**Overview**

Configuring security mechanisms in HTCondor can be complex and confusing. Debugging a failed setup can be difficult. Confirming that a setup is working exactly as expected can also be difficult. This document proposes and specifies a design for a command-line tool that helps debug, diagnose, test, and verify the security configuration settings for an HTCondor installation. The tool will be called “condor\_ping”.

Currently, there is a tool called “condor\_test\_auth” which lets users and administrators test various authorization policies. However, it does not actually do any network communications, test authentication, encryption, or integrity mechanisms. As such it doesn’t detect, for example, when authentication is desired but fails, or if the user is mapped to something unexpected. Furthermore, the API for condor\_test\_auth is not particularly user-friendly and is intended mostly for use in the HTCondor regression test framework.

Similar to “globusrun –a”, this tool will perform the actual network connection, negotiate security settings, perform authentication, enable encryption and integrity checks as specified in the security policy, perform user mapping, and finally test authorization. The tool then displays a summary and/or analysis of what transpired. Various options for output will exist, ranging from “-quiet” (no output, just state) to “-verbose” which displays everything.

A more comprehensive analysis of security policy could be done by repeatedly invoking this tool to attempt an exhaustive security sweep, using all combinations of security policies and mechanisms for the client and trying all authorization levels. However, this document focuses only on the condor\_ping tool itself.

**Architecture**

**Command Line API**

Because different HTCondor daemons may have different security policies, and perhaps different policies depending on the level of access required (READ/WRITE/DAEMON/etc), the user will specify:

1. The client security policy
2. Which daemon to communicate with
3. Which commands and/or authorization levels they would like to check

The default client security policy will be what the current configuration environment specifies. However, to facilitate testing other scenarios, the user will be able to provide a configuration input file using the “-config” command line option, in regular condor\_config style formatting.

The daemon can be specified using the “-address” flag and the address in sinful string format. This removes the need to query the collector to locate the daemon, since the collector may have a different policy, different mechanisms, and is not (always) the desired target of this test.

Querying using “-pool” and “-name” will also be supported. When used in this mode, the collector will be queried for the desired daemon using the existing security settings, BEFORE the configuration file is loaded.

The commands to perform can be specified in three different ways: i) as an authorization level; ii) as a command int; iii) as a command name. You can pass several of them on the command line, or use the special keyword “ALL” which tests each authorization level.

**Internals**

The security policy to test is loaded as a local config file.

It is possible that the security policy specifies that no security negotiation be used (the 6.2.X style communications). In that case, the tool will not actually perform any communications and simply report that the command was sent completely unsecured (and why).

In all other cases, we now have a security context that we know will involve at least some network traffic. Once the security context is initialized and we know which address and port to connect to, the actual communication begins. The “new style” (that is, 6.3.X and newer) protocol will be used which means that the raw CEDAR command int is always DC\_AUTHENTICATE. As it already exists, the client then sends the raw command int followed by a classad which contains security policy. The Command attribute within this classad will be set to a new command, DC\_AUTHORIZE, with the SubCommand attribute set to the eventual DaemonCore command that would be performed within this session. When this happens, DaemonCore establishes the session as normal using the authorization level of SubCommand, including authentication, the enabling of encryption and integrity checks, and testing of authorization, but does not actually invoke the command handler for SubCommand.

This does require changes to the wire protocol, and as such this tool will only be able to be give full details when run against HTCondor daemons that are 7.9.5 and newer. If it fails, it can fall back to using DC\_AUTHENTICATE which give a lot of info but excludes the authorization step. The nature of this is such that all communication will be done via TCP. Even when HTCondor does UDP internally, it uses TCP to establish the session exactly as above, by sending DC\_AUTHENTICATE via TCP and specifying the eventual UDP command as the SubCommand. This tool is piggybacking on very similar functionality.

In the event of success, we can simply look up the session and extract all the needed information from the associated ad. In the event of failure, no session is created. However, because the startCommand() API takes an ErrStack and it is used extensively in the security code, much information can be gained even in the event of failure. However, some changes may need to be made to enrich the existing propagation of errors using the ErrStack, which will result in improved output for this tool as well the debugging output for all other communications in HTCondor.

**Development Plan**

1. Clone condor\_advertise, rename it, and get it building cleanly in cmake (1 days)
2. Modify the new tool to accept the command line arguments (1 day)
3. Have the tool use the supplied security policy (1 day)
4. On success, have the tool pull out and print crucial information from the resulting session ad (1 day)
5. As needed, instrument code with more error codes to propagate more interesting information even in the case of failure (1 day)
6. Massage all the data into a nice-looking report (2 days)