Security and Trust Negotiation in Open Environments

Bachelor thesis

from
Sebastian Wittler
05.04.2004-05.08.2004

First Examiner: Prof. Dr. techn. Dipl.-Ing. Wolfgang Nejdl
Second Examiner: Prof. Dr.-Ing. Gabriele von Voigt
Supervisor: Dipl.-Inf. Daniel Olmedilla

Outline

- Trust Negotiation
- PeerTrust
- Tasks of this Bachelor thesis
  - authenticatesTo-predicate
  - Verification of the proof tree
  - Verification of credentials

Motivation for Trust Negotiation

- This identity-based establishment of trust (registering/login) has disadvantages:
  - Client unable to find out if the server can be trusted
  - Verifying the registration-data transmitted by the client
  - Often unrelevant information is required
  - Keeping track of all login-names and passwords
  - Not suitable for the Semantic Web
- The automatic Trust Negotiation-approach overcomes those restrictions

Trust Negotiation

- Based on properties which must be proven by credentials
- Sensitive resources (credentials, documents, services, ...) may be protected by policies
- Other parties must satisfy those policies by showing appropriate credentials before getting access to them
- Credentials are digital counterparts of real world ones (drivers license, credit card etc.)
- Software agents automatically negotiate trust by disclosing suitable policies and credentials
- This negotiation can consist of more than one step and is bidirectional, even third parties may be involved

Example for Trust Negotiation

- Customer wants to play Lotto online
  - He has a credit-card credential
  - This one is protected by a policy: Other party must show a license-credential signed by Toto Lotto
- Web page offers the possibility to play Lotto online
  - It has an account for each user
  - These are protected by a policy: Other party must show a credit-card credential

