The Trellis Security Infrastructure: A Layered Approach to Overlay Metacomputers

Morgan Kan, Danny Ngo, Mark Lee, Paul Lu,
Nolan Bard, Michael Closson, Meng Ding,
Mark Goldenberg, Nicholas Lamb, Yang Wang
Department of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8
Canada
Email: paullu@cs.ualberta.ca

Ron Senda and Edmund Sumbar
Computing and Network Services
University of Alberta
Edmonton, Alberta
Canada

***Abstract*— Researchers often have access to a variety of different high-performance computer (HPC) systems in different administrative domains, possibly across a wide-area network. Consequently, the security infrastructure becomes an important component of an *overlay metacomputer*: a user-level aggregation of HPC systems. The Grid Security Infrastructure (GSI) uses a sophisticated approach based on proxies and certification authorities. However, GSI requires a substantial amount of installation support and it requires human-negotiated organization-to-organization security agreements.**

In contrast, the Trellis Security Infrastructure (TSI) is layered on top of the widely-deployed Secure Shell (SSH) and systems administrators only need to provide unprivileged accounts to the users. The contribution of the TSI approach is in demonstrating that a single sign-on (SSO) system can be implemented without requiring a new security infrastructure. We describe the design of the TSI and provide a tutorial of some of the tools created to make the TSI easier to use.

I. Introduction

Some workloads and experiments in computational science require large amounts of resources, both in terms of capability and capacity. Highly-capable systems, such as those with large processor counts within a shared memory or high-performance networks, tend to exist entirely within one administrative domain. In capacity computing, where high throughput is often the main goal, aggregating different high-performance computing (HPC) systems is a common technique to provide the needed capacity. The individual jobs of the capacity-oriented workload can be mapped to different servers in the aggregate. However, when the HPC systems reside in different administrative domains, there may be important issues related to cross-domain security.

A practical problem that exists today is that many researchers have access to a variety of different computer systems in different administrative domains (Figure 1). The researcher merely has an account on each of the systems; the different administrative domains may not have and may not be interested in entering into cross-domain security arrangements. For example, Researcher A has access to his group's system, a departmental system, and a system at a high-performance computing centre. Researcher B has access to her group's server and (perhaps) a couple of different high-performance computing centres, including one centre in common with Researcher A. It would be ideal if all of the systems could be part of one metacomputer. But, the different systems may be controlled by different groups who may not run the same security software or have not have negotiated cross-domain security policies. Yet, Researchers A and B would still like to be able to exploit the aggregate power of their systems.

Understandably, systems administrators desire modern and effective cross-domain mechanisms for authentication, authorization, and data management. For example, grid computing and the Globus Alliance [1] include wide-ranging efforts to define new service and software standards for sharing general computer resources (not just HPC systems) across different organizations. The Grid Security Infrastructure (GSI) [2], [3] and, more generally, the Globus Toolkit, use Web services, X.509 certificates, and other well-known standards to build a scalable, cross-domain security infrastructure. Within the original design goals of GSI, the resulting system is a modern and elegant cross-domain security solution. However, GSI is non-trivial to set up, requires administrator-level security agreements, and is not yet widely deployed.

There is one cross-domain security tool that is both widely trusted (from both technical and social perspectives) and is almost universal across HPC and personal computing systems: the Secure Shell (SSH), especially the OpenSSH implementation [4]. SSH supports public-key authentication and secure channels using strong encryption, has the *forced command* mechanism for authorization, and can use familiar local protection mechanisms for data sharing and other aspects of authorization. Consequently, the Trellis Project [5] has proposed that a practical and portable security architecture based on SSH can create an *overlay metacomputer* (Figure 1): a user-level aggregation of HPC systems [6], [7]. The overlay metacomputer is per-user and can be as simple as one computer, or as complicated as hundreds of computers in many administrative domains.

