The ReachOut Project
Developer's Guide

Jeremy T. Bouse
Jeremy.Bouse@UnderGrid.net

February 14, 2003
(Revision 1.2)

Contents

Listings

Abstract

The goal of the ReachOut Project is to design a fully modular system which will serve as the core framework able to load and unload modules dynamically upon execution or by configuration options. Along with the Application Program Interface (API) for the modules which will allow for new modules to be created and added to the system without the need to rebuild the entire core framework again.

The core framework should provide all functions necessary to hand the necessary network communications in an Address Family (AF) independent manner along with providing the hooks needed for module initialization and operation. The core framework should define daemon and be able to execute independent of any modules which extend the daemons ability.

The module API should provide a complete outline of the initialization and operation any given modules should implement to be compliant. Any compliant module should work with the core framework provided the API version is the same or backwards compatible. Modules themselves will be divided into seperate classes, with the initial design being for an authentication, service and database classes.

The modular design will allow for the various authentication, services and back-end server database module support to be written by various groups and organizations for various platforms and vendors allowing for a versatile client/server architecture for dynamic services.

Id: reachout.tex,v 1.2 2003/02/14 04:06:45 undrgrid Exp

1 Core framework

The ReachOut core framework should operate properly independent of any ReachOut modules added at anytime dynamically at run-time. These will either be loaded automatically or based on directives given in the configuration file.

1.1 Network Communication

1.1.1 Protocol

The ReachOut core framework should be able to operate independent of any particular AF although primary support should be geared towards Internal Protocol version 4 (IPv4) and Internal Protocol version 6 (IPv6) and should also be able to handle any future protocols.

1.1.2 Transport

The ReachOut core framework should utilize a stable Transmission Control Protocol (TCP) transport protocol allowing for a two-way handshake between the client and server. This handshaking ability will allow the client and server to be able to verify and operation was successful prior to terminating the connection. Unlike User Datagram Protocol (UDP) the TCP transport this will not require the need to maintain a timestamp from the client being sent to the server as the communication will be confirmed at all steps of the connection.

1.1.3 Security

The ReachOut core framework while it should not be rigid enough to prevent it, at this time there is no decision as to whether the transport should be done over Secure Socket Layer (SSL)/Transport Layer Security (TLS) at all times, it should be an option for the client if the server supports it, or be made a module which can be turned on or off based on configuration. The general thought is though that if SSL/TLS support is included it should listen on a port number one higher than the standard port.

1.2 Packet

The ReachOut core framework should handle a packet designed to be machine independent allowing communication regardless of the endian of the client or server. To accomplish this the packet should be designed to utilize the flexibility of eXternal Data Representation (XDR) which allows for machine independent packet encoding. XDR allows allows for multiple versions of a particular service protocol to be handled by both the server and client allowing for the enhancement of services while still maintaining backwards compatibility with older versions.

1.2.1 DHSP Packet

Line 59 of the XDR listing shows how the Dynamic Service Hosting Protocol (DSHP) shows how the lowest level of the packet is design. As you can see this is able to handle many different versions all declared within the ProtocolVersion enumeration (Line 48) which are backwards compatible. This is the structure that will be passed between the client and the server during any transaction.

Listing 1: DSHP XDR Packet

```
48 enum ProtocolVersion
49 {
50     PROTOCOL_ERROR,
51     DSHPv1        /* DSHP version 1 */
52 };

54 enum DSHPPacketStatus
55 {
56     INVALID_PROTOCOL
57 };

59 union DSHPPacket switch (ProtocolVersion protocolVersion)
60 {
61     case DSHPv1:
62         DSHPv1Packet          dshpv1;
63     default:
64         DSHPPacketStatus      status;
65 };
```

1.2.2 DHSP v1 Packet

The actual data within the packet itself can be left in clear text or can implement some form of data encryption. Line 28 outlines the enumeration of what packet encoding is available but only those within the DHSPv1Packet (line 40) union are available for a given DSHP version.

Listing 2: DSHP v1 XDR Packet

```
22 struct DSHPv1Plaintext
23 {
24     DSHPv1MessageType          type;
25     DSHPv1Message              msg;
26 };

28 enum EncryptionType
29 {
30     ENCRYPTION_ERROR,
31     PLAINTEXT
32 };

34 enum DSHPv1PacketStatus
```

```
35 {
36     INVALID_MODULE,
37     XDR_DECODE_FAILURE
38 };

40 union DSHPv1Packet switch (EncryptionType encryptionType)
41 {
42     case PLAINTEXT:
43         DSHPv1Plaintext      pt;
44     default:
45         DSHPv1PacketStatus   status;
46 };
```

1.2.3 DHSP v1 Message

Once any data encoding is handled you get to the actual packet message (line 12) which is designated by the MessageType (line 1) enumeration. A MessageType can be declared but are only available once defined within the DSHPv1Message.

