Pre-print version of Future Generation Computer Systems. 936 (2002) 1–13, December 2002. Elsevier Science BV

The PERMIS X.509 Role Based Privilege Management Infrastructure

David W Chadwick, Alexander Otenko
IS Institute, University of Salford, M5 4WT, England
Email: d.w.chadwick@salford.ac.uk o.otenko@pgt.salford.ac.uk
Telephone: +44 161 295 5351
Fax: +44 161 745 8169

Abstract

This paper describes the EC PERMIS project, which has developed a role based access control infrastructure that uses X.509 attribute certificates (ACs) to store the users' roles. All access control decisions are driven by an authorization policy, which is itself stored in an X.509 AC, thus guaranteeing its integrity. All the ACs can be stored in one or more LDAP directories, thus making them widely available. Authorization policies are written in XML according to a DTD that has been published at XML.org. The Access Control Decision Function (ADF) is written in Java and the Java API is simple to use, comprising of just 3 methods and a constructor. There is also a Privilege Allocator, which is a tool that constructs and signs ACs and stores them in an LDAP directory for subsequent use by the ADF.

Keywords

Trust Management, X.509, Attribute Certificates, Role Based Access Controls, XML, Privilege Management Infrastructure

The EC PERMIS Project

The EC funded PERMIS project was given the challenge of building an X.509 role based Privilege Management Infrastructure (PMI) that can be used by very different applications in 3 cities of Europe. The project has members from the cities of Barcelona (Spain), Bologna (Italy) and Salford (UK). All three centres already have experience of running pilot Public Key Infrastructures (PKIs), and so it was natural for them to want to add a PMI capability so as to complete the strong authentication and authorisation chain. The chosen applications of the 3 cities were very different in character, and so it would be a good test of the generality of the developed PMI if it could accommodate them.

In the case of Bologna, the city wants to allow architects to download street maps of the city, to update the maps with their proposed plans, and to upload the new building plans and requests for building licenses to the city planning office's server. This should significantly improve the efficiency of the current system, as the plans and requests are currently sent by post as paper documents to the city hall.

Barcelona is a major tourist and commercial centre and has many car hire locations throughout the city and at the airport. However, parking in Barcelona is very restricted, and many parking tickets are frequently issued to hired cars. By the time the car hire companies receive the parking tickets, the hirers have long since left the country. The plan is to give the car hire companies on line access to the city's parking ticket database, so that when cars are returned at the end of their hire period, the company can instantly check to see if any parking tickets have been issued for this

car. The company will be able to send the details of the driver to the city, thereby transferring the fine to the individual. Data protection legislation requires that a car hire company can only access the tickets issued to its own cars, and not to those of other car hire companies, and so authorisation will need to be at the record level.

Finally Salford is implementing an electronic tendering application. The tendering process will start when the city places the Request for Proposal documents on its web site, allowing anyone to download them. However, in some restricted tendering instances, only companies previously authorised by Salford will be able to submit tenders. In other cases it may be a requirement that a company has ISO 9000 or other certification in order to submit a tender. Once the tenders have been submitted, they must remain anonymous until the winner has been chosen. The city tender officers must not be given access to the electronic tender store before the closing date of the RFP, and tenderers must not be allowed to submit tenders after the closing date of the RFP.

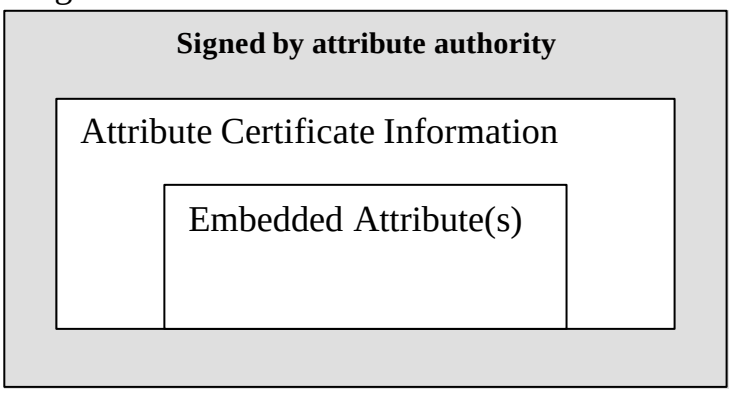
The challenge for the PERMIS project was to build a policy controlled role based X.509 PMI that can cater for these very different applications, and in so doing indicate that it will be useful to a much wider range of applications.

Introduction to X.509(2000) PMIs

Most people familiar with PKIs will know that the standard format for a public key certificate is specified in the X.509 standard [14], published jointly by the ITU-T and ISO/IEC. The primary purpose of a PKI is to strongly authenticate the parties communicating with each other, though the use of digital signatures. But strong authentication on its own is insufficient for a process to determine who is allowed to do what. An authorisation mechanism is needed for this. The recently released edition 4 of X.509 is the first edition of X.509 to fully standardise a strong authorisation mechanism, which it calls a Privilege Management Infrastructure (PMI). PMIs provide the authorisation function after the authentication has taken place, and have a number of similarities with PKIs. This paper assumes the reader is already familiar with the general concepts of PKIs, and these will not be repeated here. Readers wishing to learn more about PKIs may consult texts such as [1] or [12].

The primary data structure in a PMI is an X.509 Attribute Certificate (AC) (see Figure 1.) This strongly binds a set of attributes to its holder, and these attributes are used to describe the various privileges of the holder bestowed on it by the issuer. The issuer is termed an Attribute Authority (AA), since it is the authoritative provider of the attributes given to the holder. Examples of attributes and issuers might be: a degree awarded by a university, an ISO 9000 certificate issued by a QA compliance organisation, the role of supervisor issued by a manager, file access

Figure 1 Attribute Certificate



permissions issued by a file's owner. The whole data construct is digitally signed by the AA, thereby providing data integrity and authentication of the issuer.

The attributes are embedded within the Attribute Certificate Information data construct (see Figure 2). This contains details of the holder, the issuer, the algorithms used in creating the signature on the AC, the AC validity time and various optional extensions. Anyone familiar with the contents of an X.509 public key certificate (PKC) will immediately see the similarities between a PKC and an AC. In essence the public key of a PKC has been replaced by a set of attributes. (In this respect a public key certificate can be seen to be a specialisation of a more general attribute certificate.) Because the AC is digitally signed by the issuer, then any process in possession of an AC can check its integrity by checking the digital signature on the AC. Thus a PMI builds upon and complements existing PKIs.