The primary purpose of this paper is to describe and provide a tutorial of the Trellis Security Infrastructure (TSI). Working

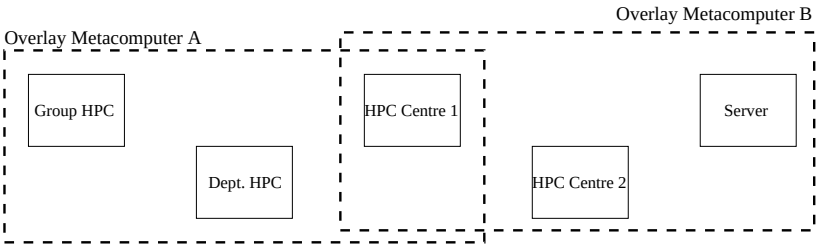

Fig. 1. Overlay Metacomputers

entirely at the user-level with unprivileged accounts, TSI forms the basis for all the other Trellis Project efforts in global scheduling, data movement, and distributed file systems. As per the desired *single sign-on* (SSO) capability, once the TSI has been configured and launched, the user (and applications) can securely cross administrative domains without having to type passwords or passphrases multiple times. Consequently, TSI's contribution to the field of metacomputing is in showing that basic SSO functionality can be provided by layering on top of the existing SSH infrastructure, instead of replacing it with an altogether new system. In the future, it is our intention to make the TSI available as open-source software.

II. Trellis System Overview

The Trellis system is a thin layer of *software* that allows a set of jobs to be load balanced (i.e., via a scheduler [6], [8]) across multiple HPC systems while also allowing the jobs to access their data (i.e., via a distributed file system [9]). A user submits jobs to the Trellis scheduler and it automates the placement of jobs, movement of data, and collection of the results. The Trellis Project, along with many partners, performed the Canadian Internetworked Scientific Supercomputer (CISS) experiments in 2002 and 2003 [7], with 18 different administrative domains at 16 different institutions. Earlier versions of the TSI were used for CISS and it has since been improved. The lessons learned from the CISS experiments have prompted the evolutionary design changes, described here, to the underlying SSH-based security infrastructure. For example, our experience with CISS reinforces our belief that scalability concerns with SSH are primarily an implementation (not architectural) issue. Improvements have also been made to the user-level management of SSH keys and sessions. Finally, the basic functionality of TSI has been extended. For example, third-party data transfers are now possible (e.g., Agent A transfers data from Agent B to Agent C without first moving the data to A)

In contrast with other systems, Trellis is unique in its relative simplicity, cross-platform support, and track record with large-scale, on-line tests using real scientific applications. We have learned many lessons by using Trellis to build the CISS metacomputer [7]. Putting large HPC resources at the disposal of scientists makes a complete parameter sweep of a design space [10] or simulating large systems more practical.

III. Related Work

Of course, the basic idea of using a collection of HPC resources has been around for decades. In various forms, and with important distinctions, it has also been known as distributed computing, batch scheduling, cycle stealing, peer-to-peer systems, and (most recently) grid computing. Recently, there is renewed interest in metacomputing and the broader area of grid computing, fueled by the growth of high-speed wide-area networks (WAN).

Some well-known application-oriented examples in this area include SETI@home [11] and Project RC5/distributed.net [12]. But, the contemporary challenge is in supporting arbitrary applications across administrative domains. Related examples of middleware infrastructure include Condor [13] and the projects associated with the Globus Alliance and the Open Grid Service Architecture (OGSA) [1]. SETI@home and RC5 (and similar projects) are targeted at single applications (e.g., signal processing) with low resource needs (e.g., can run on a laptop), whereas Condor, Globus, and Trellis target arbitrary applications with large resource requirements. Of course, there are many other related projects around the world, including some in Canada (for example, Grid Canada [14] and the University of Victoria Grid Testbed [15]).

Of the recent systems and security infrastructure for grid computing, Globus and GSI are the best known examples. Authentication, confirming the identity of users and principals, in GSI is based on the cryptographic signing of credentials and keys by a certification authority (CA) [2]. CA-signed keys can be used to securely identify a user or a process acting on behalf of a user. Once the identity of a principal has been established, configuration files can map a global identity to a local identity and to associate privileges with the principal (i.e., authorization). Proxies are temporary identities that allow the delegation of rights and privileges. A key element of GSI is the ability to accept credentials that have been signed by different CAs. Therefore, if different administrative domains have at least one trusted CA in common, it is possible for a user to authenticate once, create a proxy or proxies as necessary, and then login and access resources across the domains. This is the SSO aspect of GSI.

GSI does have pragmatic, inter-related weaknesses, including: (1) the *a priori* need for GSI software on all the system, to be installed by systems administrators and configured according to a (human) security agreement, such as the trust

accorded to CAs, (2) relative complexity, due to the wide spectrum of potential application domains, and (3) the relative lack of widespread deployments.

