Using and Extending PVProxy

David Hovel
GroupWyse Consulting
June, 2011
Version 0.3

Overview

The ProbeView iPhone application allows its users to 'ping' network sites and determine TCP-based service availability. However, many users want to know if important servers and services are active and available behind their business or home firewalls. To 'tunnel' across a firewall, there must be a participating agent on the other side. PVProxy is exactly that.

For example, assume you have a single-office business and your internet connection is configured as most are today with some type of modem/router combination that includes a firewall. The addresses used on the 'private' side of the network cannot be used on the internet, and small organizations tend to have only one public IP address. If PVProxy is running on a machine on the 'private' side and the router is configured to forward requests to it, the **ProbeView** iPhone application can pass 'ping' requests to the proxy. The proxy then performs the request *in the context of the private network* and returns the result.

Beyond simple proxy behavior, PVProxy can easily be extended to perform arbitrary, user-defined tests. This document explains how to create extensions.

Getting PVProxy

There are two versions of the PVProxy server, one for Windows 7 and one for Mac OS X (10.6+, Intel Macs only). These can be downloaded from the GroupWyse web site.

Running PVProxy

PVProxy is a very simple web server. **ProbeView** sends it a standard HTTP URL requesting data and server responds as necessary. The resulting data is XML (EXtended Markup Language).

In order to operate as a web server, the machine running PVProxy must receive incoming HTTP requests on a configured TCP port. The default port is 8080, but you may change this as necessary. You must configure your firewall to pass requests on the chosen port to the machine running PVProxy.

Security with PVProxy

You may configure PVProxy to require a password for all requests. However, the messages exchanged between PVProxy and its client are not encrypted (i.e. HTTPS is not used). This

may seem like a security threat, but remember that PVProxy is only reporting the presence of services and hosts on the other side of a firewall; it is not directly reporting their capabilities or information.

Running PVProxy on Windows

PVProxy is written in C# using the .Net 4.0 run-time. This software is available on all Windows 7 and Windows Vista machines. For other machines, visit Microsoft's website and see if the .Net 4.0 redistributable is available for your machine.

Because of Windows' security features, you will probably need to run PVProxy with administrator privileges. When you start the **PVHTTPServer** program, the main screen will appear. It allows you to set the active server port (default value is 8080). It also allows you to set a password if you desire.

The Windows version uses certain features of Windows networking to simplify HTTP message routing. The "HTTP Prefix" is set, by default, at the most general value. If this doesn't work for you, see http://msdn.microsoft.com/en-us/library/aa364698(VS.85).aspx?ppud=4 for details on the allowed values of the prefix setting.

By default, the Windows version logs all request traffic. You can configure this and other settings using the menu items and controls on the toolbar.

The Windows version of PVProxy has a 'client' window you can open using the main menu. The client window allows you to send requests to the local PVProxy server or any other, examine the generated request URL and see the results of the request. See "Testing Your PVProxy Installation" below.

Running PVProxy on Mac OS X

On OS X, PVProxy is a Cocoa (graphical) application; it is only available for Intel-based Macs running OS X version 10.6.5 and later. Although it may build for non-Intel platforms, doing so has not been tested.

OS X PVProxy contains a standard configuration sheet that allows you to set the TCP port, password and control certain behaviors of the application.

Testing Your PVProxy Installation

There are two ways to test your PVProxy installation without using the iPhone application.

First, you can use any web browser. If you send a correctly formatted URL, PVProxy will reply with a small XML document. For example, open a web browser on the machine running the proxy and enter the following line in the browser's address field:

http://localhost:8080/?type=ping&host=localhost&pxypw=password

Replace the string '8080' with the port number you've assigned. Also replace the word "password" with whatever password you have chosen for the proxy. (If you don't use a password you can leave off the "&pxypw=" section of the URL.) You should receive a response like this:

```
<PVHTTPServer Type="Ping" DateTime="2010-12-07T09:05:36.042375-08:00"
Host="localhost"><Response RoundTrip="0" IPAddress="127.0.0.1"
Status="0"/></PVHTTPServer>
```

The URL you entered is a request that the local computer ('localhost') ping itself.

