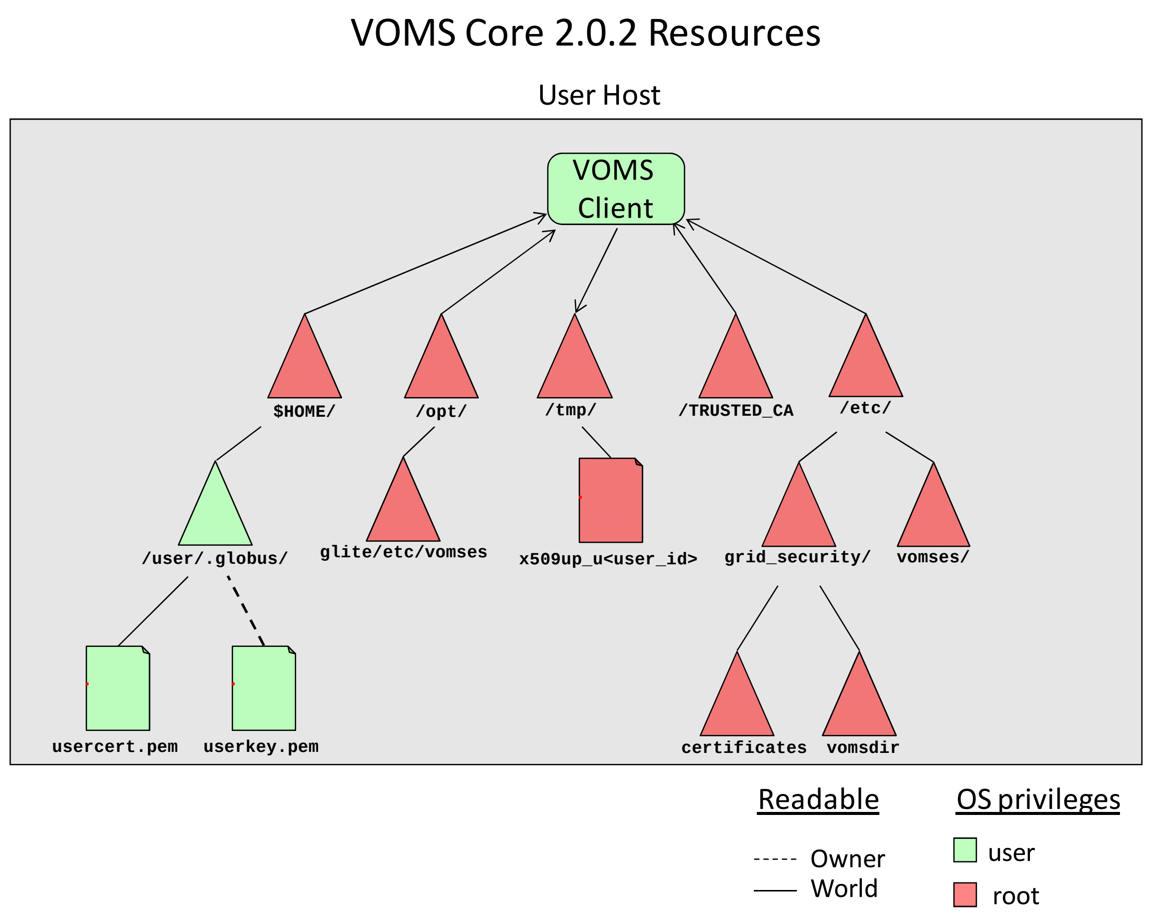

Figure 4. VOMS Core Resources Diagram (VOMS Client).

On the client side, as shown in Figure 4, the main resource that the VOMS Client manages is the user certificate/key, called respectively *usercert* and *userkey*. Both PKCS12 and PEM formatted credentials are acceptable.

Privilege Analysis

The VOMS server runs under root privileges in the operating system and the VOMS client runs with the same privileges as the user who is executing it.

The proxy certificate file generated by VOMS Client is stored by default in the */tmp/* directory. The proxy file is user owned by user executing the client program, and has permissions such that only that user can read, write and delete the file. Also, this exposure is somewhat muted because proxy certificates should have a short life and the credential used to generate the proxy is not at risk.