Within a project like WestGrid (`www.westgrid.ca`), which has acquired and configured a wide range of HPC systems in Western Canada, it is possible to deal with Point 1 since the (human) administrative infrastructure already exists and the use of GSI was mandated *before* any system was installed. However, what about other, independent HPC consortia in Canada and what about existing projects and groups? How can nation-wide HPC systems be aggregated if they are not using GSI? In fact, many individual research groups already have large HPC resources (witness the popularity of project-specific clusters) that do not use GSI. Without the common security infrastructure, the systems cannot be aggregated. In the future, GSI may become much easier to install and configure, but what happens in the meantime and what about groups that have limited systems administrator support? Although Point 3 is related to Point 1, they are not the same; there are many systems that are both widely deployed and also require at least some systems administrator support. GSI is a technically-strong system, but there are significant social factors that affect the adoption of any new system.

As an overlay system, Trellis and the TSI are designed to run on top of on any platform, including GSI. For example, within WestGrid, GSI handles the cross-domain authentication between the WestGrid sites and the TSI handles the security between WestGrid and non-WestGrid systems (e.g., our departmental systems). The TSI is able to use GSI when it is available; it is just that there are many systems where GSI is not available and SSH is a practical alternative.

One disadvantage of the SSH-based approach is that the user (not the systems administrator) assumes the responsibility of managing the various keys, identities, and configuration. Consequently, it might be argued, the basic SSH-based architecture may not scale to large numbers of systems. We agree that this is a current (but diminishing) weakness of the Trellis approach. However, in our experience, the common case for overlay metacomputers is likely to be four or five different administrative domains. For example, the Trellis Project has access to over 1,500 processors in just four administrative domains: (1) our group's non-GSI cluster, (2) the GSI-based WestGrid facilities at the University of Alberta, (3) at the University of Calgary, and (4) at the University of British Columbia and TRIUMF. And, as CISS has shown, the Trellis system does scale, in practice.

Furthermore, the basic concepts behind SSH are well understood by, literally, hundreds of thousands of daily users. And, the growing body of experience with SSH in production environments means that bugs (like most systems, OpenSSH is not necessarily bug-free) are fixed quickly and systems administrators trust external users accessing the system over SSH (as much as they trust local users), without the need for special human security agreements. In fact, with SSH, there is no need for human-negotiated organization-to-organization security agreements; the systems administrators only have to agree to give the user a normal, unprivileged account. Nonetheless, the improvements in the TSI described here are our attempts to ameliorate the actual and perceived weaknesses of the SSH-based approach.

SSH has become more than "just a remote login facility" and can be a platform for additional security mechanisms. In fact, with GSI-enabled SSH, it is possible for Trellis to create an overlay metacomputer that includes both GSI-based and non-GSI systems. We demonstrate this capability in Section IV. Notably, the extra Trellis software exists entirely at the user-level and can be deployed by unprivileged users.

Finally, it should be noted that Plan 9's factotum [16] is similar to SSH's agent. As with SSH's `ssh-agent`s, factotums are also per-user "self-contained agents". Whereas `ssh-agent`s are (currently) only used by SSH, factotums are designed to be used directly by a variety of programs that require authentication. By layering on top of `ssh-agent`s, TSI gives other applications access to the security features of SSH, albeit via a level of indirection (i.e., applications use SSH, which in turn uses the agent).

IV. The Trellis Security Infrastructure

The Trellis Security Infrastructure (TSI) relies on the existing SSH mechanisms of public-key authentication and agents. Using a public-private pair of keys, it is possible to securely authenticate to a system. In combination with the `ssh-agent` program, it is possible to use public-key authentication **and** not require the user to type in passwords or passphrases multiple times.

The high-level strategy to provide SSO is to configure and launch `ssh-agent` processes on all the hosts such that any of the user's processes can, without human intervention or manual authentication, access any other remote host. The low-level implementation issues are related to making TSI easier to configure for the different hosts, how to launch the `ssh-agent`s, and how to check for (and fix) common connectivity problems. Specifically, TSI's SSO relies on the agent on each system being set up and configured correctly beforehand. Also, we need a way to verify that the agents are working properly to create the *SSH overlay*: the SSH connectivity between machines of an overlay metacomputer.

SSH overlays can have any shape: a fully-connected graph, a star graph (e.g., client-server), or any other graph, depending on the system setup. On top of the SSH overlay, the other components of Trellis, including the global scheduler and distributed file system, help to create the abstraction of the overlay metacomputer.