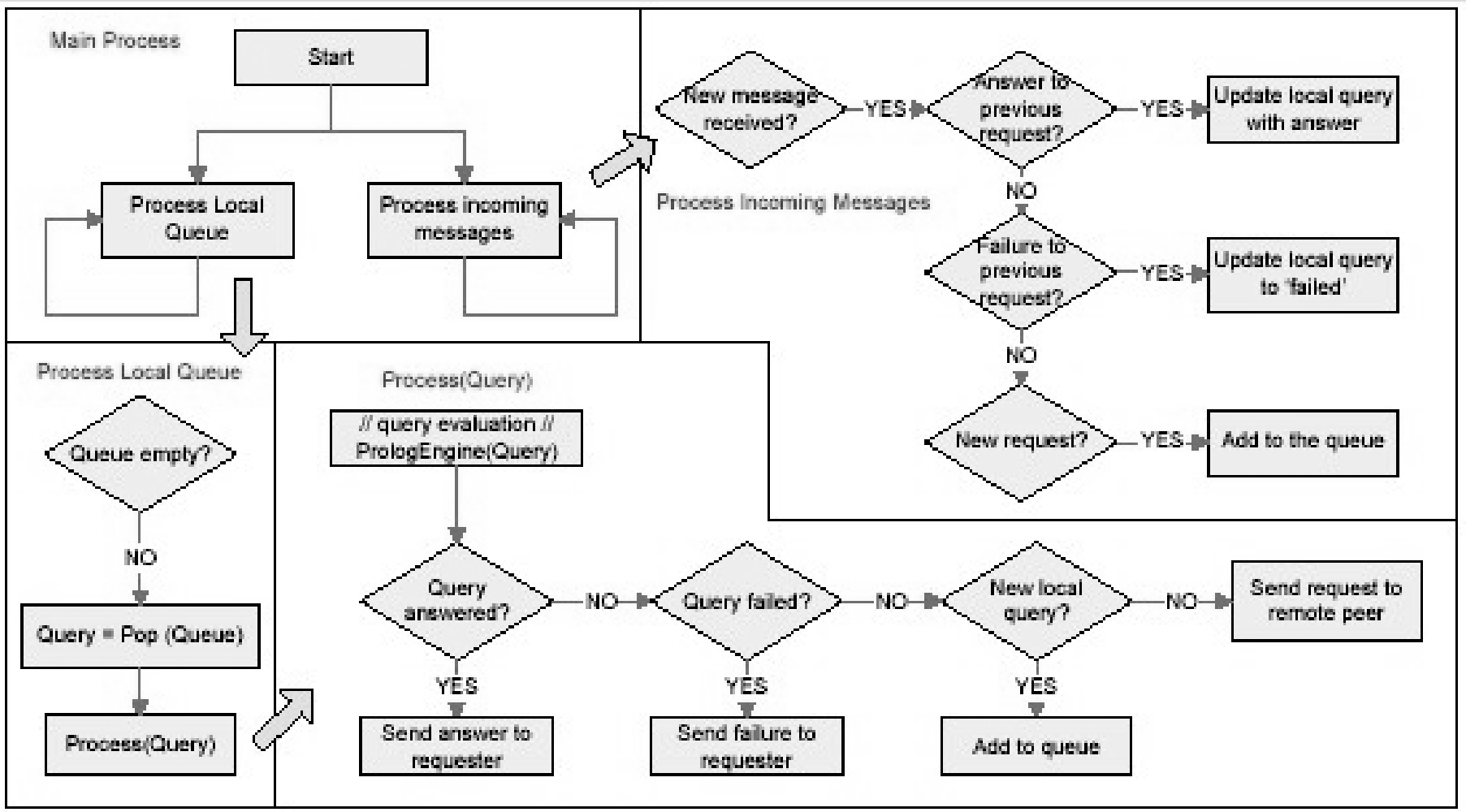
PeerTrust

- Trust Negotiation software
- In prototype phase
- Open source
  http://sourceforge.net/projects/peertrust/
- Language:
  - Based on first order Horn rules: a(x) ← b(x) , ... , c(x)
  - Issuer-argument (who must evaluate the literal)
    - lit@Issuer
  - Requester-argument (who has delegated the literal)
    - lit@Requester
  - Normal and signed rules
  - RDF-metadata can be used in policies

PeerTrust Implementation

- Written in Java
  - Available as application or signed applet
- Uses Prolog engine (Minerva)
  - Inference engine contains
    - PeerTrust policies
    - RDF metadata
    - credentials
- Communication over Secure Sockets (SSL/TLS)
- Credentials are X.509 certificates

Negotiation In PeerTrust



Tasks of Bachelor thesis

- Increase security in PeerTrust by
  - Implementation of authenticatesTo-predicate
    - Special predicate for authentication
    - Parameters specify how party must authenticate
  - Verification of the proof tree
    - Is the answer to a query correct?
  - Verification of credentials
    - Is the credential correct?
    - Also a part of the second task

authenticatesTo-predicate

- First task in Bachelor thesis
- Definition:
  - authenticatesTo(Identity,Party,Authority) @Requester
    - True if **Requester** can prove to **Party** that he possesses the identity **Identity** issued by authority **Authority**
- Implementation:
  - Authentication with X.509 certificates
  - Two approaches:
    - Use the TLS handshake
    - Manual authentication

