Akademia Górniczo – Hutnicza
im. Stanisława Staszica
w Krakowie
Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Katedra Informatyki

Jan Meizner

Bezpieczeństwo w systemie Wirtualnego Laboratorium

Praca magisterska

Kierunek: Informatyka
Specjalność: Systemy rozproszone i sieci komputerowe

Promotor:
dr inż. Marian Bubak

Konsultacja:
dr inż. Maciej Malawski

Nr albumu: 120564

Kraków 2009

Oświadczenie autora

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Jan Meizner

AGH University of Science and Technology
in Kraków

Faculty of Electrical Engineering, Automatics, Computer Science
and Electronics

Institute of Computer Science

Jan Meizner

Security in Virtual Laboratory System

Thesis

Major: Computer Science
Specialization: Distributed Systems and Computer Networks

Supervisor:
Dr. Marian Bubak

Consultancy:
Dr. Maciej Malawski

Album id: 120564

Kraków 2009

Oświadczenie autora

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Jan Meizner

Abstract

The thesis presents a work that has been done to provide a flexible security mechanism for the Virtual Laboratory (VL). It was focused on preparation of solutions to cover complex requirements of a non-web based part of the VL. Specifically it needs to provide access to distributed resources for various types of users, who work for many separate organizations. This constraint required the creation of a federated solution that allows each organization to hold separate credential databases, even though their users need access to the whole infrastructure. It also had to allow a seamless integration with other security components, mostly those created for the web-based subset of VL components.

The work described in the thesis provides both theoretical background related to this security solution, as well as detail of the software that has been created. This software is composed of both complete tools (like ShibIdpCliClient, Policy Distribution Point) and components providing security functionality for existing software (ShibIdpClient, MOCCA Shibboleth Authenticator).

After introductory information including presentation of the VL, motivation for the work as well as the goals, existing security solutions including cryptographic algorithms (AES, RSA, Diffie-Helman and SHA), security standards and protocols (PKI, X.509 public key certificates, TLS and SAML) and security frameworks (GSI, Shibboleth, ShibGrid, GridShib and OpenID) were described. Subsequently, all system requirements were analyzed, both directly related to the security as well as others. After that the solution based on the Shibboleth augmented with newly created software for non-web authentication (ShibIdpClient, ShibIdpCliClient) and authorization (MOCCA Shibboleth Authenticator, Policy Distribution Point, it's client and administrator's tool) was presented, it's components design was shown as well as the implementation. Finally solution was successfully validated by performing security audit on critical components, it's performance was evaluated and found to be sufficient, then final conclusions were presented.

Key words

Virtual Laboratory, security, Grid, Shibboleth, federated authentication, user attributes, SAML, threat model

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Dr. Marian Bubak for his invaluable help and support. I would also like to thank Dr. Maciej Malawski for his counsel during the creation of this thesis. Additionally I would like to thank all my colleagues from ACC Cyfronet AGH, with whom I have been working on the ViroLab Project. I also wishes to acknowledge helpful contribution from Tomasz Mikołajczyk, Paweł Płaszczak and Krzysztof Wilk from GridwiseTech as well as Matthias Assel from High Performance Computing Center Stuttgart. Finally, I would like to thank all my academic teachers who guided me through my education at the AGH.

This work has been supported by the European Commission ViroLab Project [1] Grant 027446 and the PL-Grid Project [2] POIG.02.03.00-00-007/08-00.

Contents

List of Figures

List of Tables

Acronyms

AA	Attribute Authority
ACRL	Attribute Certificate Revocation List
AES	Advanced Encryption Standard
ASN.1	Abstract Syntax Notation One
CA	Certificate Authority
CRL	Certificate Revocation List
DB	Database
DES	Data Encryption Standard
EMI	Experiment Management Interface
EPE	Experiment Planning Environment
GSEngine	GridSpace Engine - ViroLab run-time environment
GSI	Grid Security Infrastructure
H2O	middleware platform for building distributed applications
HIV	Human Immunodeficiency Virus
HMAC	Keyed-Hash Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP over TLS (previously SSL)
IDE	Integrated Development Environment
IdP	Identity Provider
IT	Information Technology
ITU	International Telecommunication Union
ITU-T	Telecommunication standardization sector of ITU
JDBC	Java DataBase Connectivity
LDAP	Lightweight Directory Access Protocol
MD5	Message-Digest algorithm 5
MOCCA	CCA compliant framework
PDistP	Policy Distribution Point
PKI	Public Key Infrastructure
RDBMS	Relational Database Management System
RSA	asymmetric cryptographic algorithm invented by R. Rivest, A. Shamir and L. Adleman
SAML	Security Assertion Markup Language
SHA	Secure Hash Algorithm
ShibIdpCliClient	command line interface for ShibIdpClient
ShibIdpClient	library providing non-Web access to a Shibboleth IdP
SOAP	Simple Object Access Protocol
SP	Service Provider
SSL	Secure Sockets Layer
SSO	Single Sign-On
TCP	Transmission Control Protocol
TLS	Transport Layer Security
Tripple-DES	variation of DES
TTP	Trusted Third Party
UI	User Interface
UML	Unified Modeling Language
X.509	ITU-T standard describing Public-key and attribute certificate frameworks

Chapter 1

Introduction

This chapter introduces a notion of IT systems security emphasizing its importance for all types of such systems. Later it describes a Virtual Laboratory on a basis of the ViroLab, showing its properties that are quite unique from a security point of view. Finally it provides a motivation for the work as well as its goals.

1.1. Security of IT Systems

All types of IT systems are potentially vulnerable to various security risks. A number of threats is heavily correlated to a number of people with access to the system. Despite this, even internal systems, completely isolated from the Internet are vulnerable and might be attacked from inside. For example some kind of malware like a virus might be transmitted from the outside world, on an employee's laptop. For that reason, even a system closed to external users must be well patched and constantly monitored for security flaws. In the case of systems that are widely open to general public via the Internet, like ViroLab, it is crucial to protect them against various risk factors both external to the system (from people not being users of the system), as well as the internal ones (from malicious users). In addition to that, ViroLab is a distributed system and it's components may run on multiple nodes. This type of an architecture considerably complicates the security infrastructure as it requires to introduce a secure mechanism of credentials delegation between the nodes. Obviously, an inadequate protection of such delegated credentials would create serious security vulnerabilities in the system. Finally, as ViroLab is formed by multiple separate partners, its security system must be able to grant access to resources supplied by any partner not only to its users but to other partners users as well. To achieve this goal it is required to use a specific type of a security framework, known as the federated security system.

1.2. The ViroLab as an Example of a Virtual Laboratory

This thesis is a description of the work, that has been done to provide security solution for the ViroLab virtual laboratory - software enabling users to develop and run in-silco experiments [3]. Overview of the laboratory is shown in Fig. 1.1. In particular it covers HIV treatment related area aiming for improvement of clinical results by gathering knowledge on subjects like resistance of particular HIV mutations to specific drugs or potentially harmful interactions between various medications used to slow down development of AIDS [4]. This knowledge might be later used by the software to support medical doctors in making decisions about the best course of treatment. This goal requires a large number of users from various institutions and many disciplines like of course computer scientists but also non-IT specialists like virologists or medical doctors. This variety of organizations and users specializations imposes specific requirements on all the components including the security system. In addition to already described need for a federated system, it was necessary to design system that is user-friendly enough for convenient work of all partners, including those whose primary specialization is not related to computer science. The ViroLab has a layered infrastructure. It's core, the GSEngine [5] software, is an example of VL run-time Environment [6] that uses computational services and data sources running on the Grid infrastructure and provides services to the users through dedicated interfaces. The system must ensure security on all the mentioned levels ranging from the infrastructure (secure communication, guidelines for maintainers of system software), through providing security for service layer and the GSEngine itself and finally providing UI for the users.

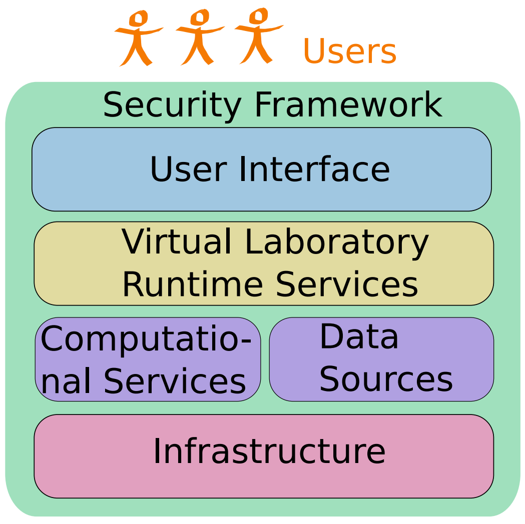


Figure 1.1. Overview of the ViroLab virtual laboratory showing it's layered structure composed of: the Grid infrastructure, various computational and data services, run-time environment and user interface protected by the integrated security framework

1.3. Motivation

As described earlier in this chapter, the ViroLab virtual laboratory requires a unique security infrastructure. The author had the opportunity to be responsible for key security components in the project. The required solution had to on one hand to support complex federated framework for all the partners, and to protect various software components while on another hand to be friendly for potential users, especially domain experts that are not computer specialists. Because of the constrains of ViroLab, and a lack of sufficient alternative solution it was decided to analyze the one already chosen for Web related part the project. The purpose of this activity was to determine which requirements it meets and what kind of customizations were needed. This information could provide a way to design and implement the missing components and to integrate them with existing external ones into a complete security solution for the virtual laboratory.

1.4. Goals of the thesis

Work done in this thesis was focused on creation of secure yet user-friendly security solution that would provide functionality required by virtual laboratory described here. This requires achieving the following goals:

1. analysis of existing security solutions and frameworks,
2. identification of elements that might be useful in creation of the complete solution,
3. creation of a formal threat model for the infrastructure,
4. enumeration system requirements,
5. discussion of the system architecture,
6. design and implementation of following system components: ShibIdpClient, ShibIdpCliClient, MOCCA Shibboleth Authenticator, Policy Distribution Point (PDistP), its client and administrator's panel as described bellow,
7. performing system validation and evaluation.

ShibIdpClient is a library that might be integrated with other external stand alone software (like the EPE [7]) components to provide access to Shibboleth [8] IdP without a need to use any web browser. It also has to be integrated with simple command line interface to create tool (ShibIdpCliClient) enabling users that prefer this type of interface rather then a web portal.

MOCCA Shibboleth Authenticator is an authenticator for H2O/MOCCA [9, 10] supplementing them with a support for a Shibboleth credentials, in order to secure access to MOCCA installations used in the ViroLab.

Policy Distribution Point (PDistP) is an XML-RPC [11] based service used to distribute authorisation policies to dispersed H2O kernels acting as containers for a MOCCA.

MOCCA Policy Distribution Point Client is a tool running on MOCCA nodes that is being used to update policies based on information supplied by the PDistP

Administrator tools for the PDistP is a web based software that enable system administrators to setup central policies for the MOCCA installations.

1.5. Summary

This chapter emphasized need for the solid security solutions for all types of IT system. It also introduced the virtual laboratory and the motivation for complex federated security solution, which defined the goals of the thesis. Issues introduced here are addressed in subsequent chapters as follows: **chapter 2** analyzes existing security solutions like cryptographic algorithms, protocols, standards and frameworks; **chapter 3** focuses on creation of a threat model [12], requirements and discussion of the chosen solution; **chapter 4** presents architecture of the security system; **chapter 5** shows the system design; **chapter 6** is devoted to the aspects of implementation; **chapter 7** presents the validation and evaluation of the solution and **chapter 8** the conclusions and further work.

Chapter 2

Analysis of Security Solutions

This chapter begins with the description of basic cryptographic algorithms which are essential for any IT security measures. Then, it moves to the solutions and standards based on those algorithms, and at the same time being building blocks of the complete security frameworks, which are described in the last section of the chapter.

2.1. Cryptographic algorithms

Algorithms described here that are relevant to the subject of this thesis are divided into five sub-groupings: symmetric, asymmetric, key exchange, hashing and keyed-hash message authentication code.

Symmetric cryptography provides a relatively fast encryption/decryption algorithms, that however require prior knowledge of a shared secret by all parties. Most notable examples are currently obsolete DES and Triple-DES [13] as well as AES [14] that took their place.

Asymmetric cryptography supplies much slower encryption/decryption and signature algorithms. It features use of a public/private key-pair in place of shared secret enabling users to use it without a need to exchange any confidential information. By downloading public key any entity is able to encrypt or verify its owner's signature, but only the owner has a private key that might be used to decrypt or sign the message. These algorithms are frequently used in the process of establishing a shared secret that might be later used for faster encryption/decryption with a help of the symmetric algorithms. A standard example of this type of algorithms is the RSA [15].

Key exchange algorithms might be used to ensure a secure exchange of a shared secret required by symmetric cryptographic algorithms. One of the commonly used examples is a Diffie-Helman [16] algorithm.

Cryptographic hashes are used, both to derive password hashes preventing attacker to decrypt stored passwords, as there are no reverse algorithms, as well as to generate a short message that might be digitally signed to protect the integrity of a larger block of data. Very well known example, but currently obsolete due to it's weakness is MD5 [17]. Currently it has been replaced in newer systems by SHA family algorithms [18] (SHA-1 and now sometimes also SHA-2).

Keyed-Hash Message Authentication Code (HMAC) is a method of generation Message Authentication Code (MAC), protecting message origin as well as its integrity [19].

Most representative examples of algorithms currently used, and not being obsoleted were chosen and described in more details later in this section.

2.1.1. Advanced Encryption Standard

The Advanced Encryption Standard (AES) is NIST approved [14] version of a block cipher originally known as Rijndael. It operates on 128-bit blocks and 128-bit (AES-128), 192-bit (AES-192) or 256-bit (AES-256) keys. The strength of this algorithm has been confirmed both in original NIST document (which pronounced AES suitable to protect non-classified sensitive information) as well as later by US National Security Agency (as suitable to protect classified information up to SECRET level for AES-128 and TOP SECRET for AES-192 and AES-256) [20].

2.1.2. RSA algorithm

RSA is an asymmetric encryption and signing algorithm originally created by R.L. Rivest, A. Shamir, and L. Adleman . Most recent version of the standard build on this algorithm is published as a PKCS#1 standard [15]. RSA features a pair of keys: a public and a private key.

A public key is used for encrypting data and verifying signatures; it includes a modulus n being a product of 2 large prime numbers p and q (in original algorithms; further standards prove that it is faster but also safe to use more then 2 prime numbers r_1 to r_u where $u \geq 2$ - so called multi-prime RSA) and a public exponent e that satisfies: $3 \leq e \leq n-1$ and $\gcd(LCM(p-1, q-1)) = 1$ or $\gcd(LCM(r_1 - 1, \ldots, r_u - 1)) = 1$ where gcd is a greatest common divider, and LCM least common multiple.