The initial configuration of the overlay, the bootstrap start-up of the overlay, and the monitoring of the overlay are all handled by helper and diagnostic programs provided by the Trellis system. In some cases, the SSH overlay will not function as expected because of connection failures. These failures are usually due to one of several common-case problems in configuring SSH and agents. For example, an agent will fail to connect to a remote host if the remote host does not have the public key of the user represented by the agent. Connection

```
dngo@st-brides hosts>createHost -i
Create Host Script - Interactive Mode Enabled
What is the name of the host you would like to create data for? st-brides.cs.ualberta.ca
What is your user name on st-brides.cs.ualberta.ca? [dngo]
What type of host is this (linux, solaris, etc.)? linux
What type of batch scheduler does this host have?
1. Zero Infrastructure
2. PBS Infrastructure
>1
<snip>
dngo@st-brides hosts>ls -l ~/.trellis/hosts
total 16
drwx--S---    2 dngo     man          4096 Jan 19 11:22 dngo@lattice.westgrid.ca/
drwx--S---    2 dngo     man          4096 Jan 19 11:21 dngo@nexus.westgrid.ca/
drwx--S---    2 dngo     man          4096 Jan 19 11:19 dngo@st-brides.cs.ualberta.ca/
drwx--S---    2 dngo     man          4096 Jan 19 11:20 pbsweb@lindale.cs.ualberta.ca/
```

Fig. 2. Adding and configuring hosts to the SSH overlay

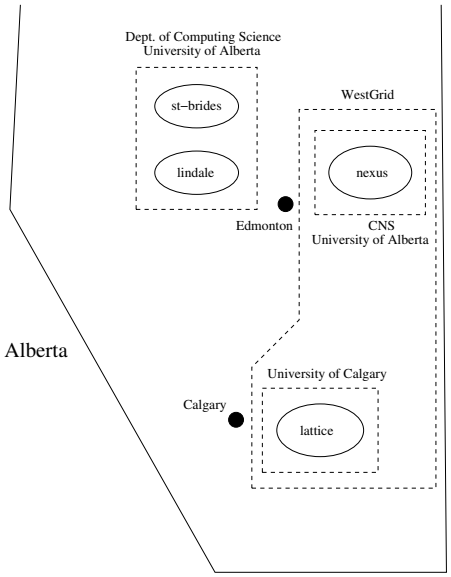

Fig. 3. Geographic view of all hosts in the example SSH overlay

failures could also simply result from the lack of an agent. Or, if an agent is present, the user's identity may not have been properly added to the agent. Most causes of connection failures can be determined and resolved with little or no user intervention. The following example sessions show how to create an overlay, start up the required processes, and diagnose common malfunctions.

A. Example: Configuring, Launching, and Testing the SSH Overlay

All of the tools shown in this section have been implemented and the interactive sessions are demonstrated using the tools themselves.

Creating and maintaining an SSH overlay has three basic steps. First, we specify and configure all the hosts that we want as part of the overlay. We use the Trellis-provided `createHost` tool to interactively add an entry for each host, as shown in Figure 2. In response to a short list of configuration questions about the host and the type of scheduler to be used, a new host entry is created. A host entry is simply a directory named as `user@host` in the user's standard `~/.trellis/hosts` directory (similar to the `~/.gnome` directory for configuring the popular GNOME desktop). Note that the user's identity on the different hosts can be different (i.e., `dngo` versus `pbsweb`). Once a host has been configured, it does not need to be re-configured (not shown) between sessions, unless there is a change.

In this example, we have created four hosts in three different administrative domains: (1) `st-brides` and `lindale` are in the Department of Computing Science, University of Alberta, (2) `nexus` is located within Computing and Network Services (CNS), the University of Alberta, and (3) `lattice` is located at the University of Calgary. Figure 3 illustrates a geographic view of these four hosts. Since both `nexus` and `lattice` are both part of the WestGrid GSI-based grid, we consider them to have some sort of administrative relationship. Specifically, Trellis uses regular OpenSSH to cross between (1)–(2) and (1)–(3), but Trellis uses GSI-enabled SSH to cross between (2)–(3).

Second, we need to launch an SSH agent and leave it running on each of the hosts. Recall that the well-known `ssh-agent` process is the existing SSH mechanism to allow a process to authenticate, via a public-private key, to a (possibly) remote host without requiring a password or passphrase [4]. Whenever a Trellis process, whether interactive or background, needs to make a remote connection, it uses a configuration file to find the per-host `ssh-agent`, set the appropriate environment variables (e.g., `SSH_AGENT_PID` and `SSH_AUTH_SOCK`), and authenticate without human intervention.