Listing 3: DSHP v1 XDR Message

```
1  enum MessageType
2  {
3      AUTHENTICATION, /* Authentication module */
4      SERVICE       /* Service module */
5  };

7  enum DSHPv1MessageStatus
8  {
9      INVALID_MESSAGE_TYPE
10 };

12 union DSHPv1Message switch (MessageType messageType)
13 {
14     case AUTHENTICATION:
15         opaque            auth<>;
16     case SERVICE:
17         opaque            service<>;
18     default:
19         DSHPv1MessageStatus status;
20 };
```

1.3 Configuration

The ReachOut core framework should make use of the eXtensible Markup Language (XML) which will allow the configuration file to grow and expand with the project. It should allow for the needed flexibility, as well it will make the configuration file parsing to be handled easily and cleanly. The configuration should be locked by the daemon process while it is

executing and save itself upon successful shutdown of the daemon process. This is required for a couple reasons, the first of which being that the ReachOut server may at times need to send a configuration update back down to the client which will need to be saved. This also allows for overwriting mistakes if the configuration is modified incorrectly while the daemon process is running. If the daemon is running appropriately with the configuration in memory then saving that configuration back will ensure that no mis-configuration has occurred.

Although it has not been envisioned at this time for a need to make major changes to a configuration after it has been released and published in Document Type Definition (DTD) format, as well as wanting to maintain complete backwards compatibility. Should the need arise to account for this situation then the ReachOut core framework should be able to handle the migration of the configuration via eXtensible Stylesheet Lanuage Transformations (XSLT) mechanisms.

Listing 4: XML Configuration file format

Listing 5: XML Configuration file DTD

2 Module API

The ReachOut modules should not require the need to be compiled at the same time as the daemon, thus allowing for new modules to be added dynamically to the system provided they are compiled against the same API version. Thus the system should have some means of maintaining the API version and check modules against when a new module is attempted to be loaded into the daemon process.

With the attempt at backwards compatibility older modules should be able to execute with newer daemons so they should be loaded and internal module checks tested to determine if it should loaded otherwise unload it. However the reverse may not always be true, due to upgrades and enhancements, so modules with a newer API version than the daemon should not be loaded. This may require the addition of a configuration stanza to keep track of module states, but the initialization process should be fast enough to not require this.

All ReachOut modules will similar declaration structure which will contain all the necessary information for the module to be registered with the core framework when it is loaded. A module need not support all the functions in the declaration as it is meant to be flexible for all general modules not any one specific module. In the case a module does not implement a procedure then the declaration should include it as a NULL value. The Module declaration listing should demonstrate how a new module would be written to declare itself to the core framework.

Listing 6: Module XDR declaration

All ReachOut modules, with exception of the Authentication modules which are always loaded upon startup, will require some configuration in order to operate properly and allow the core framework know which are to be loaded. While all modules will have some common configuration directives, each class of module may have addition configurations covered within their own section. The Module Configuration DTD listing will demonstrate the general configuration for all modules and leave module specific configuration details for later.

Listing 7: Module Configuration DTD

2.1 Authentication

The ReachOut daemon should be able to handle any means of authentication via separate modules per each authentication method or class of methods. Possibilities of authentication could include things like plain text, Simple Authentication and Security Layer (SASL), X.509 Public Key Infrastructure (PKI) or any other cipher algorithm; however due to the modular design there should be no limitation to the possible methods to authenticate the client to the server.

The ReachOut daemon should attempt to load all available authentication methods at startup and again upon re-reading the configuration file and restarting. This will allow for the client and server daemon process to agree upon a commonly supported method through the connection handshaking, as there is no guarantee that the client and server be loading the same authentication methods.

The initial development will go towards either a plain text and some form of cipher algorithm, additional authentication modules that could eventually be designed and developed could be SASL or X.509 PKI.

2.2 Service

The ReachOut daemon in keeping with its mission to be a flexible modular design should be able to handle a multitude of dynamic services which can be loaded at startup based on configuration directives. Each service module should be loaded only when there is a configuration directive calling for that module. As with Authentication modules, the API version should be checked and should not load in the event that the module API version is newer than the daemons API version. If there are no services loaded after attempting to load all configured services then the daemon should shutdown with an appropriate error message.

The initial development will go towards the Domain Name System (DNS) service module, additional service modules that could eventually be designed and developed could be Mail eXchange (MX) or Virtual Private Network (VPN).

Each ReachOut service module will have its own directives to properly configure it with the core framework. The Service module configuration DTD listing will demonstrate the configuration for a service module that should be included within the general module configuration mentioned earlier.

Listing 8: Service module configuration DTD

2.3 Database

The ReachOut daemon should be designed with its back-end database support to be modular as well to allow for the use of various database vendors depending on the operating environment of the server daemon. As each database format has subtle differences from one another for the lower level functions which can all be abstracting into the modules so that all modules will have the same available functions for the ReachOut core framework to call to provide the same actions regardless of the individual steps needed to perform it.

The initial development will go towards the MySQL database module, additional database modules that could eventually be designed and developed could be PostgreSQL, Oracle or even Lightweight Directory Access Protocol (LDAP).

Each ReachOut database module will have its own directives to properly configure it with the core framework. The Database module configuration DTD listing will demonstrate the configuration for a module module that should be included within the general module configuration mentioned earlier.

Listing 9: Database module configuration DTD

3 Software Licensing

3.1 Core framework

All ReachOut core framework code will be released under the GNU General Public License version 2 (GPL-2) or any later version and must contain the following copyright declaration at the beginning of any source file distributed.

```
The ReachOut Dynamic Service Hosting
Copyright (C) 2003  UnderGrid Network Services

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version
2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111 USA
```

3.2 Module API

All ReachOut module API code will be released under the GNU Library General Public License version 2.1 (LGPL-21) or any later version and must contain the following copyright declaration at the beginning of any source file distributed.

```
The ReachOut Dynamic Service Hosting <Module name> Module
Copyright (C) 2003  UnderGrid Network Services

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111 USA
```