Component Analysis

In this section we explain the different tests carried out during the vulnerability assessment of VOMS Core. In summary, the design of VOMS Core seemed secure and did not contain obvious serious vulnerabilities. The problems that we did find were more minor implementation flaws, and these flaws are described in this section.

Resource permissions

Due to the sensitive information passing through a VOMS server, it is intended to be run on highly secured host with limited local user access and other services. This should reduce the chances of an attack from another account on the machine.

We checked the permissions of files that have a high security value, including the certificate private keys, and the database configuration files. The permissions of these files appeared to be correct.

Denial of Service Attacks

A common problem in client-server architectures is dealing with denial of service (DOS) attacks. The only vulnerability discovered a Denial of Service vulnerability that was reported to the VOMS developers.

This vulnerability is caused by lack of limits on the number of simultaneous connections. Full details about this were reported in the vulnerability report VOMS-CORE-2011-0001.

User privileges

Since the client commands run with the privilege of the user, and the user initiates the use of the command, an attacker has limited options in carrying out an attack through the command line interface. We did not find privilege problems in the client commands.

On the server side, we found that *voms daemon* runs with root operating system privileges in order to access the resources which are only readable by root, such as *hostkey.pem*. Thus, we have checked the source code looking for flaws that may compromise the server and no privilege problems were found.

Check for dangerous functions

We checked for the use of functions that commonly result in security problems, such as *system* or *exec* functions. No vulnerabilities related to dangerous functions were found.

Authentication Issues

As explained in the Architecture Analysis, mutual authentication is performed between the client and server. This design makes the system quite strong, and reduces many possible threats.

Network Layer Security

Data sent over a network is both susceptible to eaves-dropping and to modification if precautions are not taken. The VOMS server creates a secure communication channel via Globus GSI with the VOMS Clients to handle the client requests.

This encrypted channel provides strong end-to-end data encryption and integrity. We considered these mechanisms and encryption secure, and we did not perform other tests at this regard.

Injection Attacks

We have checked for the possibility of injection attacks in the VOMS command line client in two ways.

Firstly by checking the source code to ensure VOMS correctly parses and checks the arguments passed through the command line. Secondly, through the command line by performing injection attacks using special elements, such as sequence of bytes, characters, or words that is used to separate different portions of data within a particular representation or language. Appropriate parsing is performed to protect against command injection vulnerabilities and no such vulnerabilities were found.

Buffer overflows

As VOMS Core is written in C/C++, we evaluated several potential buffer overflow problems in key parts of the code. However, we did not detected any dangerous behaviour.

Results and Recommendations

After completing the check described in this document we have found only a vulnerability caused by lack of limits on the number of simultaneous connections in VOMS Core 2.0.2.

We did not discover any serious security problems in VOMS Core 2.0.2, nor did we see any security problem with the architecture and implementation. Some of the different features that made VOMS Core strong and secure are the following:

- The attack surface in VOMS Core is very small. It is limited to the component that receives requests from a VOMS client and returns information about the user.
- VOMS Core correctly parses and checks the arguments sent from the client.
- The VOMS server uses a forking server model to handle all requests from VOMS clients. The forked server model provides isolation from faults, and contamination of the process that could affect future requests.

- The recommended operational configuration of a VOMS server node is a highly secured host with limited local user access and other services. This reduces the chances of an attack from another account on the machine.
- All communication between the VOMS server and VOMS clients is secure using secure communication channel via GSI and a mutual authentication is performed. This prevents exposure of sensitive data.

References

[1] Virtual Organization Membership Service (VOMS), https://twiki.cnaf.infn.it/twiki/bin/view/VOMS/

[2] James A. Kupsch, Barton P. Miller, Eduardo César, and Elisa Heymann. *First Principles Vulnerability Assessment*, 2010 ACM Cloud Computing Security Workshop (CCSW), Chicago, IL, October 2010. http://www.cs.wisc.edu/mist/papers/ccsw12sp-kupsch.pdf