Perspectives Developer Documentation

Dan Wendlandt
dwendlan@cs.cmu.edu

November 28, 2007

The goal of this document is to give you an overview of the Perspectives source code and hopefully get you started along the path of making changes to it. We assume you have already carefully read the Perspectives paper and understand the conceptual design on the system. Additionally, we place a particular emphasis on making client-side changes, since adding a new policy or adding Perspectives to a new piece of client software is the most likely cause of anyone needing to become familiar with the Perspectives source code.

1 Overview of Code Layout

There are three main programs withing the the Perspectives source code, each of which corresponds to a sub-directory within the source. Prior to describing modules of the code in depth, we describe the overall layout to help you zero on the portions of the code you actually need to change.

1. **notary-client:** Provides a library to connect to multiple notary servers, verify their replies, and implement simple quorum duration policies.
2. **notary-server:** A simple server daemon that responds to client requests based on the contents of its local database.
3. **key-scanner:** A program that adds entries to the database of the notary server by probing services at regular intervals. This program should be run on the same machine at the notary server code. Currently, we only have an SSH scanner, but an SSL scanner is in the works.

Additionally, there are a few other sub-directories, which we explain here:

1. **common:** Contains headers and code likely to be used by multiple of the notary programs. This includes packet parsing and packet creation routines, cryptographic functions, and database functionality.

2. **utilities:** Contains simple programs use to test, demonstrate, or benchmark functionality related to any of the notary programs. These programs can be a good place to start learning about the code because they often exercise core functionality in very simple ways.

3. **config:** Contains example configuration files for clients, servers, and scanners. It also contains a small database you can use to run the server (small.db) and a text file (small.txt) describing the contents of the database.

4. **openssh-4.4p1:** This is the entire openssh code base, slightly modified to use the notary-client libraries to authenticate un-cached keys.

5. **port-scanner:** This is a slightly modified version of nmap to do extremely parallel TCP layer port scanning to identify if a host is likely running a particular service on its standard port.

6. **keys:** Contains example keys used for development and testing and instructions on how to generate keys for actual use. Many utility programs are hardcoded to access these keys.

2 Building the Code

The code should build with gcc and make on any Unix system, though it has been tested only on Linux and BSD.

Build Dependencies: There are two primary dependencies: OpenSSL and Berkeley DB, both of which can be somewhat troublesome. For Berkeley DB, you need 4.6, which is the latest version. You can download it at http://www.oracle.com/technology/software/products/berkeley-db/db/index.html and build it yourself or install the *development* package via a package manager like apt-get. Any recent SSL version will work, and you can download it at http://www.openssl.org/source/ or just install libssl-dev from a package manager. However, a build can fail if it uses header files and libraries with conflicting version numbers. To fix this, specify the exact location location of your SSL install (usually /usr/local/ssl) using both -I and -L .

Makefiles and Build Order: Currently, there is no master makefile, so you have to invoke the makefiles in each directory manually. You need to build the 'common' directory first, as all other code links against it. Makefiles in 'notary-client', 'notary-server' and 'utilities' can then be invoked independently.

3 Common Code

This 'common' directory contains library functionality used by multiple binaries within the notary code-base. The following table describes a bit about each file.

File Name	**Purpose**
common.h	Main header file for all notary code. Contains the struct defining the notary packet header (`notary_header`) and also the `ssh_key_info_list` struct, which represents a list of observed keys (used by both notary server and client). Also includes many important #define values and macros.
bdb_storage.c	Wrappers to simplify accessing and modifying observed key data within a Berkeley DB.
debug.h	Defines the DPRINTF macro used throughout all code. Every program must include a global 'notary_debug' variable set to the desired debug level. See file for details.
notary_util.c	Contains utility functions to covert between packets and `ssh_key_info_lists` as well as code to print notary related data.
notary_crypto.c	OpenSSL wrappers to load public/private keys from buffers/files and sign or verify signatures.
net_util.c	Helper functions to parse/print IP addresses and perform DNS lookups.
patricia.c	A Patricia Trie for prefix look-ups, used to blacklist IP ranges from scanning.
list.h	A linked-list based on the linux kernal's data structure.
flex_queue	A vector that supports only simple queue operations.

Invoking 'make' builds the libnotarycommon.a archive, which is used when linking to create notary binaries in other directories.