A private key is used for decrypting data and signing it; it includes the modulus n, the same as the public key and a private exponent d. The following condition must be met: $e\dot{d} \equiv 1(mod(LCM(p-1, q-1)))$ or $e\dot{d} \equiv 1(mod(LCM(r_1, \ldots, r_u)))$

This algorithm is capable of performing both encrypting/decrypting of data as well as signing/verifying signatures.

2.1.3. Diffie-Helman Key Exchange

Diffie-Helman Key Exchange [16] algorithm has been created as a method allowing a secure exchange of a secret between communicating parties so that it cannot be eavesdropped by any malicious third parties. This algorithm prevents intercepting the key directly, however it does not ensure the authentication of communicating parties, being vulnerable to the man-in-the-middle attack. This is the reason why it must be accompanied by a solution providing authentication like some kind of signing algorithm (e.g. RSA) and a secure public key distribution method (like a Public Key Infrastructure [21]). Despite that Diffie-Helman Key Exchange algorithm is still not redundant in this scenario as it eliminates a need for direct exchange of a symmetric key (even encrypted one, e.g. by directly encrypting the key with the RSA). In this way this solution provides property known as perfect forward secrecy.

2.1.4. Secure Hash Algorithms

Secure Hash Algorithms are the group of four algorithms (SHA-1 and SHA-2 family of SHA-256, SHA-384 and SHA-512) described in Secure Hash Standard [18]. Those algorithms allow a user to compute a message digest, its fixed length representation that is unique for each message with a very high probability. This low probability of collision (the case in which two different messages have the same digest) allows using this type of algorithms to ensure message integrity, as any change in the message will most likely cause change of a digest.

2.1.5. Keyed-Hash Message Authentication Code

This type of code [19] combines s secret key with a cryptographic hash algorithm (like SHA). This combination ensures both message integrity as well as the authenticity of its source. The RFC 2104 [22] suggests a naming convention depending on a used hash function in form HMAC-`hash_name` (e.g. HMAC-SHA1).

2.1.6. Summary

The algorithms described above play a crucial role in solutions described further in this thesis. Encryption/decryption algorithms are of course required by any secure transport protocol to provide the confidentiality of transmitted data. The signature algorithms and HMAC codes accompanied by the cryptographic hash algorithms are required to provide a method of ensuring authenticity and integrity of the transmission. Finally if the algorithm requires to exchange a common key (like in case of symmetric algorithms) mechanisms providing secure key exchange are mandatory.

2.2. Security standards and protocols

After describing algorithms being the basis for the security solutions, this section aims to show standards and solutions itself that might be considered as building blocks for complete security frameworks described later in this chapter.

2.2.1. Public Key Infrastructure

Public Key Infrastructure (PKI) is a method that allows to establish a mutual trust relationship between communicating parties without a need for any previous contact (like exchange of shared credentials) between them. It is accomplished by introducing so called trusted third party (TTP) [23] that could confirm the authentication information provided by communicating parties with the help of asymmetric cryptography. They might be signed by the TTP with it's private key. This information along with other like validity period and peers public key (useful in further communication to validate its signatures and to encrypt data directed toward them) is usually enclosed in a standard format like X.509 public-key certificate [21]. Additionally, PKI also supports the infrastructure for invalidating compromised or no longer needed certificates with the help of a Certificate Revocation List (CRL). Of course for PKI to function properly TTP certificates must be already known and be trusted by all communicating parties.

2.2.2. Public-key certificates (X.509)

Public-key certificates and Certificate Revocation Lists (CRL) along with attribute certificates, its revocation lists (ACRL) and relevant authentication services are the part of the X.509 ITU-T standard [21]. These particular certificates carry mandatory data required by the PKI infrastructure including owners authentication information and the public key confirmed by Certificate Authority (CA) signature. Public-key certificates are encoded as an ASN.1 [24] binary file. They contain information presented in Tab. 2.1 (digitally signed with the CA's private key).

Field name	**Field description**
version	certificate version - currently integers from 0 (v1) to 2 (v3) are valid; some fields are valid only for specific versions of the certificate
serialNumber	certificate serial number assigned by the issuer
signature	information about the algorithm used to sign a certificate (algorithm) and optionally algorithm parameters (parameters)
issuer	name of the certificate issuer (CA)
validity	sequence of two dates marking a beginning (notBefore) and end (notAfter) of this certificate validity period
subject	name of the certificate subject (owner)
subjectPublicKeyInfo	sequence of AlgorithmIdentifier (containing used algorithm and optional parameters) for subject's public key and the public key itself
issuerUniqueIdentifier	(since v2) optional identifier of the issuer
subjectUniqueIdentifier	(since v2) optional identifier of the subject
extensions	(since v3) optional extensions that might be used to add other information not covered by standard fields like alternate subject names or non critical extensions used by GridShib technology described later in subsection 2.3.4

Table 2.1. Information that is a part of the X.509 public key certificate [21] with descriptions of all certificate fields

2.2.3. Transport Layer Security

A Transport Layer Security (TLS) [25] is a successor of a Secure Sockets Layer (SSL) protocol [26]. The protocol might be used to a establish secure communication over a reliable transport protocol, such as TCP. It provides confidentiality using symmetric cryptography algorithms (like AES) to keep the privacy and the integrity of transmitted data with the help of HMAC codes.

It must not allow the attacker to access or modify the message either by simple eavesdropping or more elaborate methods (e.g. a man-in-the-middle attack).

TLS is a layered protocol, and its lowest Record Protocol is responsible for data fragmentation, optional compression / decompression, encryption / decryption and HMAC calculations. It's functionality is used by following four higher level protocols defined by the standard (and possible other extensions):

1. The handshake protocol - is used to:
   - choose algorithms that might be used during the connection by both sides,
   - exchange of parameters for agreed algorithms,
   - server authentication or optionally mutual authentication of server and client,
   - exchange of keys used for symmetric algorithms during communication; key exchange might be performed either directly by sending generated key encrypted with RSA or with help of Diffie-Helmant key exchange algorithm in unauthenticated version (if authenticity of server is ensured) or additionally authenticate with the help of signing algorithms (RSA).
2. The alert protocol is used to communicate the discovery of an abnormal situation to another side of the connection at any point. Alerts are divided into two groups based on the severity:
   - fatal – in the case of this error the receiver and the sender must immediately close communication and forget all security information exchanged during the failed session,
   - warning – communication could continue normally (sender shouldn't close connection after sending this alert) however receiver might decide to close the connection; in this case it should send it's own fatal alert before closing the connection.
3. The change cipher spec protocol - this message is used to signal switch to the newly generated cipher specification and keys.
4. The application data protocol - this is not TLS sub-protocol, but arbitrary data depending on the third party protocol using TLS (transparent to it).

2.2.4. Security Assertion Markup Language

Security Assertion Markup Language (SAML) [27] is a XML based security assertion standard. Security information might be exchanged between parties providing identity - called Asserting Party (SAML1.1) or Identity Provider (SAML2.0) and the one receiving it - called Relaying Party (SAML1.1) or Service Provider (SAML2.0). The Standard specifies assertions, protocol, bindings and profiles.

The Assertions carry statements provided by SAML authority such as authentication information, attributes or authorization decision.

SAML protocol is simple a request/response mechanism used in assertions exchange. Party that wants to get an information sends a `<Request>` SAML element with and gets a `<Response>` element containing the requested information.

Binding is used to map SAML request/response mechanism into some kind of communication or messaging protocol. The examples might be mapping to SOAP [28] over HTTP or binding using HTTP Redirect mechanism.

Profiles describe methods for conveying information between sites. For example they might be carried as a part of an URL (browser/artifact profile) or as a POST request (browser/POST profile).

To ensure the confidentiality of transmitted assertions they should be protected with a help of transport layer encryption protocol such as TLS.

2.2.5. Standards and solutions summary

This part described various standards and solutions that make use of previously presented algorithms and on the same time that provides functionality required by security frameworks described in following section. X.509 public key certificates are a basis for such common Grid security technologies as Grid Security Infrastructure (GSI) [29]. SAML is fundamental technology used by the Shibboleth [8] framework which also needs TLS to protect privacy of the transmitted data.

2.3. Security frameworks

This section presents various security frameworks. First a complete authentication/authorization solutions like GSI and Shibboleth are described, then two frameworks enabling interoperation in heterogeneous security infrastructure - ShibGrid and GridShib and finally an identity management solution - OpenID.

2.3.1. Grid Security Infrastructure

Grid Security Infrastructure (GSI) [29] is a security system used in a Globus Toolkit. It is based on PKI, featuring authentication based on public-key certificates as defined by the X.509 standard [21]. Each user must poses own grid certificate signed by a CA trusted by all parties, for example by European national grid CA listed by EUGridPMA [30]. To enable quite secure credential delegation this solution uses notion of so-called proxy certificates. Such certificates should have reasonable short validity period (relatively to permanent certificates) to minimize the chance of its private key being compromised, as it is not encrypted and is attached to the proxy. They are signed by the owner of the Grid certificate with his/her private key rather then directly by the CA. This technology were taken under consideration as supplementary to the Shibboleth for non-web scenarios (as it is very well suited for such use cases) if the Shibboleth itself wouldn't be enough to meet all requirements. Eventually it was decided that it wasn't a case in this situation.

2.3.2. Shibboleth

Shibboleth [8] is a federated Single Sign-On framework supporting authentication and enabling creation of attribute-based authorization solution. It is based on SAML providing secure exchange of authentication and attribute assertions. Its main goals are to allow access for members of various institutions maintaining their own user databases, as well as to provide scalability and fault tolerance. Finally, it allows user authorization even without revealing any personal information if such level of privacy is needed.

The access for members of different institutions keeping separate users' databases is provided by the notion of a Home Organization located at each of them. A part of a Home Organization (HO), an Identity Provider (IdP) is responsible for maintaining its credential and attributes database and authentication system. An IdP consists of a Single Sign-On (SSO) part responsible for assigning handles - short term authentication tokens - to users, and an Attribute Authority (AA) responsible for releasing attribute assertions. Other elements of a HO - Service Providers are responsible for providing all required services. The Home Organizations that need to cooperate formulate a so-called federation that provides trust relations between them and enabling it's users to access Service Providers across the whole federation without a need for a separate account.

The described nature of HOs increase scalability, of the federation as adding a new institution requires just adding of a new HO, and do not require to increase a load on the authentication services of current members. Also the fault tolerance is increased as failure of the element of any HO wouldn't block access for all users, just for the users of the affected HO. If even higher level of scalability or fault tolerance is required it is possible to create a spare HO for single institution.

Shibboleth can provide high level of privacy through attributes that can be used to authorize users, without a need to disclose private information. It is sufficient in most cases to just tell the authorization system that a user is holding a given role at specific Home Organization, to allow it taking the authorization decision.

The Shibboleth has been already used in Web-based part of the ViroLab project as it meets all the requirements for such use case. The main goal of the work described in this thesis was to asses if it is feasible to use it alone for all the parts of the ViroLab security infrastructure (including non-web tools and services), or is there a need to combine it with other security frameworks. Next, it was necessary to find out, design and develop all required customizations of the chosen solution to meet the requirements. The analysis and work described here is based on version 1.3 of the Shibboleth that has been chosen for the ViroLab project because it was the most recent stable version at that time. At present, despite existence of version 2.0, version 1.3 is still considered as stable and fully supported. Of course, most of the work is independent from the specific software version and just it will need some minor modifications to adapt it to the newer one when it is necessary.

2.3.3. ShibGrid

This UK project aims at integration of traditional GSI model based on X.509 certificates with a Shibboleth infrastructure [31]. Its goals were to support both users holding standard grid certificates issued by a national CA as well as users with just Shibboleth accounts.

The former ones could use the project portal to store and retrieve proxy certificates in MyProxy [32] and restrict access to them with the Shibboleth, so in turn they could later use their Shibboleth accounts to access all grid resources.

The latter could get so-called low-assurance grid certificates based just on their Shibboleth attributes. Such certificates allow them limited access to the grid infrastructure without obligation to get real grid certificates.

This solution was analyzed as an alternative to using Shibboleth directly for non-web software, but it doesn't meet all the requirements. In particular it requires that user accesses the portal first, which in the case of stand-alone tools (like the Experiment Planing Environment [7]) is not the best solution.

2.3.4. GridShib

This is another solution [33] aiming at the GSI-Shibboleth integration, maintained by the institutions responsible for development of both technologies. GridShib uses MyProxy [32] online CA to issue short-lived certificates instead of standard proxy-certificates for users with Shibboleth accounts and without real grid certificates. The technology contains a few modes of operation designed to support the generation of such short-lived certificates with embedded as a non-critical extinction of the X.509 certificate Shibboleth assertion - either the authentication assertion with the handle allowing a Grid service to request the attributes, or the attribute assertion itself. For this reason and since this solution could function without a web browser it was considered as a good choice to augment Shibboleth in a ViroLab if Shibboleth-only solution wasn't sufficient. Even though in the case described in the thesis it wasn't used, such a mixed solution might be required in further development.

2.3.5. OpenID

OpenID is a very popular open identity management framework. It allows users with OpenID credentials from any of the providers, to access various websites. However, it cannot supply a complete solution for user authentication for projects like ViroLab, as owners of this type of credentials will still need to be registered in some kind of local user databases for each service. This would be quite complicated in comparison to the federated authentication provided by Shibboleth or even by GSI based solutions. The complexity would be necessary because in contrast to Shibboleth, OpenID IdPs are by design not controlled by any kind of federation that would ensure validity of user information. In fact anyone could create such a provider and anyone could register there. So no user can be trusted to access the production resources just on the basis of the fact that he/she is holding some OpenID credential. However, in further enhancement of the virtual laboratory infrastructure OpenID might be used to simplify the application for access to the grid infrastructure (user registration) or even to give a very limited access to some demonstration part of the infrastructure to users with a valid OpenID credentials (possibly with some limitation to more trusted IdPs).

2.3.6. Summary of Security Frameworks

This section described various security frameworks that were considered as main or supplementary solutions. None of the solutions guaranteed to provide all the required functionality out of the box, however Shibboleth seamed the most promising one. Next parts of the thesis were partially devoted to assess what kind of modification it requires, and if it could be used alone, or if enhancements with other described solution were needed.

2.4. Summary

In this chapter, various cryptographic algorithms were analyzed including encryption (AES, RSA), hashing (SHA), key exchange (Diffie-Helman) and for generation of Keyed-Hash Message Authentication Code. These algorithms could be used by subsequently described standards such as PKI and SAML as well as TLS protocol. Next frameworks based on them like GSI, Shibboleth, ShibGrid, GridShib and OpenID were presented. Finally conclusions were drawn, that rest of the work should be focused on analyzing, designing, implementing, validating and evaluating the solution that would augment Shibboleth to provide seamless security mechanism for a non-Web part of the system.

Chapter 3

Threat Model and Requirements

This chapter is focused mainly on discussing various system requirements both strictly related to the security as the threat model as well as other generic functional and non-functional requirements. Apart from the security requirements, the presented threat model enumerates assets protected by the system as well as threats against them, and also possible attack scenarios that need to be prevented. Finally, this chapter provides information about the chosen solution (especially in relation to the requirements) and customizations it required.