We have created two OpenSSH-based tools, `ssh-agent-remote` and `ssh-add-remote`, to help launch and load the remote agents with the appropriate keys, respectively. Both tools are wrappers around their original, non-`remote` counterparts and allow for the set-up and control of an `ssh-agent` on a remote host. Figure 4 shows how an `ssh-agent` is started on `nexus` from `st-brides`. Note that the `ssh-agent` on `nexus` is given a private key (from `~/.ssh/keyfile` on `st-brides`) without that private key ever being saved on `nexus`; we have a modified

```
dngo@st-brides hosts>ssh-agent-remote nexus.westgrid.ca
Agent for dngo@nexus.westgrid.ca started (PID 881635)
dngo@st-brides hosts>ssh-add-remote nexus.westgrid.ca ~/.ssh/keyfile
Enter passphrase for /usr/brule2/guest/dngo/.ssh/keyfile:
Identity added: Using Stdin (Using Stdin)
```

Fig. 4. Starting an SSH agent on `nexus` from `st-brides`

```
----------------------------------------------------------------
Connectivity Report

Created: Tue Jan 20 15:15:21 2004 on st-brides.cs.ualberta.ca
----------------------------------------------------------------
Hosts found on st-brides.cs.ualberta.ca:
dngo@nexus.westgrid.ca
dngo@lattice.westgrid.ca
dngo@st-brides.cs.ualberta.ca
pbsweb@lindale.cs.ualberta.ca

Legend:
<-->  Bidirectional connectivity
--->  Unidirectional connectivity
-X->  No connectivity from specific host

Depth:
-------------------------------------------------------------------------------------------------------------------------
        0                       1                       2                       3                       4
-------------------------------------------------------------------------------------------------------------------------
dngo@st-brides.cs  --->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  dngo@nexus.westgr  --->  dngo@nexus.westgr
dngo@st-brides.cs  <-->  dngo@lattice.west  --->  dngo@lattice.west
dngo@st-brides.cs  <-->  pbsweb@lindale.cs  --->  pbsweb@lindale.cs
dngo@st-brides.cs  <-->  dngo@nexus.westgr  <-->  pbsweb@lindale.cs  <-->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  dngo@nexus.westgr  <-->  dngo@lattice.west  <-->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  dngo@lattice.west  <-->  dngo@nexus.westgr  <-->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  dngo@lattice.west  <-->  pbsweb@lindale.cs  <-->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  pbsweb@lindale.cs  <-->  dngo@nexus.westgr  <-->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  pbsweb@lindale.cs  <-->  dngo@lattice.west  <-->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  dngo@nexus.westgr  <-->  pbsweb@lindale.cs  <-->  dngo@lattice.west  <-->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  dngo@nexus.westgr  <-->  pbsweb@lindale.cs  <-->  dngo@lattice.west  <-->  dngo@nexus.westgr
dngo@st-brides.cs  <-->  dngo@nexus.westgr  <-->  dngo@lattice.west  <-->  pbsweb@lindale.cs  <-->  dngo@nexus.westgr
dngo@st-brides.cs  <-->  dngo@nexus.westgr  <-->  dngo@lattice.west  <-->  pbsweb@lindale.cs  <-->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  dngo@lattice.west  <-->  dngo@nexus.westgr  <-->  pbsweb@lindale.cs  <-->  dngo@lattice.west
dngo@st-brides.cs  <-->  dngo@lattice.west  <-->  dngo@nexus.westgr  <-->  pbsweb@lindale.cs  <-->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  dngo@lattice.west  <-->  pbsweb@lindale.cs  <-->  dngo@nexus.westgr  <-->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  dngo@lattice.west  <-->  pbsweb@lindale.cs  <-->  dngo@nexus.westgr  <-->  dngo@lattice.west
dngo@st-brides.cs  <-->  pbsweb@lindale.cs  <-->  dngo@nexus.westgr  <-->  dngo@lattice.west  <-->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  pbsweb@lindale.cs  <-->  dngo@nexus.westgr  <-->  dngo@lattice.west  <-->  pbsweb@lindale.cs
dngo@st-brides.cs  <-->  pbsweb@lindale.cs  <-->  dngo@lattice.west  <-->  dngo@nexus.westgr  <-->  dngo@st-brides.cs
dngo@st-brides.cs  <-->  pbsweb@lindale.cs  <-->  dngo@lattice.west  <-->  dngo@nexus.westgr  <-->  pbsweb@lindale.cs
```

