Service Location Working Group
INTERNET DRAFT

Jonathan Wood
Roberto Tam
Sun Microsystems, Inc.
25 June 1999

The Naming and Directory Service Abstract Type
draft-ieft-svrloc-naming-directory-scheme-01.txt

Status of This Memo

This document is a submission by the Service Location Working Group of the Internet Engineering Task Force (IETF). Comments should be submitted to the srvloc@srvloc.org mailing list.

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as ``work in progress.''

To view the entire list of current Internet-Drafts, please check the ``1id-abstracts.txt'' listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), ftp.nordu.net (Northern Europe), ftp.nis.garr.it (Southern Europe), munnari.oz.au (Pacific Rim), ftp.ietf.org (US East Coast), or ftp.isi.edu (US West Coast).

Distribution of this memo is unlimited.

Abstract

This document describes the Naming and Directory Service abstract type. This abstract type serves as an umbrella type for all services which support directory-style operations for obtaining system information. This type collects a core set of attributes common to all such services.

1. Introduction

This document defines a template for the naming and directory service abstract type. Templates and service types are defined in [1]. This type can be used with SLP [2] to dynamically discover naming and directory servers. In particular, this type applies to name spaces which contain

information intended primarily for system consumption; examples of such information are UNIX-style system information (such as passwd, hosts, and service tables), and public-key certificates for authenticating principals such as people and hosts.

Currently there is a wide variety of services which are used to serve system information (i.e. NIS, NIS+, LDAP, NDS, etc.). It is common for two or more of these services to be deployed at the same time on the same network. This creates configuration complexity for naming and directory clients on the network: How does a client choose the right service, and once chosen, how does the client find the access handle to that service?

The type defined by this document manages this complexity by collecting attributes common to all naming and directory services, allowing clients to select the right service from a heterogeneous pool. This type can be used in conjunction with SLP to implement a unified discovery solution.

Note that concrete types include all attributes defined in this template; they may also define attributes specific to their service. Examples of concrete type templates can be found in [3], [4], and [5].

2. Examples

This section presents two scenarios to illustrate how this type might be used. For both scenarios, it is assumed that a number of naming and directory services have been deployed in the network, and that they are advertising their services via SLP. The following list enumerates the registered servers and their attributes. Note that in an actual registration, the template attributes would be included in the attributes list, and all attributes and URLs would be properly escaped. Here, however, these steps have been omitted for the sake of brevity and readability:

```
URL:
  service:naming-directory:nis://192.168.1.100/eng.wiz.com
Attributes:
  naming-context=eng.wiz.com
  organization=flat
  dynamic-updates=false
  jndi-sp-available=true
  master=true

URL:
  service:naming-directory:nis://192.168.1.200/eng.wiz.com
```

```
   Attributes:
     naming-context=eng.wiz.com
     organization=flat
     dynamic-updates=false
     jndi-sp-available=true
     master=false

   URL:
     service:naming-directory:nisplus://192.168.2.100/b17.eng.wiz.com
   Attributes:
     naming-context=b17.eng.wiz.com
     organization=hierarchical
     dynamic-updates=true
     jndi-sp-available=false
     master=true
     security-mechanism=dh-ext
     pubkey=(dh-ext)1a22345def3324f3ecbb...

   URL:
     service:naming-directory:ldap://192.168.3.100/dc=eng,dc=wiz,dc=com
   Attributes:
     naming-context=dc=eng,dc=wiz,dc=com
     organization=hierarchical
     dynamic-updates=true
     jndi-sp-available=true
     master=true
     security-mechanism=clear,tls,kerb5
     transport=cots
```

Note that some attributes are not defined in the template given below; they are defined in the templates for the particular services.

2.1. System Configuration

A number of UNIX platforms currently bundle clients for the NIS, NIS+, and LDAP services. When the system initially needs to configure its name service, it has no idea with what services its new network has been populated. It does, however, know that it wants an authenticated connection to its server, and that it has been configured with Kerberos and extended DH security. Also, it would like to talk to an authoritative server to populate its initial configuration.

So, in order to discover what naming services are available, the system issues the following SLP service request:

```
<type>=service:naming-directory
```

<predicate>=(&(|(security=kerb5)(security=dh-ext))(master=true))

It receives the following service handles:

service:naming-directory:ldap://192.168.3.100/dc=eng,dc=wiz,dc=com
service:naming-directory:nisplus://192.168.2.100/b17.eng.wiz.com

Each service handle provides enough information to contact and query the server.

The system then decides whether to use NIS+ or LDAP (or both), and uses the access handle to contact the server.

2.2. JNDI Configuration

The Java Naming and Directory Interface (JNDI) [6] provides a common interface to naming and directory operations. Using JNDI, it is possible to write an application which accesses directories without knowing which particular naming or directory service it is actually talking to.

However, JNDI applications require some initial configuration in order to find and query a service; this configuration is different for each different kind of service.

This example demonstrates how JNDI applications can use SLP to configure themselves in a unified manner. All the application needs to know in advance is in what manner it wishes to use a naming or directory service.