3.1. Introduction

The solution analyzed here provides security for the ViroLab virtual laboratory, solution for in-silco experiments shown on Fig. 3.1. As it is presented there ViroLab utilizes resources provided by the Grid, groups of computers connected by local network forming a cluster or single machine. This resources provides ability to run various computational services such as the plain old jobs submitted to the Grid but also Web Services and components, as well as data services like those provided by DAS [34] or regular databases. Based on provided services the VL run-time environment [5] is able to execute experiments to provide the core functionality. Experiments could be developed by experiments developers using dedicated Eclipse [35] based platform called Experiment Planning Environment (EPE) and used by Scientists and Clinical Virologist with the help of user friendly web-based tool called Experiment Management Interface (EMI) [7].

The solution that provides computational services that was especially adressed in this thesis is the MOCCA [10]. It is a framework that could run distributed components. To provide it with security mechanism compatible with the requirements of the ViroLab, the new authenticator for H2O [9] (container used to run MOCCA components) pluggable authenticator module had to be created.

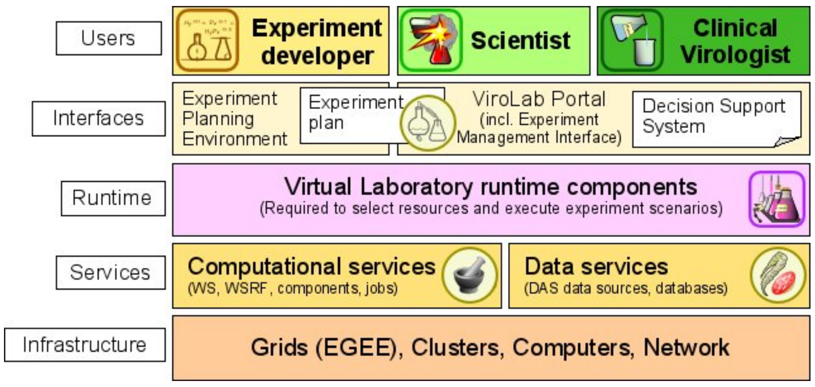

Users
Interfaces
Runtime
Services
Infrastructure
Experiment developer
Scientist
Clinical Virologist
Experiment Planning Environment
Experiment plan
ViroLab Portal (incl. Experiment Management Interface)
Decision Support System
Virtual Laboratory runtime components
(Required to select resources and execute experiment scenarios)
Computational services
(WS, WSRF, components, jobs)
Data services
(DAS data sources, databases)
Grids (EGEE), Clusters, Computers, Network

Figure 3.1. Architecture of the ViroLab virtual laboratory showing its components: including infrastructure, computational and data services provided through various technologies, run-time environment, EPE and EMI interfaces and different groups of users

3.2. Threat Model

This section describes analysis of security requirements that must be met by the virtual laboratory, the assets protected by the security infrastructure and the threats against them, as well as enumerates possible attack scenarios and methods that should be used to prevent them.

3.2.1. Security Requirements

Basic security requirements for this VL system are authentication, credential delegation, authorization, confidentiality, integrity, availability and non-repudiation.

The authentication solution needs to ensure that the user is who s/he claims to be. Additionally, as mentioned already VL requires to be provided with a Single Sign-On mechanism for all the services.

Credential delegation is specific for the distributed system. It requires that user's software can be run on various nodes which in turn prompts the need to safely delegate user's credential to each following nodes after the authentication to the first one.

Authorization needs to control access to the services by verifying if a user is authenticated and has required attributes. As it has been already described this type of the authorization mechanism is required by the virtual laboratory.

Confidentiality is needed as the access to transmitted as well as stored data including experiments, results, users credentials and attribute must be kept private.

All the elements mentioned above must be also safe from being tempered with, as it might lead to a breach of system security or creation of phony experiment results. This prompted the need to ensure **integrity** of stored and transmitted data.

Sufficient availability must be provided as any interruptions (e.g. those caused by some kind of a denial of service attacks) would cause problems for users and might lead to lose of some computational results.

Non-repudiation is also required as it must be possible to prove who uploaded the experiment in case some kind of malicious code were uploaded.

3.2.2. Assets and threats

This section describes assets such as medical databases, user databases, experiments scripts and results as well as computational and network resources that should be protected by the system. It also shows threats against this resources related to its theft, destruction or possible abuse for criminal purpose. Mentioned here information is presented in a Tab. 3.1 and Tab. 3.2. The most dangerous would be of course possibility to alter user database as it may give the attacker unlimited access to all the resources. Also ability just to read this database might lead to similar results if the attacker could access passwords hashes. Passwords are hashed with already described strong algorithm (SHA1) and a so-called salt is added before hashing to prevent usage of the Rainbow Tables [36] to get plain-text passwords much faster then with a brute force search. Despite that it still might be feasible to use brute force to crack some of these passwords especially if the attacker could acquire access to large (e.g. distributed) computational resources. Also computational and network resources are quite critical as the former might be used for mentioned here password cracking and the latter could allow to perform very dangerous Distributed Denial of Service attack from the VL network.

3.2.3. Attack scenarios

The main goal of a security infrastructure is to protect the system from various attack, that might be directed toward it [37]. This section presents results of an analysis of the most likely attacks and comments how the system might be designed to stop them.

Plain-text eavesdropping - the simplest attack, requiring that data are being sent without encryption; To prevent this attack all connection that are not sufficiently secured at lower level must be encrypted either at the transport level (e.g. via TLS [25]) or at the message level.

Man in the middle attack - even encrypted transmission isn't always safe as someone might try to establish a connection with both communicating sides implying to each of them that s/he is the opposite one. That way, if successful, the attacker would be able to decrypt analyze, or modify and re-encrypt all the communication. To prevent such attack, a well configured PKI [23] is needed with all the CA certificates securely distributed so no fake certificates could be trusted as legitimate ones.

Password cracking - an attacker might try to guess users passwords either using a dictionary to check if passwords are common phrases or Rainbow Tables [36] if hashes were unsalted or by exhaustive search of all valid characters combinations

Assets	Descriptions	Threats
Medical databases	This category contains various medical data like types and mutations of HIV, drugs and its effectiveness for specific mutations and interactions. This data are anonymized to make them less sensitive, but they are still valuable for potential attacker.	The attacker might try to steal a data or tamper with them in some way. The second case may lead to further coruption of calculated results The attacker might also try to destroy the data.
User databases	Credentials along with attributes are being kept in LDAP database.	The biggest threat against this type of data is tamper that might lead to entering illegitimate credentials or attributes opening the system for the attacker or escalating privileges of current user. Theft of the data may allow password hashes cracking or at least leak of personal data.

Table 3.1. Table enumerating assets stored by the system that need to be protected [37]. Detailed descriptions as well as threats against those assets are included

(so-called brute force method). Such search will be much easier if the attacker posses database of password hashes, even though s/he cannot directly decrypt them (as cryptographic hashing functions don't have reverse functions by definition) however it will enable him/her to validate password candidates without a connection to the system, saving time and risk of blocking such repeated connection attempts. It is highly important to ensure that users do not have weak passwords (short ones or based on words present in dictionaries), because a password complicated enough would make the highly complex brute force attack infeasible. It is also important to use the salted hashed and to protect the credential databases containing password hashes.

Phishing [38] - is an example of a social engineering technique aimed to trick a legitimate user of the system to reveal his/her credentials to the malicious party. Most common cases involve sending the user the e-mail claiming that for some reason s/he need to go to a given (attacker's) website and give his/her password, otherwise something bad will happen (e.g. an account is going to be blocked). Usually such a web-page well mimics the real one (in case of a ViroLab it might be e.g. the main Portal or the stand-alone version of Experiment Management Interface). It is crucial to instruct the users to watch out for such fake sites, by checking URLs and certificates presented by web pages, and not to give a password even to people claiming to be system administrators.

Pharming [39] - in opposition to phishing, this technique is based not on social

Assets	Descriptions	Threats
Experiment scripts	Experiment scripts are used for computations during in-silco experiments They contain valuable intellectual property rights.	Experiments might be stolen or tampered with. Like in the case of input data from medical databases an illegal modification might lead to the generation of faulty results.
Experiment results	Results are generated during in-silco experiment.	Attacker might try to steal, modify or destroy them.
Computational resources	Distributed system being the backend for the VL engine is also valuable for the attacker.	These resources may be abused by malicious party, for activities such as password cracking or decrypting encrypted data.
Network resources	In a similar way to hardware resources ViroLab posses huge network resources that could be a tempting target for the attacker.	The attacker might try to use network resources after obtaining access to the system to perform Denial of Service attacks on the external targets. On the other hand s/he may try to use own resources to attack ViroLab network resources.

Table 3.2. Table enumerating assets stored by the system that need to be protected [37]. Detailed descriptions as well as threats against those assets are included (continuation)

engineering, but on redirecting communication by supplying fake DNS query results. A social engineering however might be used to install malicious software on users' computers that will alter valid DNS resolver addresses. After being redirected, the user might be attacked similarly to phishing, so the user might be directed to illegitimate website pretending to be entry point to the system and asking for a password. The attacker might also try to redirect other software to perform the man-in-the-middle attack. In both cases care must be taken to ensure that both DNS servers, as well as client machines are not compromised. Additionally in the first case similar measures should be taken as in case of phishing, the second case should not work unless conditions described in men-in-the-middle section are not met.

Social engineering [40] (other then phishing) - in addition to very common case called phishing, there are also other variation of this type of attack. All of them are aimed not against the software or hardware but against users of the system. For example a user might not necessary be asked to reveal his/her credential - instead the attacker might ask him/her to access a web site with some interesting content, or open e-mail attachment containing something either funny or shocking. Such action on the side of the user would result in installation of some kind of malware, that might for example log user keyboard activities (especially passwords) or simplify other attack like pharming (by replacing DNS configuration) or man-in-the-middle attack (by adding illegitimate certificates to the list of trusted ones). Like in the case of phising the only protection is to educate users not to trust such messages/offers.

Exploiting software vulnerabilities - attacker might also try to exploit vulnerabilities in installed software, either third-party (like web servers, LDAP servers or RDBMSes) or custom created for the project by the author or others. To prevent an attack against the third-party software it is needed to frequently check for security announcements and keep software up to date and well patched. In the case of custom software it is necessary to perform strict check of the created source code for possible vulnerabilities and making sure that no vulnerabilities are present in production system. If, despite that any are found, there is necessity to release and install appropriate bug-fixes as fast as possible. In addition to that it is prudent to perform a security audit of the installed software from time to time.

3.3. Generic System Requirements

This section enumerates functional and non-functional requirements for the security framework developed in the scope of this thesis, other then the basic security requirements that already have been explained in details.

3.3.1. Functional Requirements

The system must provide following functionality: ability to store user credentials and set of attributes, ability to provide authenticated access for non-web applications, authorization mechanism for MOCCA/H2O, ability to store, look up and update local MOCCA policies and providing administrator's tool for policy distribution mechanism.

System must provide *means to store and modify* both user credential used for authentication, as well as data that might be used by the services for authorization. Examples of such authorization data are users attributes.

Authenticated access for non-web applications must be provided (as functionality for web portal was already provided by external partner). This element includes the need for appropriate library and reference command line client for users preferring this form of user interface rather then web site.

It was also necessary to augment MOCCA/H2O with functionality enabling *user authorization* based on attributes provided by the external partner's software (ShibAuthAPI and ShibRPC) [37]. This module needs to dynamically assign users with specific attributes to groups (like deployer, administrator) based on local set of policies (specific for MOCCA/H2O part of the VL) providing additional fine grained access control in addition to more coarse grained one provided by global policies controlled by ShibAuthAPI.

The ability to store, look up and update local MOCCA policies was also needed. This requires creation of solution that would allow distributed MOCCA nodes to verify if it's policies are up to date (in relation to the centrally stored ones) and otherwise to download the new one. The system must allow *easy centralized administration* of them via dedicated tool.

3.3.2. Non-functional Requirements

There are also several non-functional requirements such as user friendliness, efficiency, scalability and maintainability.

User friendliness is especially important for parts of the system (tools) that are dedicated for non-IT experts, like virologist or medical doctors. Security system should try to keep them away from procedures that might be complicated for them (like requesting or renewing certificates in the case of certificate based solutions common in the Grid systems).

The system must be *efficient*, to ensure that time required for authentication and authorization procedures as well as policy updating is short enough for the users of the system.

Both off the shelf components used to build the base security infrastructure as well as custom software that were written should be designed in the way that *allow scaling* in the case of increased load.

Maintainability is needed so the designed system could be extended or modified if either new requirements arise, like the need to incorporate new security frameworks, or the current components became obsolete and need to be replaced by the new versions (which is very important especially in the case of a security system).

3.4. Discussion of the Chosen Solution

The Shibboleth have been chosen as a single solution for the whole infrastructure. This section is aimed to show the chosen solution in general, then describes how it meets the requirements and finally mentions what customisations were required to meet all of them.

3.4.1. Shibboleth as the Chosen Framework

When working on this thesis the constraint was that the solution must be integrated with the Shibboleth elements that already were developed by other partners of VL. Part of the task was to analyze if Shibboleth could be tuned and customized to meet the requirements for non-web components developed at Cyfronet or whether to choose a framework that could inter-operate with the Shibboleth if the former option was infeasible. After performing the careful analysis of existing frameworks and the requirements, it was decided that the Shibboleth is capable to support the whole infrastructure. Consequently, the missing parts that needed custom solutions were designed, implemented validated and evaluated. Finally, integration of these new solutions and third-party software (including the one provided by external partners) was performed.

3.4.2. Relation between the Solution and the Security Requirements

Shibboleth meets most of the mentioned security-related requirements and provides solutions helping to meet all of them. The following discussion shows how each security requirement is being met by the chosen solution.

Authentication - Shibboleth provides a ready to use solution for authentication of users in Web environment. In the case of non-web part of the infrastructure a creation of custom solution was necessary as described later.

Credential delegation - is provided by delegating Shibboleth handles which could be seen as a short lived credentials suitable for this task.

Authorization - Shibboleth provides for each authenticated user set of attributes that are a good basis to authorize them for specific resources, based on set a of policies. In typical web application the authorization is supported out of the box. While specific needs of ViroLab Portal where addressed by external partner, on the other hand authorization module for MOCCA (non-web application) needed to be created from scratch as a part of this thesis.

Confidentiality - as required by guideline for Shibboleth IdP installation [41] all the communication with it must be carried via the encrypted transport (TLS).

Integrity of the communication between components of the VL is also ensured by the TLS. Additionally, all Shibboleth assertions holding authentication- and authorization relevant data are digitally signed by the Shibboleth Identity Provider.

Availability - architecture of the Shibboleth as federated SSO solution is designed in such way that each organization is maintaining it's own components (including IdP). This ensures that a failure of a single IdP doesn't block access to all users, just those from the same organization. There is also no problem with adding redundant Home Organizations (containing IdPs) for single organization in the case of Shibboleth.