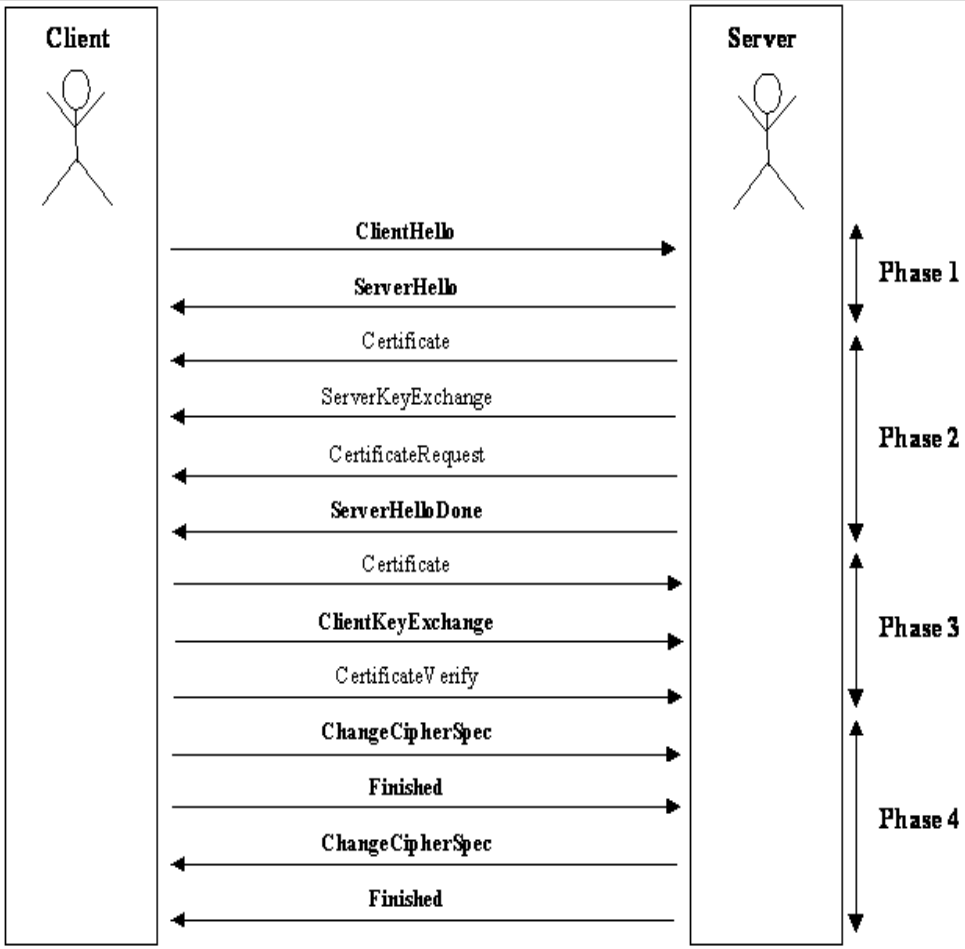
X.509 Certificates

- Common certificate format
- Used for authentication
- Entries:
  - Owner (Subject)-information
  - Issuer-information
  - Security entries:
    - Lifetime period
    - Public key of Subject
    - Digital signature
    - Algorithm names for hashing and signing
  - Extensions

Certificate Chains

- One single certificate may not be sufficient
  - Public key of Issuer missing
  - Maybe party has no valid certificate but certification path
- Sequence of certificates
- Subject must match Issuer of previous certificate
- Certificate (chain) typically signed by Certificate Authority (CA)

SSL/TLS Handshake



- SSL/TLS: protocols for secure connections
  - Authentication
  - Confidentiality
  - Data integrity
- Handshake: Trial to establish a secure connection
- Left figure:
  - Optional messages are plain
  - **Phase 1**: Exchange of parameters
  - **Phase 2**: Server may authenticate
  - **Phase 3**: Client may authenticate
  - **Phase 4**: Establishment of secure channel
- CertificateRequest-message
  - Tell Client that he must authenticate
  - Contains list of authorities the Server trusts
  - Ideal for authenticatesTo-predicate

Java and SSL/TLS

- SSL/TLS-support in Java: JSSE (Java Secure Socket Extension)
- e.g. javax.net.ssl- and java.security-packages
- Variants:
  - https
  - secure sockets
    - SSLSocket- and SSLServersocket-class
- Factory classes for extensibility
- Behaviour in handshake defined by
  - KeyManager-interface (which certificate (chain) should be presented?)
  - TrustManager-interface (which certificate (chain) should be trusted?)
  - Method setNeedClientAuth(boolean) in SSLServerSocket-class forces Client to authenticate

Custom TrustManager

- TrustManager for use with X.509 certificates
  - javax.net.ssl.X509TrustManager-interface
- Methods:
  - public void checkClientTrusted(X509Certificate[] chain,String authType) throws CertificateException
  - public void checkServerTrusted(X509Certificate[] chain,String authType) throws CertificateException
  - public X509Certificate[] getAcceptedIssuers()

authenticatesTo with TLS/SSL

- Server
  - Receive authenticatesTo-predicate
  - Send Client a special message with Identity, Authority and a port number
  - Create custom TrustManager which only accepts Authority and Identity
  - Create SSLServerSocket which waits for Client to connect
  - When Client has connected, the handshake will start
  - If the handshake fails (Exception will occur), so does the predicate
  - Otherwise, the predicate is satisfied
  - Close socket connection and SSLServerSocket
- Client
  - Receive special message from Server
  - Create an SSLSocket and connect to the specified port
  - Close socket connection

Disadvantages of TLS-approach

- For each client a SSLServerSocket
  - Imagine several clients authenticate at the same time
  - Memory usage and processing power not accurate
- Does not integrate well in the existing communication system of PeerTrust
  - Unflexible
- But:
  - Solution without TLS/SSL must provide manually
    - Certificate chain construction and transmission
    - Private key proof for Client

authenticatesTo without TLS

- Server
  - Receive authenticatesTo-predicate
  - Send Client a special message with Identity, Authority and a random text
  - Receive answer from Client
  - Encrypt decrypted random text with public key of Client
  - Compare result to random text sent (private key proof)
  - Check certificate chain if it satisfies the Authority- and Identity-parameters
  - Predicate is satisfied, if both verifications succeed, otherwise not
- Client
  - Receive special message from Server
  - Construct a suitable certificate chain
  - Decrypt random text with private key
  - Send back answer with certificate chain and decrypted random text

Private Key Proof

- Server creates random text

```
Random random=new Random();
byte enc_bytes[]=new byte[Math.abs(random.nextInt())%31+20];
random.nextBytes(enc_bytes);
String enc_str=new String(enc_bytes);
```