The second way to test PVProxy outside of the iPhone application is to use the Windows version's "client" test window. If you launch PVProxy on Windows you'll see that there is an Options menu item that will show a "client" window; there is also a toolbar button that does the same thing.

This client window is a test form that can test any PVProxy running anywhere. You just enter the name, port and password of the proxy you want to test, along with the details of what you want the proxy to test. The client form will generate the correct URL, send the message and show you the response.

PVProxy Messaging

Because the proxy is designed to support the iPhone application, the data exchanged between the iPhone application and the proxy is very simple. The design is loosely based on REST (http://en.wikipedia.org/wiki/Representational_State_Transfer) and entails a simple URL HTTP "GET" request and an XML (text/xml) response.

Request Message

The query terms of the URL are:

- **type=** Normal values are '**ping**' and '**tcpping**'. Any other value is considered to be the name of an extension (see below).
- **host=** This is the host name given in standard internet domain or IP addressing format.
- **port=** This is the port number, which must be a positive integer between 0 and 65535.
- **pxypw=** This is the proxy password.

Since the request message is a URL with query terms, there are certain limitations as to embedded spaces and non-ASCII characters.

Response Message

The response message is a small XML document with a root tag name of **PVHTTPServer**. The attributes of the root tag are **Type**, **Host** and **Port** (specified as in the request URL) and **DateTime**, formatted as a SQL date.

The inner tag is **Response**. Its attributes are:

- **Status** This indicates the result of the operation. Any value other than zero indicates failure.
- **RoundTrip** This is a number, defined as the number of milliseconds required for the proxy to complete the transaction.
- **IPAddress** This is a standard IPV4 address and port number, indicating the result of DNS querying.
- **ErrorMessage** This is a string error message. It may be returned even in cases where the status is zero.

Extending PVProxy

In order to test its targets, PVProxy performs standard (ICMP) 'pings' or TCP open/close (port probe) operations. These functions are important, but they don't let you know if a service is truly functional beyond mere availability.

When you create an extension to PVProxy on either platform you'll be defining a new message type. You must give it a name; this name can be entered on the iPhone application when you set up a 'target'. The name you define takes the place of the standard message types ('ping' or 'tcpping') in the URL query string.

Upon receipt, PVProxy routes the request directly to an extension object (instance of a software 'class'). In an extension, the meaning and contents of the other query terms and the response XML attributes are entirely up to you. The only constraint is that the status code must be zero for successful operations and non-zero otherwise.

As an example, imagine that you'd like to know if your database server is functioning correctly. To do this, you might need to perform a simple SQL query and check its results. You can't do this with the off-the-shelf PVProxy components because they cannot know the details of your database and its operation. However, you create your own specialized version of PVProxy.

In the case of a database server, you might decide to create a new extension message type of 'dbok'. Then you can extend PVProxy on your chosen platform to recognize this message type and perform the correct SQL query, verify the result and return the correct status to your iPhone. If your extension returns a status of zero, the iPhone application will consider the request to have been successful.

Overview of Extending PVProxy

Both versions of PVProxy are similar. They are both web servers, and each application services incoming HTTP requests on separate threads. Along with handlers for the default request types, each application has a set of generators (factories) that know how to create 'extension request' objects for the special extension types. These extension objects are created on demand as requests arrive and run on independent request threads.

When a request for a special type is received, the generator registered for that type is asked to create an object to handle the request. The details of the object and its behavior vary between the platforms, but the intention is the same: *you must create a 'request handler' class and a 'generator' class for each type of special request you want your PVProxy version to handle.*

Each version of PVProxy comes with a default (trivial) extension called 'xtest1'. This extension is used to demonstrate how to extend PVProxy.

It may seem somewhat limiting that the incoming REST-style URL contains only the request type, hostname and port number. However, remember that an extension can treat this information in any way it cares to. They could, for example, provide the name of a stored SQL procedure to run and the expected result. As for the return values (round trip time, status, error message, etc.), the only reserved value is the status: any status other than zero is considered to be a failure condition. This gives each proxy extension a wide range of information to return about problem conditions.