Figure 2 Attribute Certificate Info

```
Attribute Certificate Info ::= SEQUENCE {
        version                 AttCertVersion,
        holder                  Holder,
        issuer                  Issuer,
        signature               AlgorithmIdentifier,
        attrCertValidityPeriod  AttCertValidityPeriod,
        attributes              SEQUENCE of Attribute,
        issuerUniqueID          UniqueIdentifier   OPTIONAL,
        extensions              Extensions OPTIONAL }
```

Since a PMI is to authorisation what a PKI is to authentication, there are many other similar concepts between PKIs and PMIs. Whilst public key certificates are used to maintain a strong binding between a user's name and his public key, an attribute certificate (AC) maintains a strong binding between a user's name and one or more privilege attributes. The entity that digitally signs a public key certificate is called a Certification Authority (CA), whilst the entity that signs an attribute certificate is called an Attribute Authority (AA). Within a PKI, each relying party must have one or more roots of trust. These are CAs who the relying party implicitly trusts to authenticate other entities. They are sometimes called root CAs[1] or trust anchors. Popular Web browsers come pre-configured with over 50 PKI roots of trust. The root of trust of a PMI is called the Source of Authority (SOA). This is an entity that a resource implicity trusts to allocate privileges and access rights to it. The SOA is ultimately responsible for issuing ACs to trusted holders, and these can be either end users or subordinate AAs. Just as CAs may have subordinate CAs to which they delegate the powers of authentication and certification, similarly, SOAs may have subordinate AAs to which they delegate their powers of authorisation. For example, in an organisation the Finance Director might be the SOA for allocating the privilege of spending company money. But (s)he might also delegate this privilege to departmental managers (subordinate AAs) who can then allocate specific spending privileges (ACs) to project leaders. When a problem occurs in a PKI, a user might need to have his signing key revoked, and so a CA will issue a certificate revocation list (CRL) containing the list of PKCs no longer to be trusted. Similarly if a PMI user

[1] Unfortunately X.509 did not standardise the term root CA or any term for the root of trust. Unfortunately disparate meanings for "root CA" have now evolved.

needs to have his authorisation permissions revoked, an AA will issue an attribute certificate revocation list (ACRL) containing the list of ACs no longer to be trusted. The similarities between PKIs and PMIs are summarised in Table 1.

Concept	PKI entity	PMI entity
Certificate	Public Key Certificate (PKC)	Attribute Certificate (AC)
Certificate issuer	Certification Authority (CA)	Attribute Authority (AA)
Certificate user	Subject	Holder
Certificate binding	Subject's Name to Public Key	Holder's Name to Privilege Attribute(s)
Revocation	Certificate Revocation List (CRL)	Attribute Certificate Revocation List (ACRL)
Root of trust	Root Certification Authority or Trust Anchor	Source of Authority (SOA)
Subordinate authority	Subordinate Certification Authority	Attribute Authority (AA)

Table 1. A Comparison of PKIs and PMIs

Implementing Authorisation Schemes with X.509

Various authorisation schemes have been devised in the past. The traditional, most popular and well known model is the Discretionary Access Control (DAC) scheme [24]. In the DAC scheme, users are optionally given access rights to resources by the resource administrator. In traditional systems the access rights are typically held as access control lists within each target resource (or application). In an X.509 PMI, the access rights have been removed from the resource, and are held within the privilege attributes of attribute certificates issued to users. Each privilege attribute within an AC will describe one or more of the user's access rights and the resources they are valid for. A target resource will then read a user's AC to see if he is allowed to perform the action that he is requesting.

Another authorisation scheme, popular with the military, is the Multilevel Secure (MLS) system, which is a type of Mandatory Access Control (MAC) scheme. In the MLS scheme, every target is given a security label, which includes a classification, and every subject is given a clearance, which includes a classification list. The classification list specifies which type of classified target the subject is allowed to access. A typical hierarchical classification scheme used by the military is: unmarked, unclassified, restricted, confidential, secret, and top secret. A typical security policy, designed to stop information leakage, is "read down and write up". This specifies that a subject can read targets with a lower classification than his clearance, and can write to targets with a higher classification. A user with clearance of confidential, who logs in as such, under this policy could read from unmarked to confidential targets, and write to confidential to top secret targets. The same user could also log in with a lower clearance level, say, unclassified and write to an unclassified target. X.509 supports MLS, by allowing subjects to be given a clearance AC. The privilege attribute in the AC now holds the users clearance. Targets can be securely configured with their own security label and the security policy that is to direct them. When a user contacts the target, his clearance can be extracted from the presented (or retrieved) AC and the target can act according to the policy.

More recently, research has focussed on Role Based Access Controls (RBAC) [23] [25]. In the basic RBAC model, a number of roles are defined. These roles typically represent organisational roles such as secretary, manager, employee etc. In the authorisation policy, each role is given a set of permissions i.e. the ability to perform certain actions on certain targets. Each user is then assigned to one or more roles. When accessing a target, a user presents his role(s), and the target reads the policy to see if this role is allowed to perform this action. RBAC has the advantage of scalability over DAC, and can easily handle large numbers of users as there are typically far fewer roles than users.

X.509 supports simple RBAC by defining role specification attribute certificates that hold the permissions granted to each role, and role assignment attribute certificates that assign various roles to the users. In the former case, the AC holder is the role, and the privilege attributes are permissions granted to the role. In the latter case the AC holder is the user, and the privilege attributes are the roles assigned to the user.

The hierarchical RBAC model is a more sophisticated version of the basic RBAC model. With this model, the roles are organised hierarchically, and the senior roles inherit the privileges of the more junior roles. So for example we might have the following hierarchy:

employee > programmer > manager > director.

If a privilege is given to an employee role e.g. can enter main building, then each of the superior roles can also enter the main building even though their role specification does not explicitly state this. If a programmer is given permission to enter the computer building, then managers and directors would also inherit this permission. Hierarchical roles mean that role specifications are more compact. X.509 supports hierarchical RBAC by allowing both roles and privileges to be inserted as attributes in a role specification attribute certificate, so that the latter role inherits the privileges of the encapsulated roles.