- Client decrypts this text with his private key

```
Signature signature=Signature.getInstance(CRYPT_ALGORITHM);
...
signature.initSign((PrivateKey)vectorPrivateKeys.elementAt(i));
signature.update(enc_str.getBytes());
byte dec[]=signature.sign();
```

- Server encrypts it with the Client's public key and compares it with the original text

```
Signature signature=Signature.getInstance(CRYPT_ALGORITHM);
signature.initVerify(certs[0].getPublicKey());
signature.update(enc_str.getBytes());
signature.verify(dec);
```

Certificate Chain Class

- New class for X.509 certificate chains implemented
- Attributes:
  - Array of X509Certificate-objects
- Methods:
  - Two methods for automatically constructing a chain
    - Simple algorithm
      - If issuer of chain is not specified
    - Complex algorithm
      - If issuer of chain is specified

Certificate Chain Construction

- **Algorithm in pseudo code**

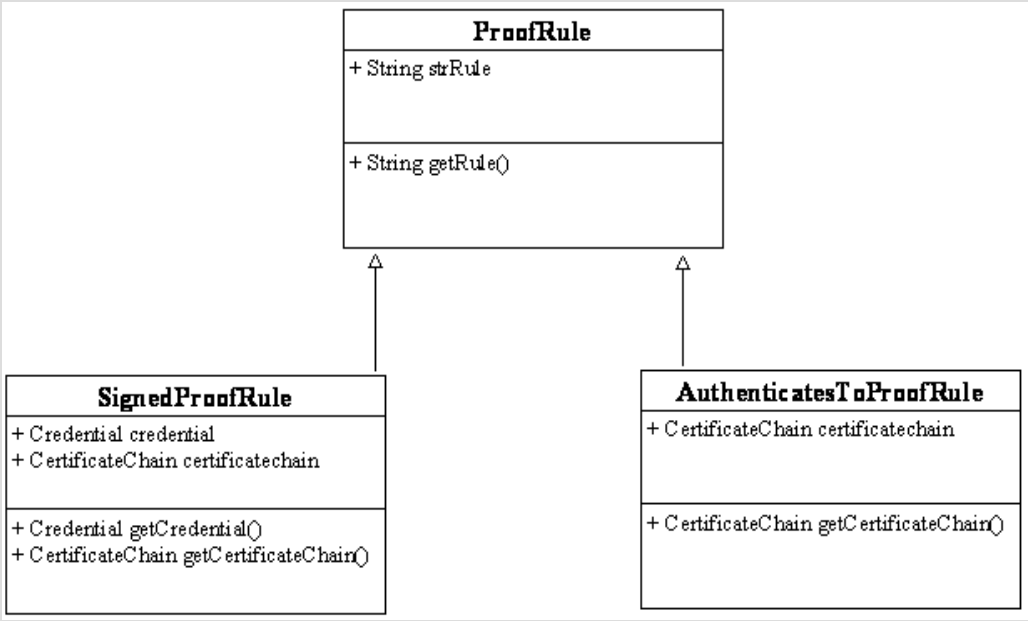
```
Vector vectorTree,vectorTemp,vectorCerts (out-parameter);
store all users certificates in the keystore which correspond to Identity to vectorTemp;
add vectorTemp to vectorTree;
if (vectorTemp is empty)
        return;
while(true) {
        if (vectorTree is empty)
                return;
        vectorTemp=last element of vectorTree;
        if (vectorTemp is empty) {
                remove last element from vectorTree and vectorCerts;
                continue;
        }
        transfer first certificate in vectorTemp to vectorCerts;
        String temp_issuer=certificate's issuer;
        if (temp_issuer==Authority)
                return;
        clear vectorTemp;
        add all valid certificates in the keystore to vectorTemp whose subject is temp_issuer;
        add vectorTemp to vectorTree;
}
```

Verification of the Proof Tree

- Second task of Bachelor thesis
- Can an answer be trusted?
  - Other party might lie
  - Answer might be manipulated
- Mechanism needed to verify answer
  - In PeerTrust, every answer or locally processed query has a proof tree
  - Proof tree consists of all used
    - rules/policies (normal and signed ones)
    - credentials
- Algorithm needed to verify proof tree

New Proof Tree Representation

- Previous proof tree was a String
  - Not suitable for signed rules/credentials
- New representation
  - Rules must be stored with appropriate credentials or certificate chains
  - Vector with objects of the following classes:
    - AuthenticatesToProofRule
      - For answer of authenticatesTo-predicate
    - ProofRule
      - For normal rules
    - SignedProofRule
      - For signed rules