4 Notary Client Code

Code within the 'notary-client' directory does not build a binary, rather it creates a library to link against when adding Perspectives functionality to a third-party client application (e.g., the openssh client, or the Firefox browser). When building these applications, you will need to link against the libnotaryclient.a file created within this directory, as well as the libnotarycommon.a file created in 'common'.

There are only three files in this directory:

File Name	**Purpose**
notary_local.c	Implements the main SSHNotary interface used by clients. This includes parsing configuration files describing notary servers, generating a request packet and verifying and parsing the reply. All functionality in this file is independent of the actual socket layer used communicate with the server. (We should probably also remove all file I/O to make it completely platform independent.)
contact_notary.c	Non-blocking posix sockets to contact notary servers and receive replies using UDP. Handles simple retransmission.
client_policy	Implements client policies based on the replies from multiple servers. Currently, this is just a simple quorum duration policy.

An example of how to use the interfaces provided by this client code can be found in 'utilities/full_client.c'. While the many interface names refer to 'ssh', the interface will remain essentially the same for 'ssl'.

The file 'config/client_config.txt' provides an example of the file format required to inform the SSHNotary interface about notary server addresses and the associated public keys. This file simply uses the public key in the 'keys' directory as the public key for all notary servers, which is ok for development purposes.

5 Notary Server

Even if you are planning only on creating a Perspectives client, you will still want to understand enough about the server in order to build and configure it correctly for testing. Typing 'make' will build the executable 'notary-server/notary_server'. It takes all configuration parameters from a single file of name-value pairs. We provide an example of such a file in

‘config/server_config.txt’. It allows you to specify the IP address and port for the server, as well as two Berkeley DB parameters. The first database parameter indicates the “DB environment directory filename”, which is a directory that will contain the database as well as metadata related to the database (e.g., logs and locking info). The second is the DB filename itself. If you want to run a server without a scanner (the easiest approach if you’re developing a client) you can copy the file ‘config/small.db’ to some directory that will serve as the database environment directory (the current server config sets this directory to be ‘/notary’ by default) .
TODO: Describe outline of the server code.

6 Utilities

The ‘utilities’ sub-directory contains the source for many (often) simple binaries that demonstrate, test, or modify notary behavior. We will briefly describe those most likely to be of interest to developers. All example shell commands below are executed from within the ‘utilities’ directory.

6.1 full_client

This simple program almost exactly mirrors the API calls that a real-world client application would use to load a notary configuration from file, contact the notary servers, and interpret the results. As you can see, doing so is pretty simple. However, this client does not use the notary replies to invoke a “key acceptance policy”, since it has no real key to test. Still, it should give you a very good idea of what your client will need to do in order to use a notaries. Consider if someone wanted to find all keys observed for the host gs5025.sp.cs.cmu.edu on port 22 ?

Example Usage:
```
./full_client ../config/client_config.txt gs5025.sp.cs.cmu.edu:22
```

Note: This example assumes that you have a server running and have modified the client configuration to point to this server.

6.2 db2file and file2db

Database files used with Berkeley DB (e.g., ‘config/small.db’) are binary file and thus are unreadable and unmodifiable from the command-line. These two utilities convert between a text representation of a notary’s database and the binary Berkeley DB format. The following command prints an entire database to a text file:

Example Usage:
```
./db2file ../config/small.db db_dump.txt
```

Similarly, if you modified db_dump.txt to change some of the database contents, and wanted to change store those changes back to a new database file, you could do the following:

Example Usage:
```
./file2db db_dump.txt modified.db
```

A couple of important things to note. First, when parsing from text to to the database format, file2db only looks at the integer representation of timespans (i.e., it ignores the printed dates). As a result, changing the date text will not change the timespan. You need to change the integer value itself. If you want the contents of a database created by file2db to contain valid signatures, you must provide a third parameter that indicates a valid private key (e.g., '../keys/private.pem'). If no key is provided, the signature contents are simple garbage. Also, the data format is very particular, so be careful to replicate it exactly. Data about a single key (i.e., the key and all of its timespans) is terminated by a blank line. Information about all keys from a single host is terminated using the a line that contains only the text "End Host".

Note that these two utilities combine to provide a simple tool for replicating a single database on multiple servers that have different public/private keys. This can be useful for testing more complex scenarios.