Another extension to basic RBAC is constrained RBAC. This allows various constraints to be applied to the role and permission assignments. One common constraint is that certain roles are declared to be mutually exclusive, meaning that the same person cannot simultaneously hold more than one role from the mutually exclusive set. For example, the roles of student and examiner, or the roles of tenderer (one who submits a tender) and tender officer (one who opens submitted tenders) would both be examples of mutually exclusive sets. Another constraint might be placed on the number of roles a person can hold, or the number of people who can hold a particular role. X.509 only has a limited number of ways of supporting constrained RBAC. Time constraints can be placed on the validity period of a role assignment attribute certificate. Constraints can be placed on the targets at which a permission can be used, and on the policies under which an attribute certificate can confer privileges. Constraints can also be placed on the delegation of roles. However many of the constraints, such as the mutual exclusivity of roles, have to be enforced by mechanisms outside the attribute certificate construct e.g. within the policy enforcement function.

Of course, every target that relies on X.509 ACs to confer privileges, also needs to be configured with a policy, or a set of rules, that will tell it which access methods are to be granted by which privileges. Unfortunately X.509 does not standardise any type of

policy and leaves this up to the applications using the X.509 PMI. This is one of the biggest challenges for anyone deciding to use an X.509 PMI.

The PERMIS PMI Architecture

The PERMIS PMI is, according to Blaze's definition in RFC 2704 [3], a trust management system. Consequently it must have the following five components:

i) A language for describing `actions', which are operations with security consequences that are to be controlled by the system.
ii) A mechanism for identifying `principals', which are entities that can be authorized to perform actions.
iii) A language for specifying application `policies', which govern the actions that principals are authorized to perform.
iv) A language for specifying `credentials', which allow principals to delegate authorization to other principals.
v) A `compliance checker', which provides a service to applications for determining how an action requested by principals should be handled, given a policy and a set of credentials.

X.509 attribute certificates specify mechanisms for ii) and iv). Principals are the holders and issuers of ACs and can be identified by their X.500 General Name (usually an X.500 distinguished names or IP address, URI or email address) or by reference to their public key certificate (issuer name and serial number) or if the principal is a software object by a hash of itself. Credentials are specified as X.500 attributes, which comprise an attribute type and value. Defining the policy (iii) and action (i) languages were significant tasks of the project, as was building a privilege allocation subsystem that creates X.509 ACs, and a compliance checker (v) (which we have termed a privilege verification subsystem) that validates them.

Defining the Policy Language

The authorisation policy needs to specify who is to be granted what type of action on which targets, and under what conditions. Domain wide policy authorisation is far more preferable than having separate access control lists configured into each target. The latter is hard to manage, duplicates the effort of the administrators (since the task has to be repeated for each target), and is less secure since it is very difficult to keep track of which access rights any particular user has across the whole domain. Policy based authorisation on the other hand allows the domain administrator (the SOA) to specify the authorisation policy for the whole domain, and all targets will then be controlled by the same set of rules.

Significant research has already taken place in defining authorisation policy languages. The Ponder language [7] is very compact and very powerful, but does not have a large set of supporting tools. The Keynote policy language [3] is also very comprehensive and covers many of our requirements, but is focussed on DAC rather than RBAC. Also, the policy assertions are very generic and are not related specifically to X.509. For example, the authoriser and licensees fields are opaque strings whereas we wanted them to have structure and meaning. Further, it does not seem to be possible to control the depth of delegation allowed from one authoriser to a subordinate. Finally, the syntax, comprising of ASCII strings and keywords, is specific to Keynote. The PERMIS project wanted to specify the authorisation policy in a well-known language that could be both easily parsed by computers, and read by the SOAs (with or without software tools). We decided that XML was a good

candidate for a policy specification language, since there are lots of tools around that support XML, it is fast becoming an industry standard, and raw XML can be read and understood by many technical people. Shortly after we started our work, Bertino et al published a paper [2] that showed that XML was indeed suitable for specifying authorisation policies. Later, the OASIS consortium began work on the eXtensible Access Control Markup Language [18], and this is now at an advanced draft state.

We needed a language tailored to X.509 and RBAC, so we specified a Data Type Definition (DTD) for our X.500 PMI RBAC Policy. The DTD is a meta-language that holds the rules for creating the XML policies. Our DTD comprises the following components:

- SubjectPolicy – this specifies the subject domains i.e. only users from a subject domain may be authorised to access resources covered by the policy
- RoleHierarchyPolicy – this specifies the different roles and their hierarchical relationships to each other
- SOAPolicy – this specifies which SOAs are trusted to allocate roles. By including more than one SOA in this policy, the local SOA is effectively cross certifying remote authorisation domains
- RoleAssignmentPolicy – this specifies which roles may be allocated to which subjects by which SOAs, whether delegation of roles may take place or not, and how long the roles may be assigned for
- TargetPolicy – this specifies the target domains covered by this policy
- ActionPolicy – this specifies the actions (or methods) supported by the targets, along with the parameters that should be passed along with each action e.g. action Open with parameter Filename
- TargetAccessPolicy – this specifies which roles have permission to perform which actions on which targets, and under which conditions. Conditions are specified using Boolean logic and might contain constraints such as “IF time is GT 9am AND time is LT 5pm OR IF Calling IP address is a subset of 125.67.x.x”. All actions that are not specified in a Target Access Policy are denied.