Algorithm to verify Proof Tree

- The algorithm:

```
public static boolean isProofTreeOk(Tree tree,Configurator config) {
        ProofRule proofrule;
        try {
                MinervaProlog engine=new MinervaProlog(config);
                engine.loadFile("proof");
                Vector vector=tree.getProofRuleVector();
                for(int i=0;i<vector.size();i++) {
                        proofrule=(ProofRule)vector.elementAt(i);
                        if((proofrule instanceof SignedProofRule)&&
                                (!checkSignedProofRule((SignedProofRule)proofrule)))
                                return false;
                        else if((proofrule instanceof AuthenticatesToProofRule)&&
                                (!checkAuthenticatesToProofRule(AuthenticatesToProofRule)
                                proofrule,config)))
                                return false;
                        engine.execute("asserta("+proofrule.getRule()+")");
                }
                return engine.execute("proof("+tree.getLastExpandedGoal()+","+
                        tree.getRequester().getAlias()+")");
        }
        catch(Exception e) { ... }
        return false;
}
```

Check authenticatesTo-answer

- Algorithm in pseudo code

```
private static boolean checkAuthenticatesToProofRule(AuthenticatesToProofRule authrule) {
        X509Certificate certs[]=authrule.getCertificateChain().getCertificates();
        if(certs.length==0)
                return false;
        if ((subject of certs[0]!=parameter in answer)||(issuer of certs[certs.length-1]!=parameter
                in answer))
                return false;
        try {
                for(int i=0;i<certs.length;i++) {
                        certs[i].checkValidity();
                        if(i<certs.length-1)
                                certs[i].verify(certs[i+1].getPublicKey());
                        else
                                return checkLastCertificate(certs[i],config);
                }
        }
        catch(Exception e) {
        }
        return false;
}
```

Verification of Credentials

- Third task of Bachelor thesis
- In PeerTrust, Credentials are X.509 certificates
  - Credential-text in Subject.AlternativeNames-entry (extension)
  - Credentials appear in SignedProofRule-objects in the proof tree
  - Overlap with second task
- Algorithm for checking credentials needed

Algorithm to Verify Credentials

- Algorithm in pseudo code:

```
private static boolean checkSignedProofRule(SignedProofRule signedproofrule) {
        Credential cred=signedproofrule.getCredential();
        if(cred==null)
                return false;
        X509Certificate cert=certificate of cred;
        if (cert is expired)
                return false;
        if (credentials content doesn't fit to signed rule)
                return false;
        if (issuers of signed rule and certificate doesn't match)
                return false;
        CertificateChain certchain=signedproofrule.getCertificateChain();
        if(certchain==null)
                return false;
        X509Certificate certs[]=certchain.getCertificates();
        if (certificate with credential can't be verified with public keys of certificate chain)
                return false;
        for(int i=0;i<certs.length;i++) {
                if (certs[i] is expired)
                        return false;
                if((i<certs.length-1)&&(!certs[i].verify(certs[i+1].getPublicKey()))
                        return false;
        }
        return true;
}
```

References

- Project pages
  - http://sourceforge.net/projects/peertrust/
  - http://www.learninglab.de/peertrust/
- Material used
  - http://www.learninglab.de/~olmedilla/pub/negotiationOnTheGrid.pdf
  - http://www.l3s.de/~olmedilla/pub/PeerTrust-NoRegistration.pdf
  - http://www.l3s.de/~olmedilla/pub/PeerTrust-ATN.pdf
  - http://java.sun.com/j2se/1.4.2/docs/api/
  - http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html
  - http://www.isoc.org/isoc/conferences/ndss/02/proceedings/papers/hess.pdf
  - http://dais.cs.uiuc.edu/pubs/winslett/itrust03.ps
  - http://www.research.att.com/~trevor/papers/JimPODS2001.pdf
  - http://www.cl.cam.ac.uk/users/mywyb2/publications/becker04cassandra-csfw2004.pdf
  - http://www4.ncsu.edu:8030/~tyu/pubs/tissec03.pdf
  - http://crypto.stanford.edu/~ninghui/papers/rt_oakland02.pdf
  - http://crypto.stanford.edu/~ninghui/papers/discovery_jcs03.pdf
  - http://www.ietf.org/rfc/rfc2459.txt
  - http://www.ietf.org/rfc/rfc2246.txt
  - http://www.rtfm.com/puretls/
  - http://www.research.att.com/~trevor/papers/JimOakland2001.pdf

The End

Thanks for your patience!

Questions?

May also be sent to seb0815@gmx.de