Extending PVProxy on Windows

The Windows version of PVProxy is written in C# using .Net 4.0 and Visual Studio 2010.

To extend PVProxy, you need to create a C# (.Net) library (DLL). This DLL must reference the **PVHTTPExtensions** library that comes with the Windows version of PVProxy; it is located in the applications folder where you installed PVProxy.

To define your request processing class you must subclass the abstract class **ProxyExtension**, which is the request handling object. Then declare a subclass of **ProxyExtensionSet**. This class is a collection of one or more extension generators. In its (required) parameterless constructor, use the generic generator class **ProxyExtensionGenerator<>** to create a generator for your new extension. For a complete example, see the sample Visual Studio 2010 proxy extension project called **DemoExtension** available for download from our web site. Your extension set can contain as many extensions as you desire.

When the PVProxy server is started it browses all the libraries (DLLs) in its application folder to find declared subclasses of **ProxyExtensionSet**. When it encounters one, it locates that subclass' single parameterless constructor and uses it to create an instance of the set class. It then enumerates the set and adds each generator in it to the server's extension generator collection. The PVProxy server displays which extensions were found and loaded when the server is started. You can also check the "Help/About" dialog to see if your library was successfully loaded.

Every subclass of the abstract class **ProxyExtension** must override the **ProcessRequest** method and set the various return values as necessary (error message, status, IP address, round trip time, etc.). The rest is automatic.

Because each request is on its own thread, an extension can safely perform blocking operations such as database queries or even start additional threads and wait for them to complete.

The **DemoExtension** project comes with two examples: **DemoExtension** and **ShellableExtension**.

Example: DemoExtension

The class **DemoExtension** uses the string token 'demox1' as its type. This is the string you would enter on an iPhone, browser or Windows PVProxy client application in order to invoke this extension.

DemoExtension is merely an 'ack' (acknowledgement)—it just returns successfully.

Example: ShellableExtension

The **DemoExtension** project contains a more interesting extension called **ShellableExtension**. This class uses the string 'shelly' as its extension identifier and interprets the hostname string as the name of a batch file to be executed. The code then waits for the batch script to complete and returns its exit code as the status value.

While this might seem like a serious violation of security, it is intended only as a demonstration project. You should always use passwords to protect PVProxy servers, and any extension like shelly should utilize an internal table of valid test names to help defeat hackers.

Extending PVProxy on OS X

The OS X version of PVProxy is written in Objective-C using XCode 3.2 on OS X 10.6.5. The server is based on **CocoaHTTPServer** (http://code.google.com/p/cocoahttpserver/). It was extended to incorporate the REST-based PVProxy URL handling and to make its behavior more general.

To extend PVProxy on the OS X platform you must download the source code and build your own private version of PVProxy. You will need to create your own subclasses of **MTHExtensionResponse** and **MTHExtensionGenerator** for each new extension you want to create.

Please read the detailed comments in **ExtentionTest.m** and **MTHExtensionGenerator.h** for more information. The files **ExtensionTest.*** declare and define the 'xtest1' extension, and the files **XGenTest.*** define the generator for that extension.

When you've done that, open the file **MTHServerExtensions.m** and edit the method **initializeExtensions:** to create the extension generator for your new extension and add it to the server's dictionary.

As with the Windows version, the logging area of the main screen will display the extensions that have been registered in the system.

Testing Proxy Extensions

You may test extensions using any browser or with the Windows client of the PVProxy application as usual. The only difference is that you substitute your extension's name in the URL 'type' query term.

Connecting Extensions to the ProbeView iPhone App

Once you have verified that the extension's behavior is sound, you may use it from the iPhone application. To do this, bring up the Options page and set the "**Enable Extensions"** switch to **On**. Then create a target, mark it as type "Proxy Extension" and enter the extension's name in the correct text field. When you test the new target you should see the resulting request and response on the main window of your PVProxy server.