Table 2 shows a portion of the DTD that specifies the rules for the Role Assignment Policy, and below it is an example Role Assignment Policy for the Salford application.

```
<!ELEMENT RoleAssignmentPolicy (RoleAssignment)+ >
<!ELEMENT RoleAssignment  (SubjectDomain,Role,Delegate,SOA,Validity) >

<!ELEMENT SubjectDomain EMPTY>
<!ATTLIST SubjectDomain ID IDREF #REQUIRED>

<!ELEMENT Role EMPTY >
<!ATTLIST Role Type IDREF #IMPLIED
        Value IDREF #IMPLIED >

<!ELEMENT SOA EMPTY>
<!ATTLIST SOA ID IDREF #REQUIRED>

<!ELEMENT Validity (Absolute?, Maximum?, Minimum? ) >
<!ELEMENT Absolute EMPTY>
<!ATTLIST Absolute Start CDATA #IMPLIED
```

```
                End CDATA #IMPLIED >
<!ELEMENT Maximum EMPTY>
<!ATTLIST Maximum Time CDATA #IMPLIED >
<!ELEMENT Minimum EMPTY>
<!ATTLIST Minimum Time CDATA #IMPLIED >

<!ELEMENT Delegate EMPTY >
<!ATTLIST Delegate Depth CDATA #IMPLIED >
```

```
<RoleAssignmentPolicy>
   <RoleAssignment>
<!-- Role assignment for tender officers.
They must be employees of Salford City Council.  Valid only from close of tender.
Delegation not permitted -->
          <SubjectDomain ID="Employees"/>
     <Role Type="permisRole" Value="TenderOfficer"/>
     <Delegate Depth="0"/>
     <SOA ID="Salford"/>
     <Validity>
       <Absolute Start="2001-09-21T17:00:00"/>
     </Validity>
   </RoleAssignment>
   <RoleAssignment>
<!-- Role assignment for tenderers.
They must be dot com or co.uk companies.  Valid only until close of tender.
Delegation not permitted -->
          <SubjectDomain ID="Companies"/>
     <Role Type="permisRole" Value="Tenderer"/>
     <Delegate Depth="0"/>
     <SOA ID="Salford"/>
     <Validity>
       <Absolute End="2001-09-21T17:00:00"/>
     </Validity>
   </RoleAssignment>
   <RoleAssignment>
<!-- Role assignment for companies who are ISO9000 Certified.
They must be dot com or co.uk companies.  Valid only for a maximum of one year, as
companies have to be annually re-accredited. Certificates are issued by BSI.
Delegation not permitted -->
          <SubjectDomain ID="Companies"/>
     <Role Type="ISOCertified" Value="ISO9000"/>
     <Delegate Depth="0"/>
     <SOA ID="BSI"/>
     <Validity>
       <Maximum Time="+01"/>
     </Validity>
   </RoleAssignment>
</RoleAssignmentPolicy>
```

Table 2. The Role Assignment DTD and an Example Role Assignment Policy

The SOA creates the authorisation policy for the domain using his favourite XML editing tool, and stores this in a local file, say MyPolicy.XML, to be used by the Privilege Allocator to create the policy AC. A full description of the PERMIS X.500 PMI RBAC policy can be found in [4].

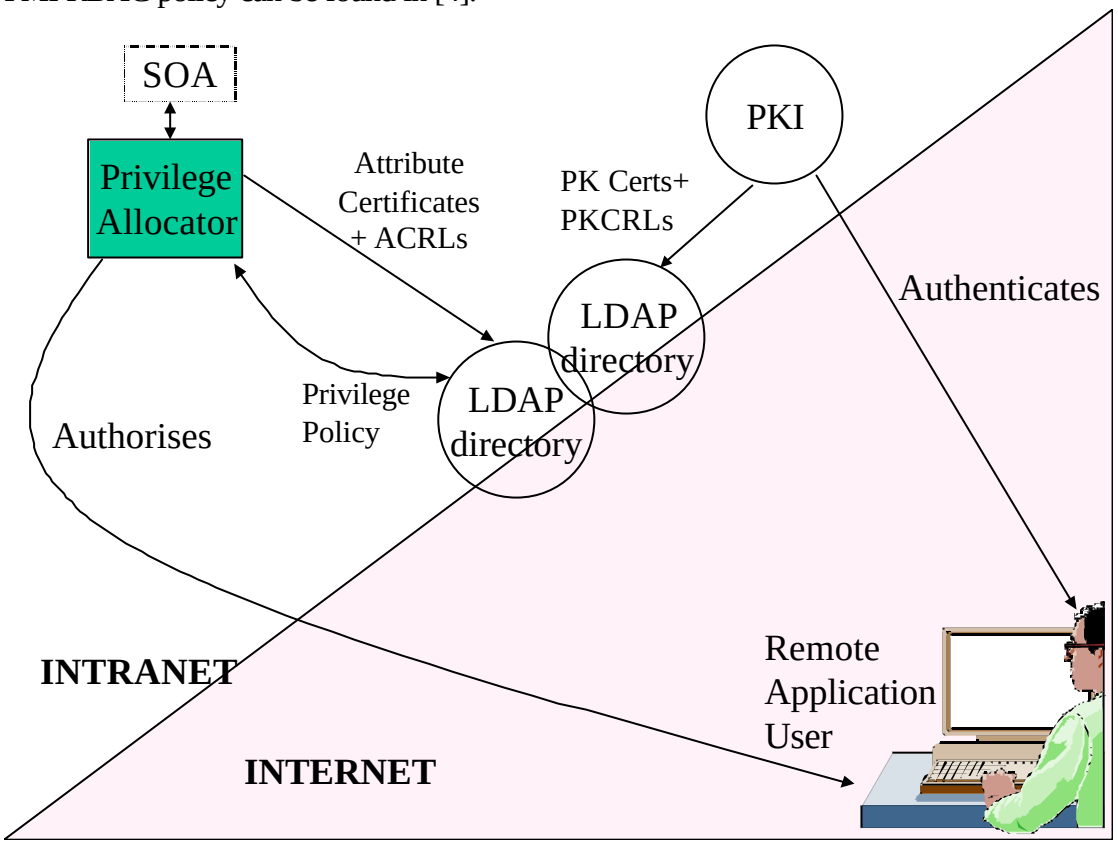


Figure 3. The Privilege Allocation Subsystem

The Privilege Allocation Subsystem

The privilege allocation subsystem (see Figure 3) comprises a Privilege Allocator (PA) (see later) that issues X.509 role assignment ACs to users, and also signs the policy AC that will control the RBAC API (see later). These are stored in an LDAP directory for subsequent use by the privilege verification subsystem. In addition to privilege allocation, each user will also need to be issued with an application specific authentication token. If a PKI is being used, this will be a digitally signed public key certificate, if a conventional authentication system is being used it will be a username/password pair. The PERMIS API is authentication agnostic.

The PERMIS API supports distributed authorisation, and so different sites can run the privilege allocator and allocate ACs to their users and store them in their local LDAP directories. This will significantly ease the management of privileges in large distributed environments, such as GRID networks, Internet marketplaces etc, as the local policy only needs to tell the PERMIS API which LDAP directories to contact and which remote SOAs to trust.