Fig. 5. Connectivity Report: Checking an SSH Overlay

version of `ssh-add` that reads a private key from standard in (i.e., `stdin`) instead of from a file. Since the `stdin` of `ssh-add` on `nexus` is connected to the `ssh` channel from `st-brides`, this means that the private key is always encrypted until it arrives at the `ssh-agent` process on `nexus`, thus maintaining security.

Another tool, `launchAgents` (not shown), invokes `ssh-agent-remote` and `ssh-add-remote` for all the hosts in `~/.trellis/hosts` and loads the remote `ssh-agent` processes with a common key. Therefore, with one tool and after typing in only one passphrase, a user can launch the whole SSH overlay.

Third, we can run the Report tool to generate a connectivity report (Figure 5) for the SSH overlay, as shown in Figure 6. Starting from the local host (the root node, `st-brides` in our example), it establishes whether or not it can connect to all the remote hosts that are specified in `~/.trellis/hosts`. Each of the remote hosts, in turn, is asked to perform the same connectivity test using with their own `~/.trellis/hosts` (which may be different from other `~/.trellis/hosts` configurations, since there may be different local file systems).

Each line of output represents a path from the local host to one or more remote hosts, and the type of connectivity that exists between two hosts in the path. For instance, the third line indicates that we can connect from `st-brides` to `lattice` and vice versa, and also from `lattice` back to itself. The connectivity report is a map of the exact SSH overlay per the *static* information in `~/.trellis/hosts` and the *dynamic* state of various `ssh-agent` processes on the hosts. The connectivity between two hosts A and B is classified as bidirectional if host A can connect to host B and vice versa, or unidirectional otherwise. All connection failures (none exist

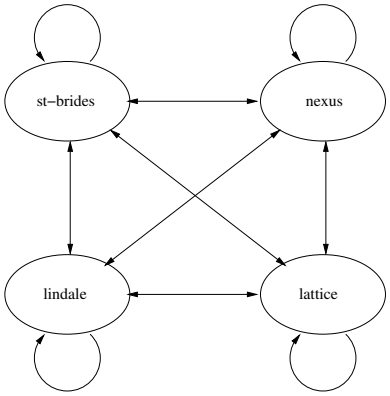

Fig. 6. Directed Graph of the SSH overlay

or are shown in the example) are also clearly marked on the SSH overlay.

Note that Figure 5 and Figure 6 represent a fully-connected graph with all the links working as desired. Once the user authenticates to, say, `st-brides` it is possible to remotely access any of the other systems in the overlay, either interactively or from a background process of the Trellis scheduler or file system. In this way, TSI provides the much-desired SSO functionality. Security is maintained through the fact that authentication still takes place between hosts; it is just that the TSI has created a system where the authentication is done automatically and transparently. Note that any private keys are, as with the original SSH system, either kept in `ssh-agent`s or on disk with a passphrase; the system is no less secure than the original SSH. Also, only the user that created and launched the SSH overlay has the SSO functionality since the host configurations and the `ssh-agent`s are normal files or user-level processes with the same identity as the user. Of course, different users can create their own per-user SSH overlays (and, thus, their own overlay metacomputers) and obtain SSO functionality.

B. Example: Diagnosing and Fixing Problems in the SSH Overlay

Naturally, the SSO abstraction is maintained as long as everything is working. But, if a failure is detected, a Diagnosis tool can be used to perform several diagnostics on the connection failures. The tests attempt to ascertain the cause of the failure and whether it lies with host A or host B. Typically, the cause of the failure is likely to be one of a small number of possibilities. For example, the `ssh-agent` process on a host A may have disappeared. Or, the `~/.trellis/hosts` entry on host A may have been misconfigured with the wrong user identity for host B. Or, the `ssh-agent` process on host A may not have been loaded with the proper key. Or, the `authorized_keys2` file on host B may be missing a required public key. By checking a list of common problems, the Diagnosis tool attempts to find the cause of a connectivity failure to restore SSO.

If the problems are diagnosed and are fixable, the Fix tool tries to resolve the connection failures iteratively. It takes the corrective action for one connection failure, and optionally checks the connectivity before repeating the process.