Non-repudiation - each user from each organization is registered and logged by services as a separate entity. If needed the Shibboleth logs from specific HO in combination with other service logs might be used to identify him/her.

3.4.3. Relation between the solution and the functional requirements

Like in the case of the security requirements presented in the threat model, the *functional requirements* are also directly met by the stock version of the Shibboleth, or could be provided by some kind of customization.

The ability to store the user credentials and attributes is provided out of the box by one of the back-ends used by the Shibboleth. In the case of ViroLab both credentials and attributes are stored in a LDAP.

The authentication mechanism for non-web tools is provided by custom library called ShibIdpClient and based on it command line tool called ShibIdpCliClient. Other tools such as EPE also integrate with this library.

The authorization functionality for MOCCA/H2O is provided by a custom Shibboleth authenticator created specifically for this purpose. This authenticator enables users to use their Shibboleth credential in a similar way as other previously supported, like GSI [42] or standard password-based credentials.

The ability to update local MOCCA policies is supported by a custom tool called MOCCA Policy Distribution Point Client being a client to another tool that was created for the project - MOCCA Policy Distribution Point, used to store, mange and provide local MOCCA policies. The ability to centrally modify MOCCA policies is provided by a web tool created for this purpose - MOCCA Policy Distribution Point Administrator's Panel.

3.4.4. Relation between the Solution and the Non-functional Requirements

Similarly to the functional requirements, a non-functional requirements are fulfilled by the chosen Shibboleth framework or derivatives based on it. As before in case of other type of requirements the following describes how they are met.

User friendliness - web-based Shibboleth solutions are much friendlier for the users then GSI based one. This is mostly because they do not force a user to apply for any certificates. An access to whole infrastructure requires just to choose user's Home Organization and enter login and password as usual. A non-web authentication mechanism that were created to supplement Shibboleth framework is designed to work in a very similar way. MOCCA authenticator requires just creation of a simple XML configuration file. Its policies are being updated automatically from PDistP, after it has been configured through simple Web tool.

Efficiency - all the components were successfully evaluated to be efficient enough for the task. Detailed information on this subject are presented later in the section 7.2.

Scalability is provided in case of Shibboleth IdP by already mentioned possibility to add additional HOs.

Maintainability - architecture has been designed in a way supporting quite manageable addition of another services and updating security solution to further versions. Introduction of a new security framework will be more complicated as it will require addition of some kind of gateway, but is possible and was even analyzed with promising results in case of integrating with the GSI (subsections 2.3.3 and 2.3.4).

3.4.5. Required Customization

However the Shibboleth was found as a solution good enough as described above, it required some customization. As part of the project and this thesis, the following software components have been created: ShibIdpClient, ShibIdpCliClient, MOCCA Shibboleth Authenticator, Policy Distribution Point, it's client and administrator's panel.

ShibIdpClient is a library which supports retrieving, authentication information (a handle) from the Shibboleth SSO and is feasible for non-web tools like the Experiment Planning Environment or mentioned below command line interface (CLI).

ShibIdpCliClient is a reference implementation of tool using ShibIdpClient library to provide authentication solution for users preferring system shell to the web interface

MOCCA Shibboleth Authenticator is a component allowing authorizing Shibboleth users the access to MOCCA framework based on their attributes and policies with various access level

MOCCA Policy Distribution Point is a centralized entity storing policies for the MOCCA Authenticator with XML-RPC [11] interface. Such an interface allows access from tools based on various technologies.

MOCCA PDistP Client is a client for PDistP that might be installed on MOCCA nodes to check if local policies are up to date and update them when needed.

MOCCA PDistP Administrator's Panel is Web application supporting changing PDistP policies. Such a change is required when it is necessary to grant or revoke access for users with some specific attributes.

3.5. Summary

This chapter was focused on describing requirements that must be met by the proper security solution for VL, as well as describing characteristic of the solution itself. The solution has been based on Shibboleth framework with some customization for non-web solutions such as ShibIdpClient for EPE and ShibIdpCliClient, and MOCCA Shibboleth Authenticator. The chapter also proves that such solution meets all necessary requirements.

Chapter 4

Architecture of the Virtual Laboratory Security System

The goal of this chapter is to depict the architecture of the whole security infrastructure, including the newly created parts and integration with third-party software. Then each component of this infrastructure is described in more details, and finally some notable samples of interaction between the elements of mentioned components performing typical use cases are shown.

4.1. General Architecture

The general architecture of the security system is shown in Fig. 4.1. Such an architecture provides the required functionality for the users. This section describes in details how each element provides the mentioned functionality.

The Identity Provider (IdP) is a standard element of the Shibboleth infrastructure and is being used both by authentication and authorization components.

Its SSO component provides SAML authentication assertions transmitted over HTTPS. ShibIdpClient library is used to access SSO and provide authentication token based on this assertion to the tools that require it, such as EPE or standalone version of EMI as well as dedicated command line client (ShibIdpCliClient) that can be used by more advanced users.

Another part of the IdP - the Attribute Authority provides SAML attribute assertions to the external partner's component - ShibAuthApi/ShibRPC that simplifies access to attributes by providing lightweight XML-RPC based protocol. MOCCA Shibboleth Authenticator is used to authorize users based on attributes returned for a given handle.

Finally, the Policy Distribution Point provides a way to store (in MySQL database) and maintain local MOCCA policies. These policies are later used by PDistP client providing the ability to keep policies of MOCCA nodes up to date after they could be modified using an administrator's panel. PDistP like ShibRPC uses XML-RPC protocol. The reason is that this protocol is lightweight, very portable to various technologies and suitable for the task. It is already used by other ViroLab software due to the usage of ShibRPC module, which eliminates necessity to add other communication libraries for PDistP.

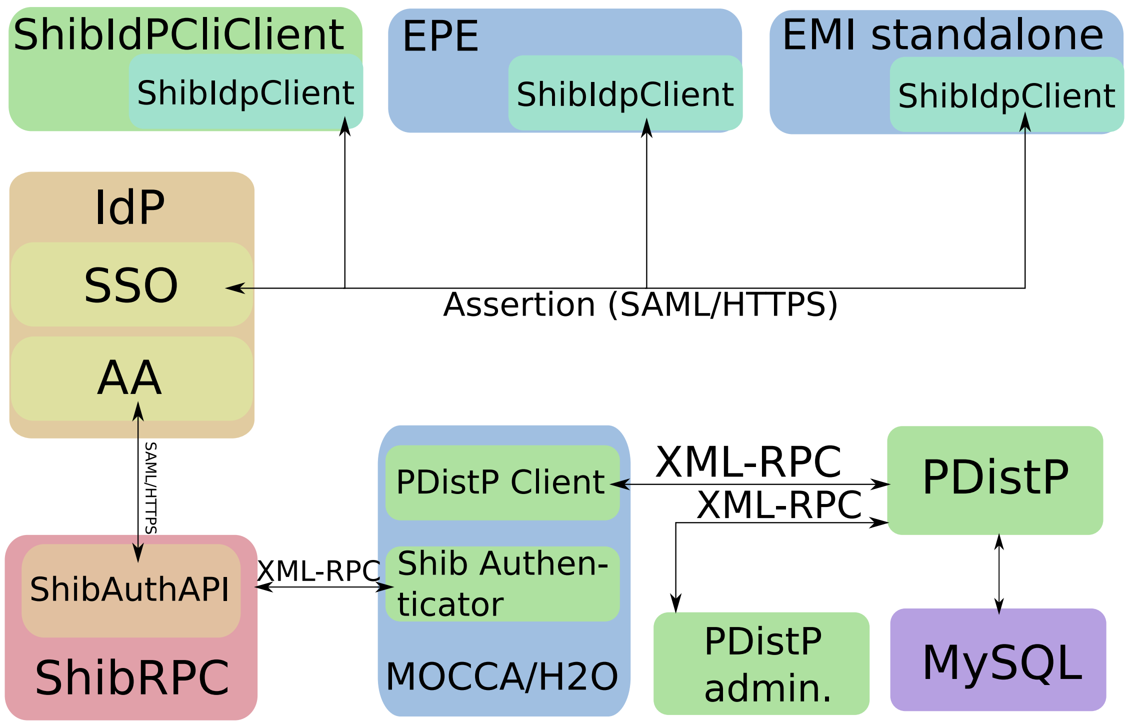


Figure 4.1. Architecture of the security system consisting of all mandatory components including authentication and authorization solutions, external client components as well as communication protocols.

4.2. System Components

This section explains in more details architecture of each component that has been created for this thesis. It's main goal is to break down the picture already described into smaller parts.

4.2.1. ShibIdpClient and ShibIdpCliClient

This section describes both ShibIdpClient (a software library) and a tool that uses it (ShibIdpCliClient). The architecture is shown on the Fig. 4.2

ShibIdpClient connects via the HTTPS protocol to the Single Sign-On (SSO) component of the Shibboleth Identity Provider being member of the appropriate

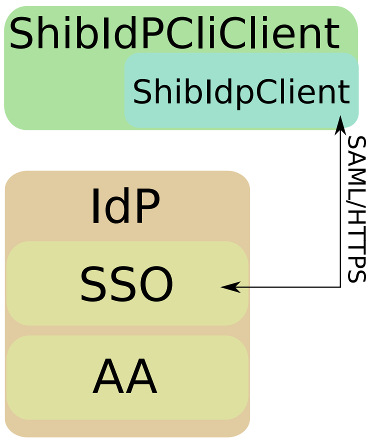


Figure 4.2. Architecture of the ShibIdpCliClient and ShibIdpClient - command line tool and library providing Shibboleth authentication capabilities for non-web based software components

Home Organization specified in the configuration. Its task is to validate the server certificate, authenticate the user and extract user's handle. For example handle might looks like this one:

`_233f6bb9258bcd1cf0473a#https://virolab.cyfronet.pl/shibboleth-idp`

It consists two parts - typical Shibboleth handle being a hexadecimal number with _ sign at the beginning with an underscore sign and additional HO address separated with a # sign.

Validation of server's certificate hosting the SSO is mandatory to make sure it has been issued by a trusted CA. Omission of this operation might lead to releasing user's credentials to fake web server pretending to be a legitimate one.

Authentication of a user via the basic HTTP authentication is required as this mechanism has been chosen as the most friendly way of protect SSO for not just users but what was important for non-Web tools also machines.

As a first step in extracting the handle, the software needs to download and parse the HTML document returned by SSO to extract Base64 encoded SAML authentication assertion. After decoding it, SAML needs to be validated and finally the handle might be extracted from it.

ShibIdpClient provides standardized interface for application using it such as ShibIdpCliClient - command line interface whose tasks are to ask a user for his/her credentials, request the handle via this interface and display handle to the user.

4.2.2. MOCCA Shibboleth Authenticator

MOCCA Shibboleth Authenticator provides the ability to protect MOCCA using a Shibboleth security framework and enabling it to securely cooperate with other parts of the infrastructure. It is used to provide capabilities to map Shibboleth attributes assigned for each user to MOCCA groups, used for authorizing users for various tasks. The architecture is shown on Fig. 4.3 .

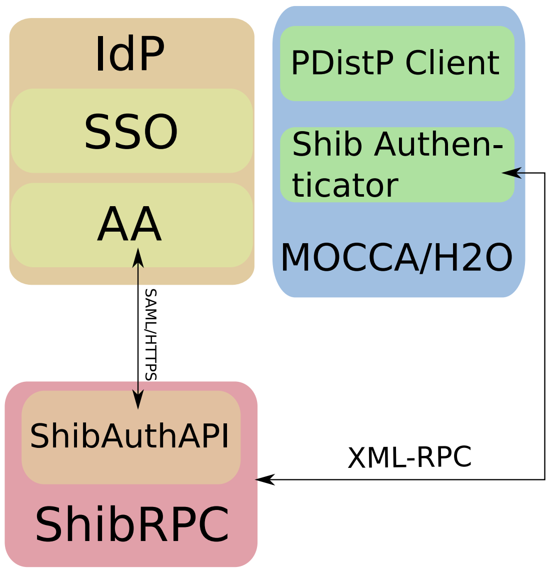


Figure 4.3. Architecture of the MOCCA Authenticator - component augmenting MOCCA/H2O with capabilities to control access based on a Shibboleth credentials

The authenticator uses third-party components ShibAuthAPI and ShibRPC responsible for direct communication with the Identity Provider, processing SAML attributes assertions and providing access to them via a XML-RPC based interface. It accepts Shibboleth credentials for incoming connections, requests attributes via ShibAuthAPI/ShibRPC based on the handle, and finally maps attributes to user groups according to the local policies, or deny if they are insufficient to grant any level of access.

The solution described here is based on the H2O pluggable authenticators. It enables developers to provide custom authenticator for required security solution. For example in addition to described here Shibboleth authenticator MOCCA features the GSI authenticator and the standard password authenticator. Usage of the specific authenticator is controlled through the `<Authenticators>` section of the `KernelConfig.xml` file.

4.2.3. MOCCA Policy Distribution Point and its Tools

A Policy Distribution Point, its MOCCA client and administrator's panel a are set of tools created to support storing, maintaining and distribution of local MOCCA policies to each node running MOCCA software. All the mentioned here components are depicted on Fig. 4.4 .

The core element is of course Policy Distribution Point itself. It is responsible for:

- storing in the RDBMS (MySQL) up to date information needed for generation of the policies,

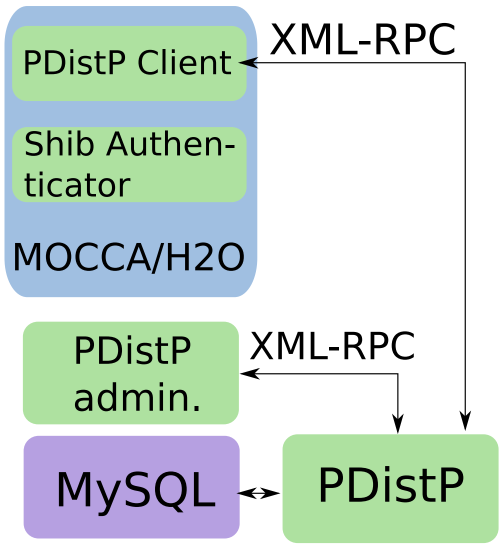


Figure 4.4. Architecture of the Policy Distribution Point; a solution for storing and maintaining local MOCCA policies; as well as its client ensuring that policies on nodes are up to date; and the administrator tool

- storing its own users credential, with assigned role of either normal user (that could download policies) as well as administrator (that could modify them),
- providing XML-RPC based interface for the client and administrator tool,
- performing authentication and authorization of the users,
- allowing managing it's own users through the interface,
- generating local policies on demand and returning them to the client,
- allowing modifying the policies via the interface.

Policy Distribution Point Client supports:

- checking with PDistP if local policies are up to date,
- requesting new policies when local version is obsolete,
- updating local version with downloaded policies.