The PA

The PA is a tool used by the SOA or an AA to allocate privileges to users. Since PERMIS is using RBAC, the SOA uses the PA to allocate roles to users in the form of role assignment ACs. These ACs will be given to all the users of the various applications in the different cities. A role in PERMIS is simply defined as an attribute type and value. We are using two attribute types *permisRole* and *ISOCertified*, whose values are IA5 strings. In the case of Bologna, there are two permisRole values: Map-Readers and Architects. Map-Readers can download any maps produced by the municipality, whereas Architects are allowed to download maps and upload digitally modified maps. In the case of Barcelona, there are also two permisRoles defined: Generalised and Authorised. Any citizen or business can be allocated the Generalised role. Anyone with the Generalised role has permission to read their own pending car parking fines. Businesses that have signed an agreement with the Barcelona city council are given the Authorised role. Authorised roles can read their own pending fines and also may modify the details of them (e.g. update the driver's name and address). Salford is different to the other sites, in that whilst it will allocate two permisRoles, that of Tenderer and Tender-Officer, it will also rely on an external SOA (in this case the British Standards Institute) to allocate the ISOCertified roles to users. (In fact in the project we plan to set up a proxy BSI SOA to allocate these roles, as the BSI is not a project partner.)

Once the role assignment ACs have been created by the PA they are stored in an LDAP directory. Since ACs are digitally signed by the AA who issued them, they are tamper-resistant, and therefore there is no modification risk from allowing them to be stored in a publicly accessible LDAP directory. This also means that authorities who issue digital ACs can store them locally, but give global access to them. We think this will be particularly useful in the case of ISO 9000 certificates, for example. Anyone wishing to know if an organisation has ISO 9000 certification may access the BSI LDAP directory and retrieve the organisation's X.509 AC. The ACRLs of revoked certificates (if any) will also be stored here. Thus there is little advantage in general of distributing the ACs to their holders, since a relying party will still need to access the issuing authorities LDAP directory to retrieve the latest ACRL (or call an OCSP responder once these become available for attribute certificates). We see that this mechanism can be extended to any type of privilege certification e.g. Microsoft Certified Engineer, BSc (Hons) University of Salford etc. and that these "roles" can easily be built into our API via the authorisation policy. We are already using this with an electronic prescribing system that we are building, to allow the Royal College of Pharmacy to allocate pharmacist roles to qualified pharmacists, and the General Medical Council to allocate prescriber roles to qualified doctors.

Another function of the PA is to create a digitally-signed authorisation policy, as a policy AC. The policy AC is a standard X.509 AC with the following special characteristics: the holder and issuer names are the same (i.e. that of the SOA), the attribute type is pmiXMLPolicy and the attribute value is the XML policy created as described above. The policy AC indicates the root of trust of the PMI, and is similar to the self-signed public key certificate of the root CA of a PKI. The PA prompts the SOA for the name of the policy file (e.g. MyPolicy.XML) and then it copies the contents into the attribute value. After the SOA has signed the policy AC, the PA stores it in the SOA's entry in the LDAP directory. Each authorisation policy is given

an Object Identifier [13], which is a globally unique number. This ensures that the PERMIS API (see later) always runs with the correct policy for the domain.

The Privilege Verification Subsystem

The privilege verification subsystem (see Figure 4) is responsible for authenticating and authorising the remote user and providing access to the target. The application gateway provides the authentication and authorisation functionality. ISO 10181-3 Access Control Framework [15], splits the functionality of the application gateway into two components: an application-specific component termed the Access Control Enforcement Function (AEF), and an application-independent component termed the Access Control Decision Function (ADF). In this way, all access controls decisions in a domain can be consistently enforced by the ADF independent of the application. The ADF makes its decisions based on the authorisation policy for the domain, and on who is initiating a request, what action is being requested on which target, and environmental factors such as the time of day. An application programmable interface (API) between the AEF and ADF has already been defined by the Open Group. It is called the AZN API [27], and is specified in the C language. A similar API is also being developed by the IETF, called the Generic Authorization and Access control (GAA) API [21]. PERMIS has drawn on the completed work of the Open Group and made the following changes to the AZN API. Firstly we have specified the PERMIS API in Java rather than in C, and secondly we have significantly simplified the AZN API by assuming that the Target and the AEF are either co-located or can communicate with each other across a trusted LAN. Without this latter simplification the authorisation token carried from the AEF to the Target would need to be protected, for example as an X.509 attribute certificate, and so we would have gained very little from a remote ADF. We also assume that only a single authorisation service will be available, and that the API doesn't need to support the export of authorisation tokens, as ACs are already in an exportable format.

To summarise, in the PERMIS model, a user accesses resources via an application gateway. The AEF authenticates the user in an application specific way, then asks the ADF if the user is allowed to perform the requested action on the particular target resource. The ADF accesses one or more LDAP directories to retrieve the policy AC and the role ACs for the user, and bases its decision on these. If the decision is *grant*, the AEF will access the target on behalf of the user. If the decision is *deny*, the AEF will refuse access to the user. The AEF talks to the ADF via the PERMIS API.