To further demonstrate the functionality of the connectivity tools, consider the four hosts in the artificially-created states as shown in Figure 7. `st-brides`, which has an arc to itself, has a proper agent and its `authorized_keys2` file contains the correct public key. `lindale` has a proper `authorized_keys2` file to allow an incoming connection from `st-brides`, but it lacks an agent so it cannot make any outgoing connections. `lattice` has a proper `authorized_keys2` file to allow an incoming connection from `st-brides` and there is a local agent, but the agent is missing the proper keys that allows for outgoing connections. `nexus` has an agent but its `authorized_keys2` file does not have the proper public key, which prevents incoming connections from the other hosts.

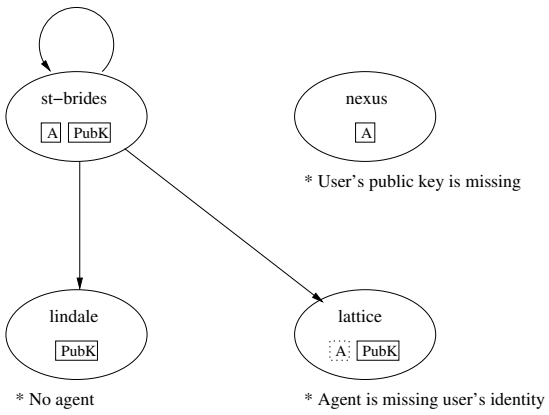

Fig. 7. Example: Causes of Connectivity Failures

Figure 8 indicates the initial SSH overlay that we obtain from the Report tool. Because the user's public key is recognized by `lattice` and `lindale`, we have no trouble in making connections. Their lack of working agents, for outgoing connections, results in the unidirectional connectivity. On the other hand, `st-brides` and `nexus` have no connectivity at all since public-key authentication fails on `nexus` as expected.

In order to determine these problems, the initial SSH overlay is given to the Diagnosis tool. As Figure 9 shows, it performs tests that check for problems on the client and server sides of a connection. In the second connection failure from `lattice` back to `st-brides`, for example, the tool first pings `st-brides` to verify its existence. Next, it checks `lattice` and discovers that the user's identity is missing from the agent. `st-brides` is also checked, but no problems are found.

The information from the diagnostics is in turn passed onto the Fix tool. In an interactive session, the tool presents the user with a list of the connection failures and the most likely causes, as shown in Figure 10. For the first connection failure, the tool needs to determine the exact problem beforehand in order to enable connectivity between `st-brides` and `nexus`. The Diagnosis tool is able to check `st-brides` and finds no client problems. However, it is not able to check `nexus` for server problems since it cannot find a path to reach the host. As a result, it tries to make a connection with the user's help, and is then able to determine the source of the connection failure and correct it. Appropriate action is also taken for the other two failures by setting up a working agent. Specifically, the agent on `lattice` is loaded with the user's identity, and a new agent is started on `lindale` and given the user's identity as well.

We then proceed to retest the connectivity between the hosts. The final SSH overlay is the same as the one represented

```
---------------------------------------------------------------
Connectivity Report

Created: Wed Jan 21 15:58:37 2004 on st-brides.cs.ualberta.ca
---------------------------------------------------------------
Hosts found on st-brides.cs.ualberta.ca:
dngo@lattice.westgrid.ca
dngo@nexus.westgrid.ca
dngo@st-brides.cs.ualberta.ca
pbsweb@lindale.cs.ualberta.ca

Legend:
<-->  Bidirectional connectivity
--->  Unidirectional connectivity
-X->  No connectivity from specific host

Depth:
-------------------------------------------------
        0                       1
-------------------------------------------------
dngo@st-brides.cs  --->  dngo@st-brides.cs
dngo@st-brides.cs  -X->  dngo@nexus.westgr
dngo@st-brides.cs  --->  dngo@lattice.west
dngo@st-brides.cs  --->  pbsweb@lindale.cs
```

