The X.509 attribute Parsing Server (XPS)

d.w.chadwick@salford.ac.uk

The Problem

- PKI clients cannot search for specific X.509 attributes stored in LDAP directories, e.g.
  - Find the encryption PKC for the person whose email address is fred.bloggs@myorg.com
  - Find the CRLs issued by OU=MyCA, O=MyOrg, C=US after 9am, 20March 2003
  - Find the AC for David Chadwick that contains the role attribute
- PKI clients currently can only store and retrieve X.509 attributes, by knowing the Distinguished Name of the entry they are held in – but often the clients do NOT know the DN of the entry

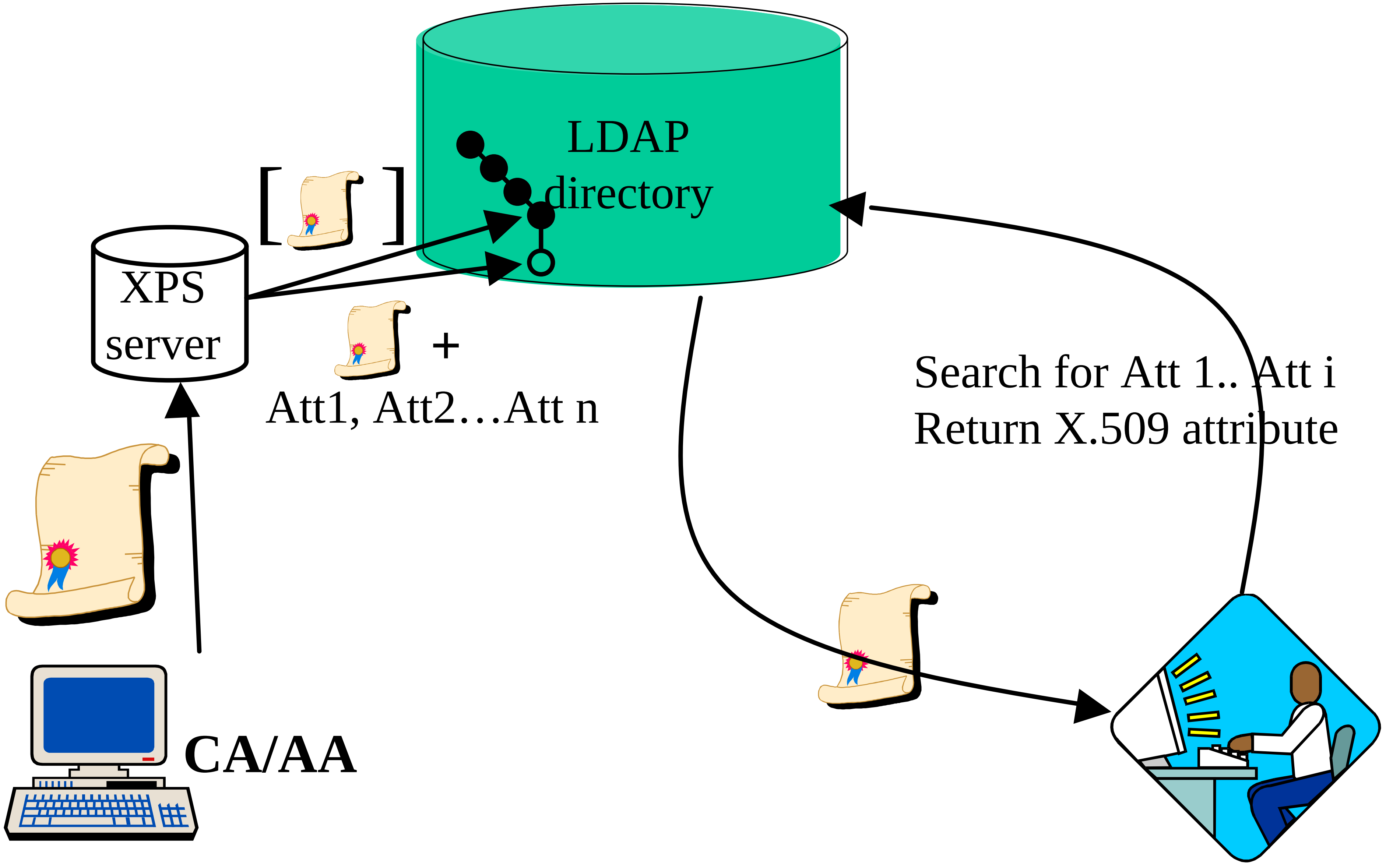
The Current Workaround

- PKI vendors such as Entrust, suggest that the PKI/LDAP administrator extracts the email address of the PKC subject from the SubjectAltName field, and store this in an email attribute in the same entry as the PKC, and then the PKI client searches the LDAP server for the email address and asks for the PKC to be returned from the same entry