Finally, the administrator panel is used for:

- managing internal PDistP users and roles,
- adding, removing and modifying MOCCA policies,
- viewing PDistP logs.

4.3. Interaction between Security Components

This section presents on diagrams of sample interaction between above mentioned security components that is performed during realization of the selected important use cases like authentication to the IdP, accessing MOCCA protected by the authenticator, updating local MOCCA policies and changing MOCCA policies.

4.3.1. Authentication to the IdP

The first use case is when a user owning a Shibboleth account wants to authenticate to the IdP (get handle) with the help of a command line client. Actions required for such a case are shown in Fig. 4.5 .

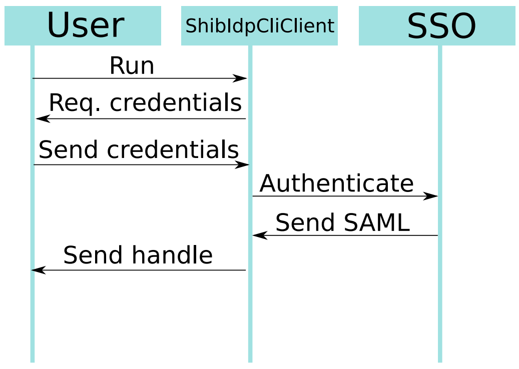


Figure 4.5. Authentication to the IdP with the help of the ShibIdpCliClient performed to acquire a Shibboleth handle without using a web browser

The authentication process consists of the following steps:

1. *Run* - user run the software,
2. *Req. credentials* - the client asks user for credentials,
3. *Send credentials* - user gives his/her credentials,
4. *Authenticate* - client authenticates to the SSO and requests authentication assertion for the given user or informs him/her that credentials are invalid,
5. *Send SAML* - SSO sends back SAML containing authentication assertion with valid handle,
6. *Send handle* - client extracts handle from the assertion and displays it to the user.

4.3.2. Accessing MOCCA Container Protected by the Authenticator

MOCCA authenticator provides authorization for various actions in H2O kernel like deploying pluglets and running them. This specific use case features authentication and authorization process of a user accessing the protected MOCCA container. As before, the process is illustrated in Fig. 4.6 and then is described in more details bellow:

1. *Provide handle* - user presents a valid handle during connection to MOCCA,
2. *Req. attributes* - MOCCA authenticator requests his/her attributes via ShibRPC
3. *Send attributes* - ShibRPC returns valid attributes for this user or informs that the handle is not valid or not trusted
4. *Authorization decision* - if the handle is invalid the authenticator refuses access and informs user. If they are valid, it maps attributes to MOCCA group ac-

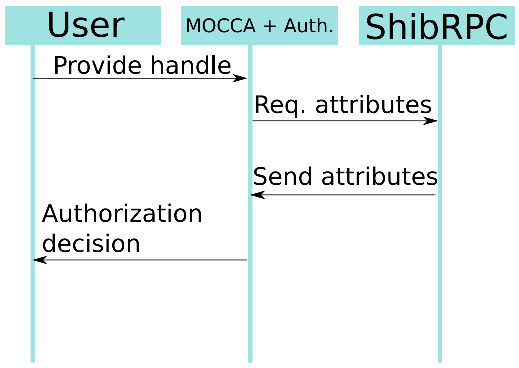


Figure 4.6. Deployment of MOCCA component to the H2O container protected by the MOCCA Shibboleth Authenticator - a solution for integration MOCCA with a Shibboleth protected infrastructure

cording to the policies, and makes authorization decision depending on whether resulting group has sufficient access level for a requested action or not.

4.3.3. Updating Local MOCCA Policies

PDistP Client need to check periodically if policies in PDistP haven't been changed and if they need to update local copy. This is required, because for smooth administration policies are modified centrally. The PDistP Client updates policies by downloading new ones to replace the obsoleted version. Such action is shown in Fig. 4.7 .

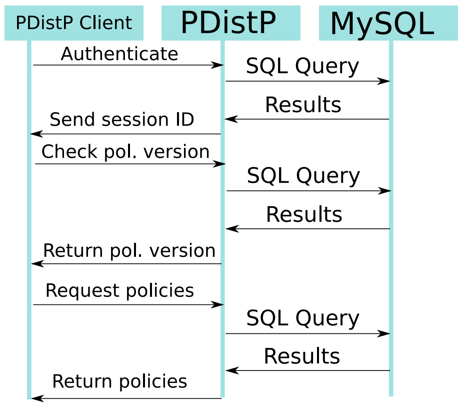


Figure 4.7. Checking if local MOCCA policies haven't been changed. If so new policies are downloaded to replace the obsoleted ones

The procedure requires to:

1. *Authenticate* - the client sends PDistP user credentials via XML-RPC,
2. *SQL Query* - PDistP checks the credentials and adds a new session if they are valid,

3. *Results* - PDistP gets a response if the credentials are valid,
4. *Send Session ID* - PDistP returns the session ID to the client (used for further operations) or denies access,
5. *Check pol. version* - if the access is granted the client asks for current policy version stored in PDistP,
6. *SQL Query* - PDistP is querying stored policies version from RDBMS
7. *Results* - policy version is returned
8. *Return pol. version* - PDistP sends policy version to the client
9. *Request policies* - if policy version is newer the client requests new ones,
10. *SQL Query* - PDistP queries for policies
11. *Results* - policies are returned from DB
12. *Return policies* - policies are sent to the Client, which stores them localy

4.3.4. Changing MOCCA Policies

From time to time it is necessary to modify policies stored in the PDistP. It has to be done each time the status of the federation changes. For example, if new HOs are added specific policies are required to give their users access to MOCCA. Similar action must be performed if some new values of existing attributes are defined (like a new role). Of course, sometimes policies must be changed even if federation is in a constant state, for example if we need to grant or revoke access to specific groups of user based on the attributes, or to change groups assigned to specific attributes. Such process is described in Fig. 4.8 .

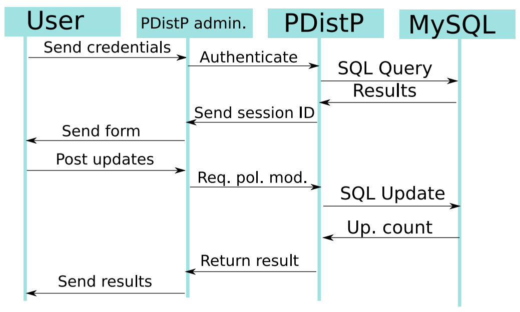


Figure 4.8. Changing MOCCA policies with the help of the Policy Distribution Point Administrator Panel

This action follows this scenario:

1. *Send credentials* - user enters admin credentials into a web site,
2. *Authenticate* - PDistP administrator panel send those credentials via XML-RPC to PDistP,
3. *SQL Query* - PDistP requests user information from DB
4. *Results* - PDistP gets user information from DB,
5. *Send session ID* - PDistP returns session ID or informs about authentication failure,

6. *Send form* - tool sends to client's browser a form allowing, modification of attributes or information that access is denied,
7. *Post updates* - if the access is granted the user sends his/her modification to the policies,
8. *Req. pol. mod.* - the panel request that PDistP modifies attributes via XML-RPC,
9. *SQL Update* - PDistP sends the update command to RDBMS to change the appropriate tables,
10. *Up. count* - DB returns the number of modified rows to indicate success or failure
11. *Return result* - PDistP returns the modification result to the panel,
12. *Send results* - panel displays (returns to user's browser) results of the modification

4.4. Summary

This chapter provided the description of a complete and general architecture view of the developed the system, as well the details for specific parts of the solution. Additionally, the interaction between described components has been presented on the basis of the most common use cases related to the discussed subject.

Chapter 5

The Design of Security Components

The goal of this chapter is to use UML class diagrams to specify the design of the security system components described in chapter 4. The designed components are divided into common groups presented in separate sections.

5.1. ShibIdpClient and SibIdpCliClient

This section contains class diagram (presented in Fig. 5.1) for components related to requesting shibboleth handle from the Identity Provider - ShibIdpClient library and ShibIdpCliClient that uses it. ShibIdpClient has been designed in such way that would provide seamless integration with other applications that requires it's functionality even for programmer without a solid background in Shibboleth related technologies. This requires that the library exposes as simple API as possible. Because of that the whole interface has been reduced to the most basic and only necessary function - getting handle based on provided configuration. The interface is as follows:

```
public interface ShibIdProviderClient {

public String acquireHandle(HandleRequesterConfig config)
throws CannotAcquireHandleException;
public String acquireFullHandle(HandleRequesterConfig config)
throws CannotAcquireHandleException;
}
```

The first method is used to get plain Shibboleth handle, and is just for backward compatibility, the second one returns handle in format currently used in the ViroLab: standard_handle#idp_url . HandleRequesterConfig is an interface supporting flexible way of providing necessary configuration by the library:

```
public interface HandleRequesterConfig {

public String getLogin();
public char[] getPassword();
public IdProviderEntry getIdProviderEntry();
public String getTrustStoreFilename();
public char[] getTrustStorePassword();
}
```

Client tool using the library just needs to implement this interface to provide user credentials, IdP configuration (IdProviderEntry class) and Trust Store location (with IdP certificate) that it might store at it's creator discretion. Then default client implementation (class DefaultShibIdProviderClient) might be used to request the handle. Additional presented classes are used as helpers (SAX parser, DTD validator preventing connection to w3.org each time and an exception). ShibIdpCliClient provides class ClientConfig implementing HandleRequesterConfig as well as Client class being main class responsible for communication with users.

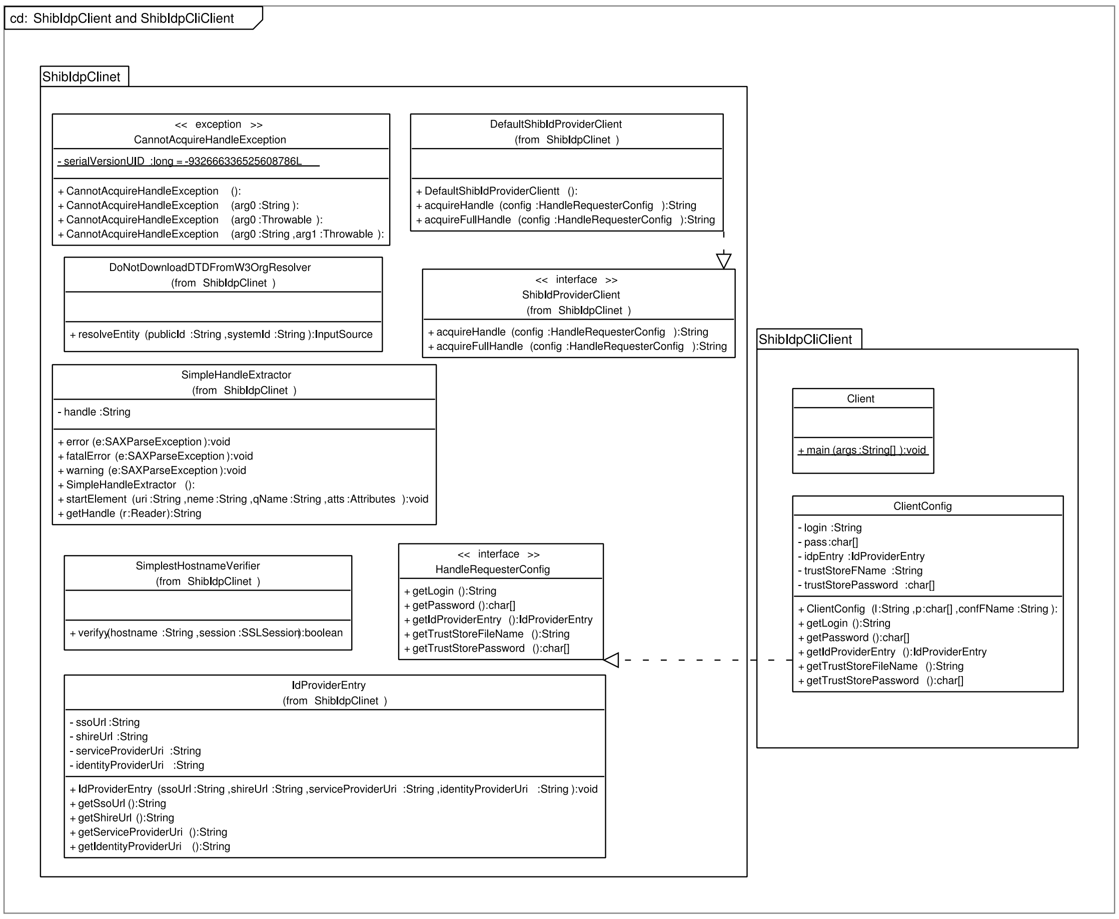

cd: ShibIdpClient and ShibIdpCliClient
ShibIdpClinet
<< exception >>
CannotAcquireHandleException
- serialVersionUID :long = -932666336525608786L
+ CannotAcquireHandleException ():
+ CannotAcquireHandleException (arg0 :String):
+ CannotAcquireHandleException (arg0 :Throwable):
+ CannotAcquireHandleException (arg0 :String ,arg1 :Throwable):
DefaultShibIdProviderClient
(from ShibIdpClinet)
+ DefaultShibIdProviderClientt ():
+ acquireHandle (config :HandleRequesterConfig):String
+ acquireFullHandle (config :HandleRequesterConfig):String
DoNotDownloadDTDFromW3OrgResolver
(from ShibIdpClinet)
+ resolveEntity (publicId :String ,systemId :String):InputSource
<< interface >>
ShibIdProviderClient
(from ShibIdpClinet)
+ acquireHandle (config :HandleRequesterConfig):String
+ acquireFullHandle (config :HandleRequesterConfig):String
SimpleHandleExtractor
(from ShibIdpClinet)
- handle :String
+ error (e:SAXParseException):void
+ fatalError (e:SAXParseException):void
+ warning (e:SAXParseException):void
+ SimpleHandleExtractor ():
+ startElement (uri :String ,neme :String ,qName :String ,atts :Attributes):void
+ getHandle (r:Reader):String
SimplestHostnameVerifier
(from ShibIdpClinet)
+ verifyy(hostname :String ,session :SSLSession):boolean
<< interface >>
HandleRequesterConfig
+ getLogin ():String
+ getPassword ():char[]
+ getIdProviderEntry ():IdProviderEntry
+ getTrustStoreFileName ():String
+ getTrustStorePassword ():char[]
IdProviderEntry
(from ShibIdpClinet)
- ssoUrl :String
- shireUrl :String
- serviceProviderUri :String
- identityProviderUri :String
+ IdProviderEntry (ssoUrl :String ,shireUrl :String ,serviceProviderUri :String ,identityProviderUri :String):void
+ getSsoUrl ():String
+ getShireUrl ():String
+ getServiceProviderUri ():String
+ getIdentityProviderUri ():String
ShibIdpCliClient
Client
+ main (args :String[]):void
ClientConfig
- login :String
- pass :char[]
- idpEntry :IdProviderEntry
- trustStoreFName :String
- trustStorePassword :char[]
+ ClientConfig (l:String ,p:char[] ,confFName :String):
+ getLogin ():String
+ getPassword ():char[]
+ getIdProviderEntry ():IdProviderEntry
+ getTrustStoreFileName ():String
+ getTrustStorePassword ():char[]

Figure 5.1. ShibIdpClient and ShibIdpCliClient components providing non-Web access to the Shibboleth infrastructure - UML diagram

5.2. MOCCA Shib Authenticator

MOCCA Shibboleth Authenticator requires specific design compatible with standard H2O pluggable authenticators structure. It's classes are presented on Fig. 5.2. The challenge was to design this module in a way that it could provide wide range of configuration options, but to rely only on supported interface for H2O authentication pluggable modules (not needing any H2O modifications). The main goal was to adapt architecture based on the data read from 2 files:

- Users.xml - listing users and passwords for standard password authenticator, as well as static bindings between users and groups
- Policy.xml - file containing H2O policies (not to be confused with MOCCA local policies that were mentioned before), which are mappings between groups and Java permissions defined for specific group

In the Shibboleth solution Policy.xml file was kept, however Users.xml was replaced with Shib.xml holding local MOCCA policies (the one used to map attributes to Groups). This file is supported by the ShibTrustDB class. It is used by the basic classes - SimpleHandleAuthenticator that is an implementation of the RemoteAuthenticator interface and internal ShibDialogV1 implementing AuthDialog. Because of that both classes must have appropriate construction of a typical H2O authenticator dictated by this interfaces. ShibDialogV1 class also uses SimpleHandleCredential class that implements RemoteCredential interface. Other classes are supplementing the main classes with exception and containers data read from Shib.xml .