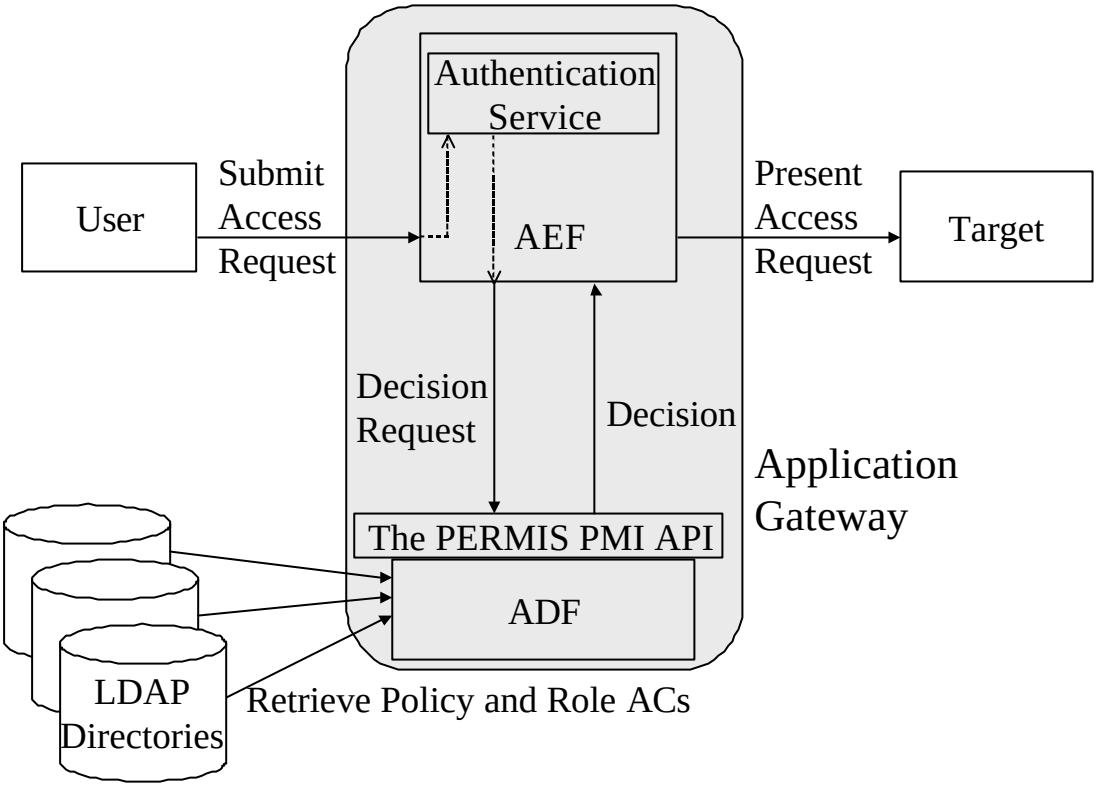


Figure 4. The Privilege Verification Subsystem

The PERMIS PMI API

The PERMIS API comprises 3 simple methods: GetCreds, Decision, and Shutdown, and a Constructor. The Constructor builds the PERMIS API Java object. For construction, the AEF passes the name of the SOA (the root of trust for authorisation), the Object Identifier of the policy, and a list of LDAP URIs from where the ADF can retrieve the policy AC and subsequently the role ACs. The policy AC is always retrieved from the first URI in the list. The Constructor is usually called immediately the AEF starts up. After construction of the API has completed, the ADF will have read in and validated the XML of the authorisation policy that will control all future decisions that it makes.

When a user initiates a call to the target, the AEF authenticates the user, then passes the LDAP DN [29] of the user to the ADF through a call to GetCreds. In the 3 cities the users will be authenticating in different ways. In Salford the user will be sending an S/MIME email message to the AEF, in Barcelona and Bologna he will be opening an SSL connection. In all cases the user will be digitally signing the opening message, and verification of the signature will yield the user's LDAP DN. The ADF uses this DN to retrieve all the role ACs of the user from the list of LDAP URIs passed at initialisation time (this is termed the "pull" model [10]). The role ACs are validated against the policy e.g. to check that the DN is within a valid subject domain, and to check that the ACs are within the validity time of the policy etc. Invalid role ACs are discarded, whilst the roles from the valid ACs are extracted and kept for the user, and returned to the AEF as a subject object. (The GetCreds interface also supports the "push" model [10] (called user-pull by Park et al [20]), whereby the AEF can push a

set of ACs to the ADF, instead of the ADF pulling them from the LDAP directories, but since our ADF currently does not retrieve CRLs, this mechanism is unused at present).

Once the user has been successfully authenticated he will attempt to perform certain actions on the target. At each attempt, the AEF passes the subject object, the target name, and the attempted action along with its parameters, to the ADF via a call to Decision. Decision checks if the action is allowed for the roles that the user has, taking into account all the conditions specified in the TargetAccessPolicy. If the action is allowed, Decision returns Granted, if it is not allowed it returns Denied. The user may attempt an arbitrary number of actions on different targets, and Decision is called for each one. In order to stop the user keeping the connection open for an infinite amount of time (for example until after his ACs have expired), the PERMIS API supports the concept of a session time out. On the call to GetCreds the AEF can say how long the session may stay open before the credentials should be refreshed. If the session times out, then Decision will throw an exception, telling the AEF to either close the user's connection or call GetCreds again.

Shutdown can be called by the AEF at any time. Its purpose is to terminate the ADF and cause the current authorisation policy to be discarded. This could happen when the application is gracefully shutdown, or if the SOA wants to dynamically impose a new authorisation policy on the domain. The AEF can follow the call to Shutdown with a new Constructor call, and this will cause the ADF to read in the latest authorisation policy and be ready to make access control decisions again.

Comparison with Related Work

A seminal model for trust management is that of Keynote [3] and this has formed the basis of the PEMIS work. However, whilst the Keynote model has been widely accepted as a good model for trust management, Keynote technology uses non-standard components, and therefore fails to attract a critical mass of users. Bacon and her colleagues at Cambridge University have been working on their OASIS [11] role based access control system for several years. OASIS conforms to Blaze's model but it has a number of disadvantages. Firstly, the work started before standardisation in this area and therefore it may be an uphill struggle for the OASIS infrastructure to be widely adopted. Secondly, their Role Membership Certificates (RMCs) are issued dynamically during user authentication and this may be an inefficient thing to do, as digital signing is processor intensive. The PERMIS PMI in contrast to OASIS uses standards based open technologies, so that components can be easily swapped or added to. Secondly the PERMIS X.509 ACs are created once only by the issuer, prior to user authentication, and can be used an arbitrary number of times, thereby being more efficient than OASIS's RMCs. Park et al [20] propose the use of "smart" certificates to support RBAC on the web. Smart certificates are standard X.509 PKCs, to which one or more AAs subsequently add attributes into extension fields of the base certificate and then sign the whole data structure putting their signature into another extension field. Unfortunately smart certificates are technically flawed, since the addition of extensions to a certificate after the CA has signed it, invalidates the CA's signature on the base certificate, so that it can no longer be used for authentication. We therefore won't consider this model any further.

Other alternative authorisation mechanisms to X.509 ACs are being experimented with today. GRID research in the USA is currently building the Globus Community Authorisation Server [6] that uses proxy X.509 PKCs rather than ACs. We believe this approach is fundamentally wrong for a number of reasons. Firstly the user is behaving as a certification authority, and is generating a key pair for the proxy application to use on his behalf. Attributes are then placed into the X.509 PKC of the proxy. This design requires the use of non-standard extensions to X.509 PKCs and is using what was designed as an authentication token to be a dual authentication and authorisation token, except that it is not the user who is authenticated, but the user's proxy. With the PERMIS design, the user authenticates as himself, and the X.509 ACs provide the authorisation tokens. So there is a clear separation of functionality, allowing different entities to perform the appropriate functions and different authentication mechanisms to be used. Secondly, the CAS server appears to be a potential bottleneck to performance, since no resource can be accessed unless the CAS service is there to provide a capability to the user. Thirdly, the policy is carried in the user's proxy certificate, therefore it has to be parsed and evaluated for each user by the resource. In the PERMIS system the policy is read in once at resource start up time, and is then ready to make decisions for any user who tries to access the resource. This should be much more efficient than the CAS design.