Automating and Extending the Workaround - Attribute Extraction

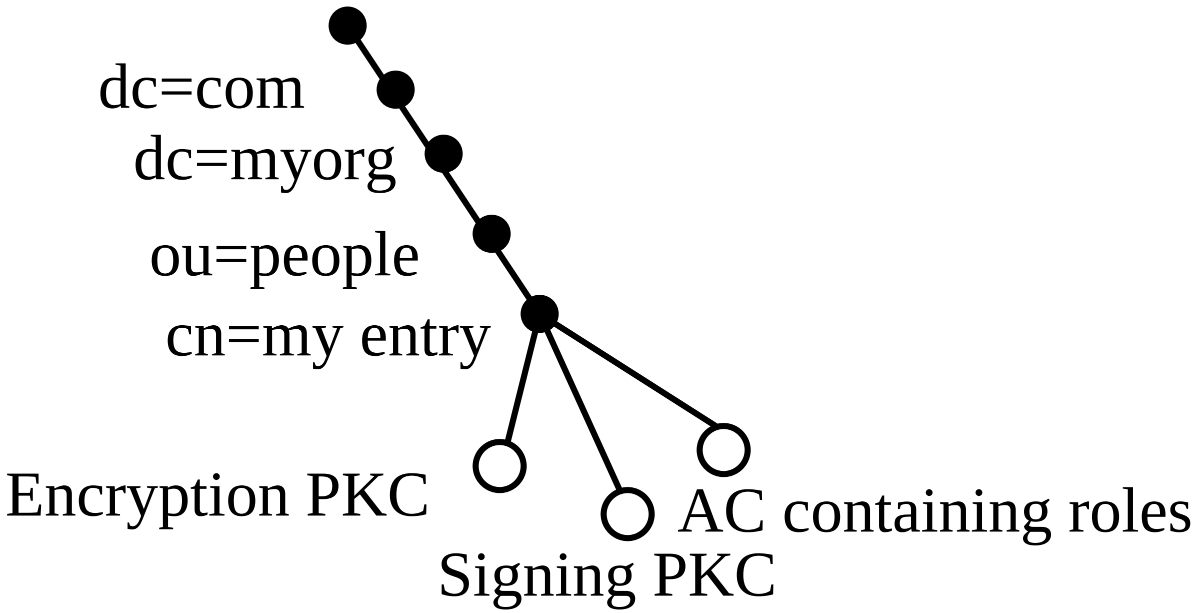
- A front end X.509 attribute Parsing Server (XPS) parses the X.509 attribute to be stored, and breaks it up into a set of LDAP attributes that use existing LDAP syntaxes
- XPS then creates a new entry for the X.509 attribute in the LDAP server, which comprises the original X.509 attribute and the set of extracted attributes
- The LDAP server adds the extracted attributes to its indexes, using existing mechanisms
- The PKI administrative client (i.e. the CA) talks to the new XPS server instead of the LDAP server
- PKI retrieval clients search the LDAP server for the extracted attributes and ask for the X.509 attribute to be returned (as now)

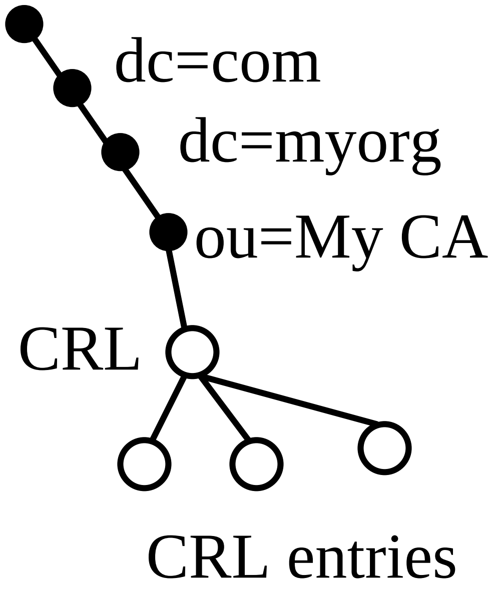
Attribute Extraction



The DIT Structure

- PKCs and ACs are held in child entries
- CRLs are held in child subtrees





Naming the X.509 attribute entries

- CRL entries are named with the x509crlThisUpdate attribute
- CRL revoked certificate entries may be named in one of 3 ways
  - x509serialNumber+x509issuer
  - x509isssuerSerial
  - x509serialNumber
- Certificate Entries may be named in one of 3 ways
  - x509serialNumber+x509issuer
  - x509isssuerSerial
  - x509serialNumber