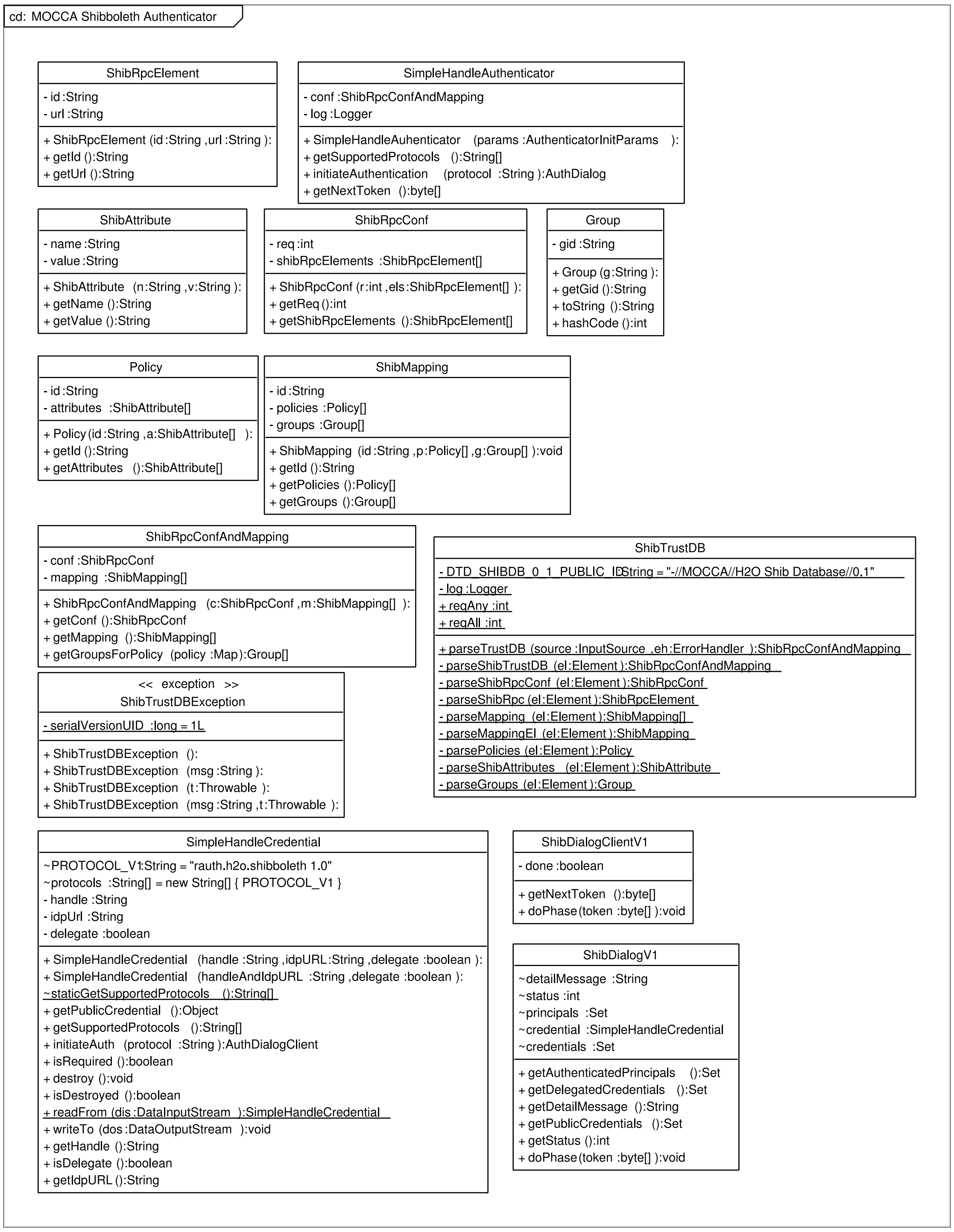

cd: MOCCA Shibboleth Authenticator
ShibRpcElement
- id :String
- url :String
+ ShibRpcElement (id :String ,url :String):
+ getId ():String
+ getUrl ():String
SimpleHandleAuthenticator
- conf :ShibRpcConfAndMapping
- log :Logger
+ SimpleHandleAuhenticator (params :AuthenticatorInitParams):
+ getSupportedProtocols ():String[]
+ initiateAuthentication (protocol :String):AuthDialog
+ getNextToken ():byte[]
ShibAttribute
- name :String
- value :String
+ ShibAttribute (n:String ,v:String):
+ getName ():String
+ getValue ():String
ShibRpcConf
- req :int
- shibRpcElements :ShibRpcElement[]
+ ShibRpcConf (r:int ,els :ShibRpcElement[]):
+ getReq ():int
+ getShibRpcElements ():ShibRpcElement[]
Group
- gid :String
+ Group (g:String):
+ getGid ():String
+ toString ():String
+ hashCode ():int
Policy
- id :String
- attributes :ShibAttribute[]
+ Policy (id :String ,a:ShibAttribute[]):
+ getId ():String
+ getAttributes ():ShibAttribute[]
ShibMapping
- id :String
- policies :Policy[]
- groups :Group[]
+ ShibMapping (id :String ,p:Policy[] ,g:Group[]):void
+ getId ():String
+ getPolicies ():Policy[]
+ getGroups ():Group[]
ShibRpcConfAndMapping
- conf :ShibRpcConf
- mapping :ShibMapping[]
+ ShibRpcConfAndMapping (c:ShibRpcConf ,m:ShibMapping[]):
+ getConf ():ShibRpcConf
+ getMapping ():ShibMapping[]
+ getGroupsForPolicy (policy :Map):Group[]
ShibTrustDB
- DTD_SHIBDB_0_1_PUBLIC_ID:String = "-//MOCCA//H2O Shib Database//0.1"
- log :Logger
+ reqAny :int
+ reqAll :int
+ parseTrustDB (source :InputSource ,eh:ErrorHandler):ShibRpcConfAndMapping
- parseShibTrustDB (el:Element):ShibRpcConfAndMapping
- parseShibRpcConf (el:Element):ShibRpcConf
- parseShibRpc (el:Element):ShibRpcElement
- parseMapping (el:Element):ShibMapping[]
- parseMappingEl (el:Element):ShibMapping
- parsePolicies (el:Element):Policy
- parseShibAttributes (el:Element):ShibAttribute
- parseGroups (el:Element):Group
&lt;&lt; exception &gt;&gt;
ShibTrustDBException
- serialVersionUID :long = 1L
+ ShibTrustDBException ():
+ ShibTrustDBException (msg :String):
+ ShibTrustDBException (t:Throwable):
+ ShibTrustDBException (msg :String ,t:Throwable):
SimpleHandleCredential
~PROTOCOL_V1:String = "rauth.h2o.shibboleth 1.0"
~protocols :String[] = new String[] { PROTOCOL_V1 }
- handle :String
- idpUrl :String
- delegate :boolean
+ SimpleHandleCredential (handle :String ,idpURL :String ,delegate :boolean):
+ SimpleHandleCredential (handleAndIdpURL :String ,delegate :boolean):
~staticGetSupportedProtocols ():String[]
+ getPublicCredential ():Object
+ getSupportedProtocols ():String[]
+ initiateAuth (protocol :String):AuthDialogClient
+ isRequired ():boolean
+ destroy ():void
+ isDestroyed ():boolean
+ readFrom (dis :DataInputStream):SimpleHandleCredential
+ writeTo (dos :DataOutputStream):void
+ getHandle ():String
+ isDelegate ():boolean
+ getIdpURL ():String
ShibDialogClientV1
- done :boolean
+ getNextToken ():byte[]
+ doPhase (token :byte[]):void
ShibDialogV1
~detailMessage :String
~status :int
~principals :Set
~credential :SimpleHandleCredential
~credentials :Set
+ getAuthenticatedPrincipals ():Set
+ getDelegatedCredentials ():Set
+ getDetailMessage ():String
+ getPublicCredentials ():Set
+ getStatus ():int
+ doPhase (token :byte[]):void

Figure 5.2. MOCCA Authenticator - H2O plugable authenticator providing Shibboleth authentication/authorization mechanism for MOCCA - UML diagram

5.3. MOCCA Policy Distribution Point and it's Client

This section features design of a classes related to the Policy Distribution Point and it's client. Policy Distribution Point features 2 classes - the MoccaPDistP is just providing basic XML-RPC service functionality and the PDistP exposes remote methods. This methods are consumed by e. g. mentioned here PDistP Client. The classes are shown on Fig. 5.3 .

All remote methods are divided into 3 groups:

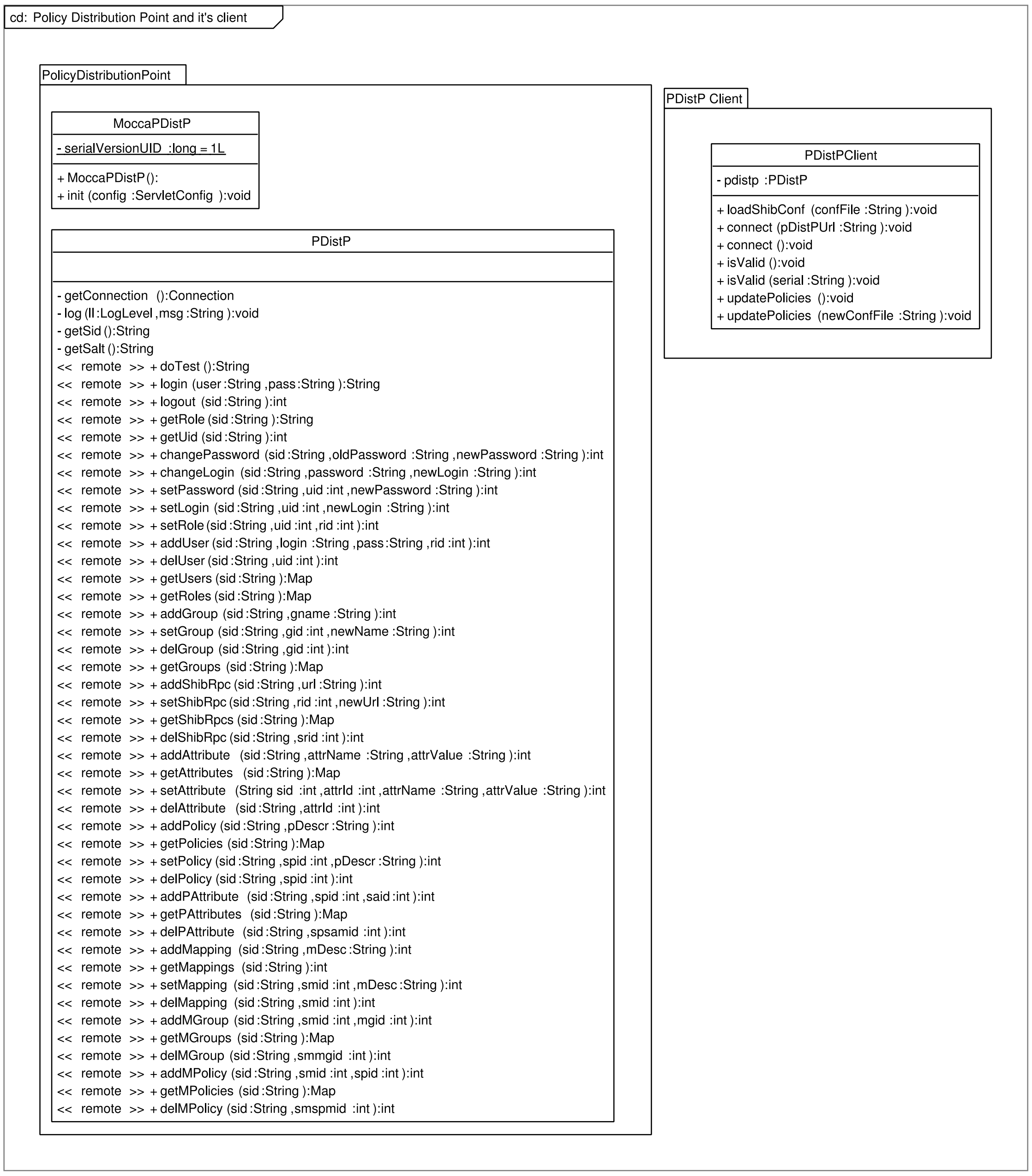
1. available for non-authenticated users (methods login() used for user login and doTest() to test interface)
2. available for all authenticated users (methods reading policies, reading and updating sessionID owners information and logout())
3. available just for administrators (changing policies, other users information, as well as adding and removing users)

Methods could access and modify various elements stored by the PDistP such as:

- local credentials - user id (uid), login, password and role,
- available shibRPCs settings
- simple MOCCA ShibTrustDB elements - Shibboleth attributes and MOCCA groups,
- more complex elements - policies that groups individual attributes,
- the most complex elements - mappings that maps policies to groups

The final Shib.xml file is contains collection of mappings and shibRpcs.