Akenti [16] is a possible alternative to the CAS design, but it uses proprietary ACs encoded in ASCII format. Akenti's infrastructure is similar to the PERMIS PMI, and uses XML to express policy, except that their policy is distributed and held in policy and use-condition certificates. They admit that this is a weakness in their design as a resource must be certain that it has collected all or none of the use-conditions necessary to control access. The PERMIS system has no such weakness as the authorisation policy is read in when the resource starts up, and any issuer policies are held in the ACs needed to grant access, so policy can never be lost. The most recent version of Akenti has attempted to overcome this weakness by inserting URLs into policy certificates that point to where use-condition certificates may be found [28], but this collection and validation process will still be far more inefficient than the single PERMIS policy AC, and their ACs are still in a non standard format.

The PAPI authorisation system developed in Spain [19] allows a home site to control the remote sites that a user is authorised to access. The home site sends a list of allowed URLs to the user's web browser. Along with these URLs is the authenticated name of the user signed by the home site's authentication server. This information is passed to a remote site's Point of Access (PoA) authorisation server who now knows the authenticated name of the user since it trust the public key of the home site's authentication server. The PoA server generates a set of cookies that are passed back to the client's web browser. The user can now visit any site in the URL list and the cookies are used to grant access to the Web servers. However PAPI is really only a protocol for passing cryptographic cookies around, it is not a policy based trust management infrastructure. No policy is defined, no access rules are defined, and no decision function is defined. Each site has to implement its own specific access control mechanism and put its private credentials into the PAPI cookies. In fact the PERMIS infrastructure could sit beneath the PoA server and be used to grant or deny access to the web servers.

The OASIS consortium SAML assertions [22], are another way of passing

authentication and authorisation assertions between remote sites. The Internet2 Shibboleth project [9] has recently decided to use SAML assertions as their way of passing attribute assertions between web sites and browsers. SAML assertions are thus an alternative to X.509 PKCs and ACs, Akenti certificates and PAPI cookies. One reason why we are using X.509 ACs and not SAML assertions is that we have found in recent research conducted under another project, that ASN.1 BER outperforms XML by approximately an order of magnitude [17]. Not only are X.509 ACs more compact, thereby taking up less storage space and taking less time to transmit (which is important in mobile phones for example), but the BER signature generation and validation processes are also significantly faster than the equivalent XML signatures recently standardised by the IETF [8]. We believe that performance is one factor that will affect user acceptance of any globally distributed authorisation system, and that the structure of the data messages is irrelevant to the users (although not admittedly to software developers). If the push model using SAML assertions is a requirement for an application, then X.509ACs can be embedded and transferred as SAML assertions and the PERMIS PMI can be used within the web server. But one of the strengths of the PERMIS PMI is that it supports the pull model for the retrieval of ACs, and with this model SAML assertions are not needed and client applications do not need to be modified.

Finally, recent research by Seamons et al [26] has shown the symmetrical nature of trust management in Internet based transactions, in that not only do electronic services have to trust the users before they will provide a service to them, but also the users have to trust the services before they will release their confidential or sensitive ACs to them. This is because some ACs may contain genuinely confidential information e.g. a user's clearance level, whilst in other cases the user may be concerned about his privacy and not wish to divulge that he possesses particular ACs until he knows he is dealing with a reputable Internet service. This situation does not usually arise in the physical world today, since the user is usually present in the physical premises of the service provider and therefore he senses how to trust it. However in the electronic world trust may need to be incrementally ramped up as each party learns more about the other and is willing to share more information with the other. For example, a consumer may not be willing to send his electronic credit card to an Internet holiday shop until he first sees the ABTA bonded AC issued to the shop. We anticipate that this symmetry could be implemented using the PERMIS PMI by installing an ADF on the user's PC, that controls access to his set of ACs, and only allows the remote server to retrieve the ACs after fulfilling certain criteria as directed by the user's authorisation policy. We hope to build such a symmetrical infrastructure using the PERMIS PMI in future research.

Conclusion

We have shown how the standard X.509 PMI can be adapted to build an efficient role based trust management system, in which the role assignments can be widely distributed between organisations, and the local authorisation policy determines which roles are to be trusted and what privileges are to be given to them. The local authorisation policy is written in XML and version 7 of the PERMIS X.500 PMI RBAC Policy DTD has been published at http://www.xml.org, and is also available from our web site http://sec.isi.salford.ac.uk/permis/Policy.dtd. The local authorisation policy governs all aspects of access to the targets in the local domain. A simple Java API is provided which allows all target applications to easily incorporate

this system. Public releases of the PERMIS API are available for non-commercial use from our web site.

The generality of the PERMIS API has already proven its worth. In another research project at Salford we have designed an electronic prescription processing system where doctors, pharmacists, nurses and patients are all allocated appropriate roles. With a suitable policy the ADF is able to make decisions about whether a doctor is allowed to issue a prescription or not, whether a pharmacist is allowed to dispense a prescription or not, and whether a patient is entitled to free prescriptions or not. It is expected that many more applications will use the PERMIS API in due course.

An earlier version of this paper [5] was presented at the SACMAT 2002 conference.

Acknowledgments

This work has been 50% funded by the EC ISIS programme PERMIS project, and partially funded by the EPSRC under grant number GR/M83483. The authors would also like to thank Entrust Inc. for making their PKI security software available to the University on preferential terms.

References

[1] Adams, C., Lloyd, S. (1999). "Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations". Macmillan Technical Publishing, 1999

[2] Bertino, E., Castano, S., Farrari, E. "On specifying security policies for web documents with an XML-based language". Proceedings of the Sixth ACM Symposium on Access control models and technologies 2001, available from ACM digital library.

[3] Blaze, M., Feigenbaum, J., Ioannidis, J. "The KeyNote Trust-Management System Version 2", RFC 2704, September 1999.

[4] D.W.Chadwick, A. Otenko. "RBAC Policies in XML for X.509 Based Privilege Management" in Security in the Information Society: Visions and Perspectives: IFIP TC11 17th Int. Conf. On Information Security (SEC2002), May 7-9, 2002, Cairo, Egypt. Ed. by M. A. Ghonaimy, M. T. El-Hadidi, H.K.Aslan, Kluwer Academic Publishers, pp 39-53.