The XPS Configuration File

- XPS config options are held in slapd.conf between the global configuration directives and the backend definitions
  - All config keywords are case sensitive. Default values in red
- EnableXPS [yes|no]
  - If no, all other XPS config options are ignored
- ldapurl [NULL|<URL of LDAPv3 server>]
  - XPS can act in standalone mode or combined mode
- pkcTypes [userCertificate, userCertificate;binary, cACertificate, cACertificate;binary]
  - Lists the incoming public key attribute types to be trapped
- acTypes [attributeCertificateAttribute, attributeCertificateAttribute;binary]
  - Lists the incoming attribute certificate types to be trapped
- crlTypes [certificateRevocationList;binary, etc.]
  - Lists the incoming CRL attribute types to be trapped

XPS Configuration Options (cont)

- DuplicateAttribute [yes|no]
  - Indicates if X.509 attribute should be stored in parent entry as well
- RevokedCertificateEntries [yes|no]
  - Indicates if subtree of entries should be created, default no
- RevokedRDNformat [x509serialNumber+x509issuer| x509isssuerSerial | x509serialNumber]
  - Only applies if RevokedCertificateEntries is yes
- CertRDNformat [x509serialNumber+x509issuer | x509isssuerSerial | x509serialNumber]
  - The RDN of the newly created certificate entries
- Walpath [path]
  - Where to store the WAL files. The default path if this parameter is missing, is /usr/local/var/xps/wals/

XPS Config options (cont)

- XPSerrorlog [path/filename]
  - The default file is /usr/local/var/xps/error.log
- x509attrtypes [path/filename]
  - The location of the attribute types mapping file. The default file is /usr/local/var/xps/x509attrtypes.txt

The Attribute Types Mapping File

- Holds the mapping between each ASN.1 component type reference from an X.509 attribute type definition, and the LDAP attribute type that is to hold the component value.
- Component reference followed by **attribute type**
  - X509Certificate.tbsCertificate.version **x509version**
  - X509Certificate.tbsCertificate.validity.notBefore **x509validityNotBefore**
- Only the LDAP attributes specified in this file will be stored in the X.509 attribute entries created by XPS

Parsing the Incoming X.509 Attributes

- When any of the attributes in the 3 lists from the config file are encountered, one of the following routines is called
  - x509AC_2_mods() - converts an Attribute Certificate into a list of Modifications
  - x509CRL_2_mods() - converts a Certificate Revocation List into a list of Modifications
  - x509PKC_2_mods() - converts a Public Key Certificate into a list of Modifications
- These routines are automatically created by an ASN.1 compiler we have written

The ASN.1 Compiler

- Built using the compiler generator tools *flex* and *bison*.
- Flex is based on UNIX lex and builds a lexical analyzer which is used to process the text input to the complier
- ASN.1 built-in type keywords used by flex are contained in input file asn1.lex
- Output from the lexical analyzer is a sequence of tokens and corresponding values that are used by the compiler
- The compiler is built using the tool *bison*, a compiler generator based on UNIX tool yacc
- The definitions for the compiler are contained in input file asn1.y
- Bison does not automatically generate parse trees and output code. These must be added to the definition file, in the form of embedded actions, to build a parse tree, and also a code generator. After the input has been scanned and a parse tree built, the code generator walks this tree and translates this into output code.
- Compiler is called with two arguments: name of file containing X.509 attribute ASN.1 type definition, and file to receive generated C code

The WAL

- XPS is acting as a transaction server, since one incoming request creates a set of outgoing requests to the backend LDAP server
- Therefore we need a Write Ahead Log
- One WAL file is opened per incoming operation
- Before any change is made to the backend LDAP server, this change is written to the WAL in LDIF format
  - Add entry, only the DN is stored in the WAL
  - Remove entry, the entire entry is read and the contents stored in the WAL
  - fflush() is called after every write to the WAL
- When the last backend operation has completed successfully the WAL is deleted

WAL File names

- Must be unique
- Filename is wal<32Ascii chars>.log
  - E.g. walKHWIHfdsafJG420ghdlT4YG.log
- Ascii chars are created using a 128 bit MD5 hash of the DN of the LDAP entry to be modified
  - lutil_MD5Init(), lutil_MD5Update(), and lutil_MD5Final() functions
- This provides concurrency control as well