In this example, the JNDI application wishes to obtain authentication information for a user, and to update preferences for that user in the directory. Therefore the required attributes for this directory are:

- dynamic-updates = true
- jndi-sp-available = true

The 'jndi-sp-available' attribute is used to find only those services for which a JNDI service provider exists, since the application needs this driver to communicate with any found services. These required attributes are all the application currently "knows"; it does not know the address or even the access protocol of a server, nor does it have any JNDI service provider available to it. All it has are the JNDI framework classes, and a Java environment.

In order to obtain service handles to suitable service instances, the application issues the following SLP service request:

<type>=service:naming-directory

<predicate>=(&(dynamic-updates=true)(jndi-sp-available))

It receives the following service handles:

service:naming-directory:ldap://192.168.3.100/dc=eng,dc=wiz,dc=com

At this point, the application now has enought configuration information to contact the server. However, it does not have the Java class files for the LDAP service provider implementation. It can dynamically discover, download, and instantiate the necessary classfiles using SLP and the JNDI Drivers service type [7]. See [7] for an example of this process.

The JNDI application is now able to configure itself to talk to a directory server, and access the necessary JNDI service provider.

3. The Naming and Directory Service Abstract Type

Names of submitters: Jonathan Wood <jonathan.wood@eng.sun.com>
Roberto Tam <roberto.tam@eng.sun.com>
Language of service template: en
Security Considerations:

If these services are used as authentication repositories, and they are compromised, any clients of the service become susceptible to forged identities. This could result in compromised systems, forged messages, etc. Some security measure, such as SLP security, should be used to authenticate service advertisements.

In particular, it is important to understand what is trusted during the process of service discovery. If no security measures are used, all service advertisements are implicitly trusted. This scenario allows for very trivial attacks: an attacker need only insert a malicious advertisement for a bogus directory server into the service name space. It should also be noted that this scenario is open to inadvertant attacks. For example, someone may be testing an LDAP server which advertises itself according to [5]. If the test server is configured in manner similar to production servers, clients will bind to the test server and find false or missing data.

The triviality of the attacks outlined above should provide a strong case for implementing security measures. However, even with security measures it is important to understand trust dependencies. Discovery frameworks like SLP [2] provide only the mechanism for making service advertisements available and authenticated. Thus advertisements obtained via the discovery framework are valid only if a correct advertisement was originally injected into the discovery name space. For example, if an LDAP server was incorrectly

configured, or became corrupted, the bogus configuration would be reflected through the discovery framework to all clients. If the configuration error pertained to security, the client and server may end up using a weaker security flavor than necessary.

Public key cryptography is ideal for discovery frameworks, since it lends itself to store-and-forward mechanisms which can be very efficient (for example, directory agents in SLP), and can be scalably deployed (see [8]). However, a PKI introduces another complex set of trust considerations. For a PKI to be reasonably secure, the public key of the principal to be authenticated must have been obtained in a secure manner. Discovery frameworks which use a PKI must thus trust that the public key or certificate it is using for authentication is valid. For example, if a PKI were implemented such that it dynamically retrieves a public key over the network in an unsecured transfer, an attacker could substitute a false public key and thus trick the discovery client into believing a principal was to be trusted. The client could then be tricked into using a false advertisement and connect to a bogus server. [8] defines mechanisms to handle the attack just described. PKI users must also trust that the site administrator has correctly configured and populated the PKI. Finally, PKI users must trust the method used to discover key servers. DNS is often implicitly trusted in this step; insertion of a bogus address for a key server into the DNS can prevent clients from accessing their PKI.

See also the security considerations from [2].

Template text:

```
-------------------------template begins here----------------------
template-type=naming-directory

template-version=0.0

template-description=
  This is an abstract service type. This type is intended to
  encompass all services which support directory-style operations
  for looking up and searching system information.

template-url-syntax=
  url-path=  ; Depends on concrete service type.

naming-context= string M
  # A list of the names of organizational units or domains which
  # this server serves.

organization= string
  # Names spaces can be either hierarchical (as in LDAP) or flat
```

```
  # (as in NIS).
flat,hierarchical

dynamic-updates= boolean
  # True if the service's namespace can be modified dynamically;
  # false if the namespace is static.

jndi-sp-available= boolean O
  # True if a JNDI service provider is available for this particular
  # service.

master= boolean
  # True if an instance of a service is the master or authoritative
  # server for a namespace. For multi-master services, all masters
  # should set this value to true.

--------------------------template ends here-----------------------
```

References:

[1] E. Guttman, C. Perkins, J. Kempf, Service Templates and service: Schemes. RFC 2609, February 1999

[2] E. Guttman, C. Perkins, J. Veizades, M. Day. Service Location Protocol. RFC 2608, April 1999

[3] J. Wood, R. Tam, The NIS Service Type. draft-ieft-svrloc-nis-scheme-00.txt, November 1998 (work in progress)

[4] J. Wood, R. Tam, The NIS+ Service Type. draft-ieft-svrloc-nisplus-scheme-00.txt, November 1998 (work in progress)

[5] J. Wood, R. Tam, The LDAP Service Type. draft-ieft-svrloc-ldap-scheme-02.txt, June 1999 (work in progress)

[6] The Java Naming and Directory Interface (TM) Specification, Sun Microsystems, Inc. Feb 1998. http://java.sun.com/jndi/.

[7] J. Kempf, J. Wood, The jndi-drivers Abstract Service Type. draft-ietf-svrloc-jndi-drivers-00.txt, June 1998 (work in progress)