[5] D.W.Chadwick, A. Otenko. "The PERMIS X.509 Role Based Privilege Management Infrastructure", Proc 7th ACM Symposium On Access Control Models And Technologies (SACMAT 2002), Monterey, USA, June 2002. pp135-140.

[6] CAS. See http://www.globus.org/security/CAS/

[7] Damianou, N., Dulay, N., Lupu, E., Sloman, M. "The Ponder Policy Specification Language", Proc Policy 2001, Workshop on Policies for Distributed Systems and Networks, Bristol, UK 29-31 Jan 2001, Springer-Verlag LNCS 1995, pp 18-39

[8] Eastlake, D., Reagle, J., Solo, D. "(Extensible Markup Language) XML-Signature Syntax and Processing" RFC 3275, March 2002

[9] Erdos, M. and Cantor, S. "Shibboleth-Architecture DRAFT v05" see http://middleware.internet2.edu/shibboleth/

[10] Farrell, S., Housley, R. "An Internet Attribute Certificate for Authorization", <draft-ietf-pkix-ac509prof-05.txt>, August 2000

[11] Hayton R., Bacon J., Moody K., "OASIS: Access Control in an Open, Distributed Environment". Proc IEEE Symposium on Security and Privacy, Oakland CA, pp3-14, May 1998.

Pre-print version of Future Generation Computer Systems. 936 (2002) 1–13, December 2002. Elsevier Science BV

[12] Housley, R., Polk, T. "Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure". John Wiley and Son, ISBN: 0-471-39702-4, 2001
[13] ITU-T Recommendation X.680 (1997) | ISO/IEC 8824-1:1998, Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation
[14] ITU-T Rec. X.509 (2000) | ISO/IEC 9594-8 The Directory: Authentication Framework
[15] ITU-T Rec X.812 (1995) | ISO/IEC 10181-3:1996 "Security Frameworks for open systems: Access control framework"
[16] Johnston, W., Mudumbai, S., Thompson, M. "Authorization and Attribute Certificates for Widely Distributed Access Control," IEEE 7th Int Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE), Stanford, CA. June, 1998. Page(s): 340 -345 (see also http://www-itg.lbl.gov/security/Akenti/)
[17] Mundy, D., Chadwick, D.W. "Performance Comparisons of Abstract Syntax Notation One (ASN.1) vs. eXtensible Markup Language (XML)" submitted to NDSS '02 for review. Available for personal study from the authors.
[18] "OASIS eXtensible Access Control Markup Language (XACML)" v0.11, March 2002, available from http://www.oasis-open.org/committees/xacml/docs/
[19] PAPI. See http://www.rediris.es/app/papi/index.en.html
[20] Park, J.S., Sandhu, R., Ahn,G. "Role-Based Access Control on the Web", ACM Transactions on Information and Systems Security, Vol 4. No1, Feb 2001, pp 37-71.
[21] Ryutov, T., Neuman, C., Pearlman, L. "Generic Authorization and Access control Application Program Interface C-bindings" <draft-ietf-cat-gaa-cbind-05.txt>, November 2000. See http://www.isi.edu/gost/info/gaaapi/
[22] SAML. See http://www.oasis-open.org/committees/security/
[23] Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E. "Role Based Access Control Models". IEEE Computer 29, 2 (Feb 1996), p38-43.
[24] Sandhu, R. and Samarati, P. "Access controls, principles and practice". IEEE Communications, 32(9), pp 40-48, 1994
[25] Sandhu, R., Ferraiolo D., Kuhn, R. "The NIST Model for Role Based Access Control: Towards a Unified Standard". In *proceedings of 5th ACM Workshop on Role-Based Access Control*, pages 47-63. (Berlin, Germany, July 2000).
[26] Seamons, K., Winslett, M., Yu, T., Smith, B., Child, E., Jacobsen, J., Mills, H., Yu, L. "Requirements for Policy Languages for Trust Negotiation", Proc. 3rd Int.Conf.on Policies for Distributed Systems and Networks, (Policy 2002)
[27] The Open Group. "Authorization (AZN) API", January 2000, ISBN 1-85912-266-3
[28] Thompson, M. R., Mudumbai, S., Essiari, A., Chin, W. "Authorization Policy in a PKI Environment", Proceedings of the First Annual PKI Workshop, Dartmouth College, April 2002, pages 137-149. see www.cs.dartmouth.edu/~pki02
[29] Wahl, M., Kille, S., Howes, T. "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC2253, December 1997.

Author Biographies

David W Chadwick is a professor of information systems security and the leader of the Information Systems Security Research Group (ISSRG) at the University of Salford. He has published widely on the topics of X.500, PKIs and X.509. He is the BSI representative to X.509 standardisation meetings, and was the international editor of X.518(93). He regularly attends IETF meetings and is currently the author of 4 Internet Drafts concerning the use of PKIs and LDAP. The ISSRG is part of the larger

Information Systems Research Group which received the top 5* research rating in the UK 2000 Research Assessment Exercise. Professor Chadwick has participated in many EC and UK security related research projects including: ICE-TEL, TrustHealth 2, ICE-CAR, Secure Exams, GUIDeS, Intelligent Computation of Trust, the Distributed Diabetic Dietician, PERMIS, the PKI Challenge, Certificate retrieval from OpenLDAP, Electronic Prescriptions Processing and Secure Discharge Notes.

Alexander Otenko is a final year PhD student at the University of Salford. He has a specialist degree in Applied Mathematics and Computing from Sumy State University in the Ukraine, where he was the top student in his year. He has been working on the Permis project as the subject of his PhD, and is the primary author of much of the Java code in Permis. Prior to that he had over 7 years of programming experience in different languages, including long term voluntary cooperation with the Regional Department of SaveBank of Ukraine. His area of specialization was writing computer virus detection and elimination software based on syntax analysis of polymorphic stammes. He won many certificates and awards in the Ukraine including Brainbench E-certification on C and C++ programming (17 Jan 2000), Komitex antiviral contest (1997, 4 levels), Simple COM viruses (Ninnishbased VD Slam 379) (3-d level), Stealth boot virus (See.you.b 512) (1-st level), Full morph COM-EXE virus (PLY 3360) (1-st level), Polymorph Boot-COM-EXE virus (Win4.0 1536) (1-st level), Diploma of the science conference participant (1995) and five Republican Olympiads.