PDistP Client contains single class PDistPClient that is capable of validating if policies are current and downloading new ones when required.



```
MoccaPDistP
- serialVersionUID :long = 1L
+ MoccaPDistP ():
+ init (config :ServletConfig ):void

PDistP

- getConnection ():Connection
- log (ll :LogLevel ,msg :String ):void
- getSid ():String
- getSalt ():String
<< remote >> + doTest ():String
<< remote >> + login (user :String ,pass :String ):String
<< remote >> + logout (sid :String ):int
<< remote >> + getRole (sid :String ):String
<< remote >> + getUid (sid :String ):int
<< remote >> + changePassword (sid :String ,oldPassword :String ,newPassword :String ):int
<< remote >> + changeLogin (sid :String ,password :String ,newLogin :String ):int
<< remote >> + setPassword (sid :String ,uid :int ,newPassword :String ):int
<< remote >> + setLogin (sid :String ,uid :int ,newLogin :String ):int
<< remote >> + setRole (sid :String ,uid :int ,rid :int ):int
<< remote >> + addUser (sid :String ,login :String ,pass :String ,rid :int ):int
<< remote >> + delUser (sid :String ,uid :int ):int
<< remote >> + getUsers (sid :String ):Map
<< remote >> + getRoles (sid :String ):Map
<< remote >> + addGroup (sid :String ,gname :String ):int
<< remote >> + setGroup (sid :String ,gid :int ,newName :String ):int
<< remote >> + delGroup (sid :String ,gid :int ):int
<< remote >> + getGroups (sid :String ):Map
<< remote >> + addShibRpc (sid :String ,url :String ):int
<< remote >> + setShibRpc (sid :String ,rid :int ,newUrl :String ):int
<< remote >> + getShibRpcs (sid :String ):Map
<< remote >> + delShibRpc (sid :String ,srid :int ):int
<< remote >> + addAttribute (sid :String ,attrName :String ,attrValue :String ):int
<< remote >> + getAttributes (sid :String ):Map
<< remote >> + setAttribute (String sid :int ,attrId :int ,attrName :String ,attrValue :String ):int
<< remote >> + delAttribute (sid :String ,attrId :int ):int
<< remote >> + addPolicy (sid :String ,pDescr :String ):int
<< remote >> + getPolicies (sid :String ):Map
<< remote >> + setPolicy (sid :String ,spid :int ,pDescr :String ):int
<< remote >> + delPolicy (sid :String ,spid :int ):int
<< remote >> + addPAttribute (sid :String ,spid :int ,said :int ):int
<< remote >> + getPAttributes (sid :String ):Map
<< remote >> + delPAttribute (sid :String ,spsamid :int ):int
<< remote >> + addMapping (sid :String ,mDesc :String ):int
<< remote >> + getMappings (sid :String ):int
<< remote >> + setMapping (sid :String ,smid :int ,mDesc :String ):int
<< remote >> + delMapping (sid :String ,smid :int ):int
<< remote >> + addMGroup (sid :String ,smid :int ,mgid :int ):int
<< remote >> + getMGroups (sid :String ):Map
<< remote >> + delMGroup (sid :String ,smmgid :int ):int
<< remote >> + addMPolicy (sid :String ,smid :int ,spid :int ):int
<< remote >> + getMPolicies (sid :String ):Map
<< remote >> + delMPolicy (sid :String ,smspmid :int ):int

PDistPClient
- pdistp :PDistP
+ loadShibConf (confFile :String ):void
+ connect (pDistPUrl :String ):void
+ connect ():void
+ isValid ():void
+ isValid (serial :String ):void
+ updatePolicies ():void
+ updatePolicies (newConfFile :String ):void
```

Figure 5.3. Policy Distribution Point and it's client supporting storing, managing and updating MOCCA local policies - UML diagram

5.4. Summary

This chapter focused on designing key components that needed to be implemented to provide required functionality. Some aspects mentioned here - like detailed description of the Shib.xml configuration file is provided in chapter 6 in more details.

Chapter 6

Description of Implementation

This chapter shows the most interesting concepts involved the development. It includes general information about chosen programming languages, important software libraries and a bit more detailed description of most challenging implementation tasks.

6.1. Overview of Implemented Components

As required by the previous analysis all mandatory system components were implemented. This section enumerates and describes this components.

1. ShibIdpClient - a Java library that provides convenient interface for integration of non-Web tools with a Shibboleth IdP,
2. ShibIdpCliClient - a Java tool being a command line interface for the ShibIdp-Client,
3. MOCCA Shibboleth Authenticator - pluggable authenticator module for H2O, suppling a Shibboleth based authentication/authorization mechanism,
4. MOCCA Policy Distribution Point - a Java, XML-RPC based service supply remote methods for managing and requesting local MOCCA policies,
5. MOCCA Policy Distribution Client - a Java, XML-RPC client that consumes PDistP's remote interface and provides functionality required to update MOCCA policies,
6. MOCCA Policy Distribution Point Administrator's Panel - a PHP web application and XML-RPC client for PDistP allowing policies management.

6.2. General Implementation Concepts

As already mentioned, the goal of this section is to describe which programming languages were chosen and why, as well as to enumerate software libraries that were essential for creation of the solution.

6.2.1. Programming Languages

The Java programming language was chosen as a basic one for most of the described development. Sun's JRE version 1.6 was used as the run-time environment. The only exception were made for PDistP Administrator's Panel which has been written in PHP. There are several reasons for choosing Java for the standalone part of the software:

- Java Virtual Machine present on many platforms made the software portable, which is highly practical especially in case of grid systems where the nodes might have various software and hardware architecture, even quite unusual,
- Java in it's present version is a fine programming language allowing swift creation of maintainable code,
- Existence of good quality IDE software for Java especially the Eclipse platform that was used to create all Java components,
- Quite good efficiency sufficient for the system needs,
- Good software engineering tools (Maven)

For web part of the software PHP language have been chosen mostly to show that it is possible to use different commonly used technologies to access PDistP via XML-RPC (Java was used for client and PHP for administrator's panel). PHP is also a reliable and very well supported technology both from perspective of existing software libraries as well as existing IDE (also for PHP Eclipse platform with PDT extension were used).

6.2.2. Most Important Software Libraries

The developed software cannot be built without a help of several existing software libraries, which are described in this section.

- OpenSAML - one of the basic libraries used by ShibIdpClient, allowing to validate Shibboleth SAML assertions signatures and parse them to extract needed information (like handle),
- Xerces - as SAX XML parser used by ShibIdpClient to parse XHTML website returned by SSO, containing Base64-encoded SAML assertion,
- Bouncy Castle - a powerful cryptographic library. Part of this library is currently used as Base64 decoder in ShibIdpClient,
- Classess from javax.net.ssl.* are used to provide HTTPS connection to the SSO required by the Shibboleth implementation guidelines to provide confidentiality of transmitted data,

- Log4j - java logging library used in various components of the solution to simplify logging process,
- Redstone XML-RPC library - used for Java components of Policy Distribution Point to support server and client for XML-RPC service,
- MySQL JDBC driver - official MySQL driver, used by Policy Distribution Point to access MySQL RDBMS used to store both it's credentials as well as policies.

6.3. Challenging Implementation Tasks

This section shows some of the more challenging tasks that had to be resolved during implementation of the solution. This tasks includes both critical functionalists - a ability to access IdP without any web browser, providing Shibboleth authenticator compatible with H2O and creating a solution for distributing MOCCA local policies.

6.3.1. Non-web IdP Authentication

The first and the most important requirement for the customization of Shibboleth was to enable ability for non-web tools to authenticate to Shibboleth Identity Provider. The Performed research didn't provide any existing solutions for such use case as most Shibboleth implementations were purely web-based. Other cases contain some kind of web gateway that allow getting Shibboleth handle or they support a shell client via web (enabling running non-web applications just on predefined nodes).Yet another ones were just producing credentials for other security frameworks like GridShib for GSI. This prompted the need to create such a solution from scratch. The solution needs to emulate a session between a user using web browser to access Identity Provider and the SSO itself. The implementation requirement was that the authentication method for IdP, which is not defined by Shibboleth 1.3 standard, must be HTTP basic authentication for all the partners. This simplifies a bit creation of the solution as it do not need to parse various complicated authentication forms that might be used instead. The library created for this purpose still needs to perform the folloing:

1. Use HttpsURLConnection class to securely connect to the SSO,
2. Calculate and set HTTP "Authorization" header to perform basic HTTP authentication,
3. Download the returned web page,
4. Parse it to find an input field with "name" property set to "SAMLResponse" containing Base64 encoded SAML Assertion,
5. Decode the assertion ,
6. Use OpenSAML library to validate the SAML end to extract the handle,
7. Return the handle to the library user.

The library has been then wrapped with a piece of UI code to create ShibIdpCliClient - simple command line tool solving described problem of non-web authentication. Also the library might be directly integrated with other software by their authors as it has been successfully done in case of EPE and EMI.

6.3.2. Integration of the Authenticator with H20

The implemented solution keeps H2O policies (Policy.xml) intact but completely replaces Users.xml with Shib.xml specifying ShibRPC configuration that is used by the authenticator to get users attributes based on handle that is supported by MOCCA client during connection. It also provides policies for dynamic mapping of users with specific attributes to groups. Then privileges might be controlled as in standard solution.

An example of such Shib.xml file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE shibTrustDB PUBLIC "-//MOCCA//H2O Shib Database//0.1"
 "shibDB-0.1.dtd">