Recovery

- When XPS first starts, if a WAL file is found, then a recovery log is opened (XPSrecovery.log)
- The WAL is opened, marked “recovery in progress” and each record is acted upon
  - If it’s a DN, the entry is deleted
  - If it’s an entry, the entry is added
- If successful the record is deleted. If it fails the record is left there
- Each action is written to the log file, along with a success or fail status
- If recovery fails, the LDAP administrator will have to tidy up manually using the log file and WALs

Operation of XPS

- If EnableXPS is No, all operations bypass it and given to OpenLDAP
- If EnableXPS is Yes, and ldapurl is null, then Bind, Unbind, Compare, ModDN, Abandon, and Search bypass it
- If Enable XPS is Yes, and ldapurl is not null, then Bind, Unbind bypass it whilst a referral to this url is returned to Compare, ModDN, Abandon and Search
- If Enable XPS is Yes, ADD, DELETE, and MODIFY Requests are trapped and acted upon

Add Request

- First check if request contains any X.509 attributes, if not either bypass or return referral
- Call appropriate x509***_2_mods() function for each trapped X.509 attribute
- Mods are turned into slap_mods2entry() or into an LDAP ADD request depending if LDAP server is internal or external
- DN of parent and child entries are added to WAL
- If internal, (be->be_add)() called for parent entry
- If external, ldap_add_ext_s() called for parent entry
- If successful, above is repeated for each child entry
- WAL is then deleted.

Delete Request

- Full subtree search from DN of entry-to-be-deleted is issued, with filter of object class present, requesting *
- If no entries returned, Delete either passed to internal server or referral returned
- If all entries except base are of object class x509base, Delete will be processed, else it will be rejected with "unwillingToPerform"
- If a returned entry has the *hasSubordinates* operational attribute = FALSE, the entry will be saved in the WAL using Entry2wal() and then deleted from the directory using (be->be_delete)() or ldap_delete_ext_s()
- If *hasSubordinates* is TRUE, the entry is held pending until all the FALSE ones are processed, then it is processed
- Once all entries have been deleted, the WAL is deleted

Modify Request (Add)

- XPS first checks if there are any X.509 attributes in the modifications. If there are none then request is passed to internal server or referral is returned
- Processing now depends upon type of modification
- For Add, a list of modifications for each X.509 attribute is created using appropriate x509***_2_mods()
- The DNs of all the children and grandchildren are added to the WAL
- The entries and child entries are then added to the directory. Move to next modification

Modify Request (Delete)

- If the modification is delete, do a 1-level Search from the to-be-modified entry for the attribute type (or value) to-be-deleted.
- Store returned entries in the WAL
- If CRL revoked entries are present, do a 1-level Search from each returned CRL entry and store these in the WAL. Then delete them from the directory.
- Delete initial returned entries from the directory
- If any errors, then add all the deleted entries back into the directory
- Move to next modification

Modify Request (finish)

- Replace is equivalent to delete the attribute and add the listed values
- Once the set of modifications have been completed successfully, the original Modify request has its X.509 attributes removed (if duplicates is false) and it is sent to the directory (unless there are no attributes left)
- As the Modify is atomic, it does not need to be written to the WAL. If it succeeds, delete the WAL, if it fails, then re-apply the changes recorded in the WAL

Governing Internet Drafts

- Gietz, P., Klasen, N. "An LDAPv3 Schema for X.509 Certificates",<draft-klasen-ldap-x509certificate-schema-02.txt>, March, 2003
- Chadwick, D.W., Sahalayev, M. V. "Internet X.509 Public Key Infrastructure - LDAP Schema for X.509 Attribute Certificates", <draft-ietf-sahalayev-pkix-ldap-ac-schema-00.txt>, February 2003
- Chadwick, D.W., Sahalayev, M. V. "Internet X.509 Public Key Infrastructure - LDAP Schema for X.509 CRLs", <draft-ietf-pkix-ldap-crl-schema-00.txt>, February 2003

Attribute Extraction

Pros

- Existing LDAP servers do not need to be modified
- Supports indexing servers based on fields within PKI attributes
- Supports enhanced matching (searching on multiple fields within a single PKI attribute)
- XPS has been built into OpenLDAP, and can either front end an existing LDAP server or be a combined XPS/LDAP server
- PKI clients need less modification, and might just need re-configuring with the new attribute types
- The European and US academic community chose this solution

Cons

- Storage requirements in the LDAP server are doubled [or tripled]
- Adding new X.509 attribute syntaxes or new certificate or CRL extensions means the XPS code has to be recompiled and rebuilt
- Cant search on multiple X.509 attributes, or an X.509 attribute and other attributes in the user's entry (needs families of entries)
- CAs and clients have different views of the DIT