<shibTrustDB version="2009-07-15 01:36:12"
  PDistPUrl="https://localhost:8080/PDistP/PDistP">
    <shibRpcConf require="any">
      <shibRpc id="0"
         url="http://virolab.cyfronet.pl:9080/ShibRPC-1.2.2/ShibRpcServlet" />
    </shibRpcConf>
    <shibMapping>
       <mapping id="0">
          <policy id="0">
             <attribute name="homeOrganization" value="Cyfronet" />
             <attribute name="virolabRole" value="Researcher" />
          </policy>
          <policy id="1">
             <attribute name="homeOrganization" value="GridwiseTech" />
          </policy>
          <group gid="ShibGroup" />
          <group gid="TestGroup" />
       </mapping>
       <mapping id="1">
          <policy id="0">
             <attribute name="homeOrganization" value="Cyfronet DEMO" />
          </policy>
          <group gid="EvilUsers" />
       </mapping>
    </shibMapping>
</shibTrustDB>
```

In the shown here examples both Cyfronet users with a role of Researcher as well as all GridwiseTech users would be mapped to ShibGroup and TestGroup that might be configured in Policy.xml to grant wide access to the MOCCA. On the other hand "DEMO" user would be mapped to EvilUsers group that could get highly restricted access (or even none).

6.3.3. Distributing Local Policies for MOCCA

Finally, it was needed to create flexible method for storing and delivering policies based on Shibboleth attributes to distributed MOCCA installations. For this purpose Policy Distribution Point was created. As a communication protocol for PDistP and its satellites (its client for MOCCA and tool for administrators) the XML-RPC were chosen. The main reason for that choice is that it met all requirements for such solution, and that authenticator already needs to support it for cooperation with external service (ShibRPC). The core of the system was implemented as XML-RPC service that provides methods for user authentications, policy modifications, verifying if policies are up to date and getting new policies.

After authentication user gets Session ID which is later used as parameter for other exposed methods to provide authentication and authorization. System checks if Session ID is valid and if it's owner has sufficient role for the operation. Session ID might be used unless it expires or user executes logout() method.

All information is stored in MySQL, with InnoDB engine because of the necessity to use more advanced MySQL features like triggers (for tracking changes in policies) and foreign keys. PDistP uses standard MySQL JDBC driver for connection to the DB.

Policies on each node are updated by small Java applications being clients for the PDistP (using XML-RPC client functionality provided by Redstone library).

For administration of the solution a PHP web application was written that allows managing of users and policies in PDistP. This application's authentication system is transparent as it delegates this functionality to PDistP by executing remote authentication methods, checking if user have sufficient role (just for user's convenience, because otherwise PDistP would still refuse to execute methods with unprivileged Session ID) and storing received Session ID in a cookie. Because all relevant information are configured remotely at PDistP this application do not need own database backend.

6.4. Summary

This chapter summarized the implementation of the custom solutions that were needed to be created to augment standard Shibboleth framework. The solutions described here in combination with the existing ones both created as native part of Shibboleth as well as by other VL partners, were integrated and provided a complete security solution that met all requirements mentioned earlier in this thesis.

Chapter 7

Validation and Evaluation of the Security System

This chapter is to provides a description of tasks that were perform to validate the security of a created solution as well as to evaluate it's efficiency. For a security system, the validation is a very important phase of the whole process of software development. That is why all key security components have been validated. Next, the performance have been evaluated, as it was also one of the important factors for the solution. Finally integration and user interface validation was performed.

7.1. Security Audit

Automatic security auditing has been performed on key system components specified below. For each component the description of the process as well as results are described in it's section.

7.1.1. ShibIdpClient

For testing purpose, the tool was provided with a valid and invalid credentials. The results were completely satisfactory, as no attempts to request handle without valid credentials where successful. Additionally, getting access to IdP after substituting a valid certificate with an invalid one were tried, mimicking situation in which user is redirected to a fake service. As planned, the connection attempt failed due to untrusted certificate. Attempts and results are shown in Tab. 7.1 .

Credentials	Certificate	Result
valid	valid	access granted / handle returned
invalid	valid	access denied
valid	invalid	access denied (untrusted cert. issuer)
invalid	invalid	access denied (untrusted cert. issuer)

Table 7.1. ShibIdpClient security audit results, during audit both valid and invalid credentials as well as certificates were provided

7.1.2. MOCCA Shib Authenticator

JUnit test scenarios where used in the case of Shib authenticator. The test cases were generated that tried to authenticate with handles that were either invalid or valid with various attributes. Results for each case were summarized in Tab. 7.2 .

Handle	Trusted HO	MOCCA Policies	Result
invalid	-	-	access denied
valid	no	no	access denied
valid	yes	no	access denied
valid	yes	yes	access granted

Table 7.2. MOCCA Authenticator security audit results, the first column informs if a tested handle were valid, the second if HO was trusted by ShibAuthApi and the third if handle's attributes were acceptable by MOCCA local policies

Those validations were also successful - user was able to gain access only in the last case and the mapped group was right.

7.1.3. MOCCA Policy Distribution Point

For Policy Distribution Point a special test client as created with strict checks for each exposed method. It provides PDistP with different credentials both invalid, as well as valid with different roles. The results are presented in Tab. 7.3 .

The results showed in the table confirmed that Policy Distribution Point is secure.

7.2. Performance evaluation

For the same key components a basic performance evaluation was performed to verify if these components might be used in a production environment. This section shows the test environment as well as the results of performed benchmarks for each of the elements mentioned here.

7.2.1. Test environment

All performance evaluations has been run on the computer system consistent of 2 machines with the following specification each:

Credential	**Role**	**Action**	**Result**
invalid pass	-	auth	failed
valid pass	-	auth	succeed
invalid sid	-	normal cmd	failed
invalid sid	-	admin cmd	failed
valid sid	user	normal cmd	succeed
valid sid	user	admin cmd	failed
valid sid	admin	normal cmd	succeed
valid sid	admin	admin cmd	succeed

Table 7.3. MOCCA Policy Distribution Point security audit results, various credentials (password for login() command, and session id for others) with different roles were tested against authentication command, normal commands as well as administrators (admin) restricted commands

- CPU: 2xIntel Xeon 5150 (2.66 GHz)
- Physical RAM: 4 GB
- SWAP: 8 GB
- Connectivity (to all tested components): 1 GBit Ethernet

7.2.2. ShibIdpClient

Performance of this component has been tested by measuring time needed by the software to request Shibboleth handle for specified user. This operation has been repeated 10 times, and then average time has been calculated. Measurement results (in milliseconds) are presented in Tab. 7.4 .

sample	1	2	3	4	5	6	7	8	9	10
time, ms	869	777	785	922	782	811	858	757	774	770

Table 7.4. ShibIdpClient benchmark results - time measurements for handle requesting process measured 10 times

Based on this results the average time needed to request a handle is: 810.5 ms . Because this process do not need to be repeated until the handle expires (8 hours), the authentication time not exceeding 1 s is acceptable for production use in described use case.

7.2.3. MOCCA Shibboleth Authenticator

In the case of MOCCA Shibboleth authenticator a special test has been created measuring the time of the typical phases of the life cycle of 2 components. Below those phases and their symbols (in brackets) are presented:

- Preparing builder (BT)
- Deploying component 1 (C1)

- Deploying component 2 (C2)
- Running components (RUN)

Each series has been repeated 10 times. The results (in milliseconds) and average values has been put into Tab. 7.5 .

sample	BT, ms	C1, ms	C2, ms	RUN, ms
1	726	68	49	568
2	749	76	53	570
3	737	79	61	587
4	759	71	51	603
5	755	77	50	596
6	730	87	50	585
7	701	70	50	600
8	750	79	68	581
9	729	69	53	594
10	738	70	48	593
AVG	737,4	74,6	53,3	587,7

Table 7.5. MOCCA Shibboleth Authenticator benchmark results - each column (exept first - sample number) represents time interval for predefined phasee of deployment process

The authenticator is used only during first phase (BT), since later on a session is opened between the H2O client and the kernel. By comparing the times of deployment of builder (BT) and component pluglets (C1 and C2) we can conclude that the time used by the authenticator is less then 700 ms. It can be considered as acceptable taking into account the steps required (including connecting with IdP via ShibRPC services).

7.2.4. Policy Distribution Point

Policy Distribution Point performance has been tested by measuring time of 10 operations called on this service. As before, the calculations has been repeated 10 times. Results are attached in Tab. 7.6 .

sample	1	2	3	4	5	6	7	8	9	10
time, ms	1365	630	1000	822	617	1030	634	785	589	784

Table 7.6. PDistP benchmark results - 10 measurements of the sequences of 10 operations performed on PDistP

Average time for 10 operations is: 825,6 ms
So average time for 1 operation is: 82,56 ms
This result prove that the solution is efficient enough for the required task.

7.3. Validation of the Integration

After the integration the components described in subsequent sections were validated to function properly.

7.3.1. ShibIdpClient

It has been validated together with the components it has been integrated with: ShibIdpCliClient, EPE and standalone version of EMI. This validation has been performed with the help of both Cyfronet credentials as well as test credentials that have been provided by other partners. The aim for checking foreign credentials was to verify that the software is functioning properly as part of the whole infrastructure combined with various federated partner's Home Organizations including Cyfronet, GridwiseTech and HLRS.

7.3.2. MOCCA authenticator and PDistP

Integration of the following components also have been thoroughly checked:

- MOCCA Authenticator with MOCCA/H2O itself,
- Policy Distribution Point Client with MOCCA (through Shib.xml files),
- Policy Distribution Point with it's client,
- Policy distribution point with administrator panel.

All the mentioned elements were found out to be working with each other.

7.4. Manual user interface test

Elements supporting user interface has been repeatedly manually tested until they were free from errors.

7.4.1. ShibIdpCliClient

A command line interface for ShibIdpClient is a simple text based user interface. Because of it's simplicity just basic tests were enough to eliminate possible mistakes (like spelling mistakes).

7.4.2. Administrator's Panel for PDistP

Policy Distribution Point Administrator's Panel as a web application is a bit more complicated. Test required manually checking each sub-page and correcting errors that were found out.

7.5. Summary

Validations and evaluations described in this chapter were performed to check if each security component created as a part of the security system is free from security flaws, provide sufficient performance as well as could be integrated with external software without any problems. As described in this chapter, those goals were fully achieved. Security of the components has been confirmed by audit results described here.The performance of most components is quite good, and even the worst result for requesting a handle is fully acceptable as this operation is performed quite infrequently relatively to others. The integration of the system in aspect of integration with components created by other ViroLab developers as well as with other HOs' infrastructure has also been successfully performed.

Chapter 8

Conclusions and Further Work

This chapter presents and summarizes the work that have been done, shows how the goals were achieved and describes the plans for further research related to the subject. The work described in this thesis had to be done to support a non-standard and novel security infrastructure for federation of organizations cooperating to create solution for distributed collaborative research environment. Part of the system including Portal security where implemented by others with standard Shibboleth mechanism. The main task of the work in the scop of this thesis is to create compatible infrastructure for other (non-web) part of the system.

8.1. Achieved goals

This section summarizes goals that were planned to be achieved before creation of this work, and which all were reached:

1. analysis of existing security solutions and frameworks including PKI, TLS, SAML, GSI, Shibboleth ShibGrid, GridShib and OpenID as well as required cryptographic algorithms - has been performed and documented in chapter 2
2. identification of elements that might be useful for creation of the complete solution like direct Shibboleth usage or supplementing Shibboleth with GSI through GridShib - were discussed in chapter 2
3. creation of a formal threat model for the infrastructure - were described in section 3.2; As the result it was found out and described that getting unauthorized access to credential database (both read-write and just read-only) would critically compromise security of the system; It also was determined that getting such access to network or computational resources might also lead to serious risks (DDoS attack or password cracking attempts),

4. enumeration system requirements - were done in sections 3.2 (security requirements like authentication, authorization, credential delegation, confidentiality and integrity) and 3.3 (other requirements like user friendliness and scalability)
5. discussion of architecture of the system - were presented in chapter 4
6. design and implementation of following system components ShibIdpClient, ShibIdpCliClient, MOCCA Shibboleth Authenticator, Policy Distribution Point (PDistP), it's client and administrator's panel - were done in chapters 5 and 6
7. performing system validation and evaluation - were successfully done in chapter 7

The conclusions are tha it is possible to use Shibboleth not only for Web, but it requires adding new tools and solving problem with policy distribution.

8.2. Plans for further research

The work described in this thesis will be continued. First, the work will be focused on further augmenting of already created parts of the system. Especially adding features for ShibIdpCliClient allowing secure, fully automatic distribution of configuration and certificates. Also it would be interesting to provide Policy Distribution Point with mechanisms enabling it to run in master slave mode to increase scalability and fault tolerance of the system even further.

Next, the support for newest version of the third-party software components (especially Shibboleth 2.0) should be provided to ensure long term usability of the created solution. Also adding support for interoperability with other security technologies is worth doing some research.

Finally, quite important might be a need to adapt the solution for other projects then ViroLab, for example to support the security for VL part of the PL-Grid project, where the virtual laboratory continues to be developed.

Bibliography

[1] ViroLab Project Consortium. ViroLab, 2009. `http://virolab.org/`.

[2] PL-Grid Project Consortium. PL-Grid, 2009. `http://www.plgrid.pl/`.

[3] Marian Bubak, Tomasz Gubala, Maciej Malawski, Bartosz Balis, Wlodzimierz Funika, Tomasz Bartynski, Eryk Ciepiela, Daniel Harezlak, Marek Kasztelnik, Joanna Kocot, Dariusz Krol, Piotr Nowakowski, Michal Pelczar, Jakub Wach, Matthias Assel, and Alfredo Tirado-Ramos. Virtual Laboratory for Development and Execution of Biomedical Collaborative Applications. In *Proceedings of the Twenty-First IEEE International Symposium on Computer-Based Medical Systems, June 17-19, 2008, Jyväskylä, Finland*, pages 373–378. IEEE Computer Society, 2008.

[4] Peter M. A. Sloot, Alfredo Tirado-Ramos, Ilkay Altintas, Marian Bubak, and Charles Boucher. From Molecule to Man: Decision Support in Individualized E-Health. *Computer*, 39(11):40–46, 2006.

[5] Eryk Ciepiela, Joanna Kocot, Tomasz Gubala, Maciej Malawski, Marek Kasztelnik, and Marian Bubak. GridSpace Engine of the ViroLab Virtual Laboratory. In *Proceedings of Cracow Grid Workshop 2007*, pages 53–58. ACC CYFRONET AGH, 2008.

[6] ViroLab team at CYFRONET. The ViroLab Virtual Laboratory Website, 2009. `http://virolab.cyfronet.pl`.

[7] Wlodzimierz Funika, Daniel Harezlak, Dariusz Krol, and Marian Bubak. Environment for Collaborative Development and Execution of Virtual Laboratory Applications. In Marian Bubak, G. Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *Computational Science - ICCS 2008, 8th International Conference, Kraków, Poland, June 23-25, 2008, Proceedings, Part III*, volume 5103 of *Lecture Notes in Computer Science*, pages 246–458. Springer, 2008.

[8] Internet 2 Project. Shibboleth, 2009. `http://shibboleth.internet2.edu/`.

[9] D. Kurzyniec et al. Towards Self-Organizing Distributed Computing Frameworks: The H2O Approach. *Parallel Processing Lett.*, 13(2):273–290, 2003.

[10] Maciej Malawski, Marian Bubak, Michal Placek, Dawid Kurzyniec, and Vaidy Sunderam. Experiments with distributed component computing across grid boundaries. In *Proceedings of HPC-GECO/COMPFRAME Workshop in Conjunction with HPDC'06*, pages 109–116, 2006.

[11] UserLand Software. Xml-rpc, 2009. `http://www.xmlrpc.com/`.

[12] Jan Meizner, Maciej Malawski, Syed Naqvi, and Marian Bubak. Threat Model for MOCCA Component Environment. In *Proceedings of Cracow Grid Workshop 2008*, pages 94–102. ACC CYFRONET AGH, 2009.

[13] National Institute of Standards and Technology. *FIPS PUB 46-3: Data Encryption Standard (DES).* National Institute for Standards and Technology, October 1999.

[14] National Institute of Standards and Technology. *FIPS PUB 197: Advanced Encryption Standard (AES).* National Institute for Standards and Technology, November 2001.

[15] RSA Laboratories. *PKCS #1 v2.1: RSA Cryptography Standard.* RSA Laboratories, June 2002.

[16] E. Rescorla. Diffie-Hellman Key Agreement Method (RFC 2631). `http://www.ietf.org/rfc/rfc2631.txt`.

[17] Ronald L. Rivest. The MD5 Message-Digest Algorithm (RFC 1321). `http://www.ietf.org/rfc/rfc1321.txt`.

[18] National Institute of Standards and Technology. *FIPS PUB 180-2: Secure Hash Standard.* National Institute for Standards and Technology, August 2002.

[19] National Institute of Standards and Technology. *FIPS PUB 198: The Keyed-Hash Message Authentication Code (HMAC).* National Institute for Standards and Technology, March 2002.

[20] National Security Agency. *National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information.* National Security Agency, June 2003.

[21] International Telecommunication Union. *ITU-T Recommendation X.509:Information technology Open systems interconnection The Directory: Public-key and attribute certificate frameworks.* International Telecommunication Union, March 2000.

[22] H. Krawczyk, M. Bellare, and R. Cannetti. HMAC: Keyed-Hashing for Message Authentication (RFC 2104). `http://www.ietf.org/rfc/rfc2104.txt`.

[23] Joel Weise. *Public Key Infrastructure Overview.* Sun Microsystems, Inc., August 2001.

[24] International Telecommunication Union. *ITU-T Recommendation X.680:Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.* International Telecommunication Union, July 1994.

[25] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2 (RFC 5246). `http://www.ietf.org/rfc/rfc5246.txt`.

[26] Frier, A. and Karlton, P. and Kocher, P. *The SSL 3.0 Protocol.* Netscape Communications Corp., November 1996.

[27] OASIS. Security Assertion Markup Language. `http://saml.xml.org/saml-specifications`.

[28] W3C. Simple Object Access Protocol (SOAP). `http://www.w3.org/TR/soap/`.

[29] Ian T. Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A Security Architecture for Computational Grids. In *ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[30] The EUGridPMA. Coordinating grid authentication in e-Science. `http://www.eugridpma.org/`.

[31] David Spence et al. ShibGrid: Shibboleth Access for the UK National Grid Service. In *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on*

e-Science and Grid Computing, page 75, Washington, DC, USA, 2006. IEEE Computer Society.

[32] Jim Basney, Marty Humphrey, and Von Welch. The MyProxy online credential repository. *Softw., Pract. Exper.*, 35(9):801–816, 2005.

[33] Tom Scavo and Von Welch. A Grid Authorization Model for Science Gateways. *Concurrency and Computation: Practice and Experience*, 2008. To appear.

[34] Matthias Assel, Piotr Nowakowski, and Marian Bubak. Integrating and accessing medical data resources within the ViroLab virtual laboratory. In Marian Bubak, G. Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *Computational Science - ICCS 2008, 8th International Conference, Kraków, Poland, June 23-25, 2008, Proceedings, Part III*, volume 5103 of *Lecture Notes in Computer Science*, pages 90–99. Springer, 2008.

[35] The Eclipse Foundation. Eclipse, 2009. `http://eclipse.org/`.

[36] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology - CRYPTO 2003*, Lecture Notes in Computer Science, pages 617–630. Springer, 2003.

[37] Jan Meizner, Maciej Malawski, Eryk Ciepiela, Marek Kasztelnik, Daniel Harezlak, Piotr Nowakowski, Dariusz Król, Tomasz Gubała, Włodzimierz Funika, Marian Bubak, Tomasz Mikołajczyk, Paweł Płaszczak, Krzysztof Wilk, and Matthias Assel. ViroLab Security and Virtual Organization Infrastructure. In Young Dou, Ralf Gruber, and Josef Joller, editors, *Advanced Parallel Processing Technologies 8th International Symposium, APPT 2009, Rapperswil, Switzerland, August 24-25, 2009 Proceedings*, volume 5737 of *Lecture Notes in Computer Science*, pages 230–245. Springer, 2009.

[38] Gunter Ollmann. The Phishing Guide: Understanding and Preventing Phishing Attacks. `http://www.technicalinfo.net/papers/Phishing.html`.

[39] Gunter Ollmann. The Pharming Guide: Understanding Preventing DNS-related Attacks by Phishers.

[40] Kevin Mitnick, William L. Simon, and Steve Wozniak. *The Art of Deception: Controlling the Human Element of Security.* Wiley, 2002.

[41] Internet 2 Project. Shibboleth Identity Provider Installation, 2009. `https://spaces.internet2.edu/display/SHIB/JKIdPInstall`.

[42] Michał Dyrda. Master of Science Thesis supervised by Marian Bubak: Security in Component Grid Systems, 2008.