Fig. 8. Connectivity Failure (Initial State) as Reported by the Report Tool

```
[FAILURE: dngo@st-brides.cs.ualberta.ca TO dngo@nexus.westgrid.ca]
Checking if dngo@nexus.westgrid.ca exists
  dngo@nexus.westgrid.ca exists.
Checking dngo@st-brides.cs.ualberta.ca
  Checking for a SSH agent
    Agent found.
  Checking if user's identity has been added to the agent
Checking dngo@nexus.westgrid.ca
  No path to dngo@nexus.westgrid.ca

[FAILURE: dngo@lattice.westgrid.ca TO dngo@st-brides.cs.ualberta.ca]
Checking if dngo@st-brides.cs.ualberta.ca exists
  dngo@st-brides.cs.ualberta.ca exists.
Checking dngo@lattice.westgrid.ca
  Checking for a SSH agent
    Agent found.
  Checking if user's identity has been added to the agent
    Identity not found in the agent.
Checking dngo@st-brides.cs.ualberta.ca
  Checking for sshd
    sshd found.
  Checking 'authorized_keys' for user's public key
    Public key found.

[FAILURE: pbsweb@lindale.cs.ualberta.ca TO dngo@st-brides.cs.ualberta.ca]
Checking if dngo@st-brides.cs.ualberta.ca exists
  dngo@st-brides.cs.ualberta.ca exists.
Checking pbsweb@lindale.cs.ualberta.ca
  Checking for a SSH agent
    No agent found.
Checking dngo@st-brides.cs.ualberta.ca
  Checking for sshd
    sshd found.
  Checking 'authorized_keys' for user's public key
    Public key found.
```

Fig. 9. Tests done by the Diagnosis tool

in Figure 5 and Figure 6, namely a fully-connected graph of the hosts.

C. Empirical Results: Basic Overheads

So far, we have not measured the real-time overheads of using the configuration, launch, diagnosis, and fix tools; those overheads depend on too many user interactions and transient network characteristics. As one would expect, since they involve a variety of SSH connections across a WAN, there is a noticeable overhead. However, since those tools are invoked infrequently, we have not attempted any significant optimizations. We expect the common case to be *using* the SSO capabilities of the overlay.

The more frequent use-case for TSI is in establishing SSH connections in support of Trellis's placeholder scheduling and distributed file system. Therefore, to stress-test the system we can do a *simple* microbenchmark. In our experiment, we compare two versions of OpenSSH: (1) the unmodified version

```
Listed below are the connection failures found and their most likely causes.
Connections are from host A to host B.

    ------------------------------  ------------------------------------------
           Connection Failure                          Reason
    ------------------------------  ------------------------------------------
    Host A          Host B
 1) dngo@st-brides. dngo@nexus.west  No path to host B
 2) dngo@lattice.we dngo@st-brides.  Private key not added to SSH agent
 3) pbsweb@lindale. dngo@st-brides.  No SSH agent running on host A

Select a connection failure to fix (q to quit): 1
  Attempting a manual connection to dngo@nexus.westgrid.ca
  Public key not found in 'authorized_keys' on dngo@nexus.westgrid.ca
  Attempting to fix.  Please enter password when required.
  Enter public key file: [/usr/brule2/guest/dngo/.trellis/keyfile.pub]
  Copying public key to dngo@nexus.westgrid.ca
dngo@nexus.westgrid.ca's password:
  Adding public key to 'authorized_keys'
dngo@nexus.westgrid.ca's password:
  Fix successful.
Retest connectivity? (y/n) [y] n

Select a connection failure to fix (q to quit): 2
  Enter private key file: [/usr/brule2/guest/dngo/.ssh/keyfile]
  Adding the specified private key to the agent
  Enter passphrase for /usr/brule2/guest/dngo/.ssh/keyfile:
  Fix successful.
Retest connectivity? (y/n) [y] n

Select a connection failure to fix (q to quit): 3
  Starting an agent on pbsweb@lindale.cs.ualberta.ca
  Enter private key file: [/usr/brule2/guest/dngo/.ssh/keyfile]
  Adding the specified private key to the agent
  Enter passphrase for /usr/brule2/guest/dngo/.ssh/keyfile:
  Fix successful.
Retest connectivity? (y/n) [y] y
Retesting connectivity...

<snip>

Listed below are the connection failures found and their most likely causes.
Connections are from host A to host B.

    ------------------------------  ------------------------------------------
           Connection Failure                          Reason
    ------------------------------  ------------------------------------------
    Host A          Host B
    <No connection failures>

Select a connection failure to fix (q to quit): q
```

Fig. 10. An interactive session with the Fix tool

that comes with the OpenSSH distribution (version 3.6p1), which uses public-key authentication, and (2) the GSI-enabled version of OpenSSH (version 3.6.1p2). Specifically, we measure the amount of time it takes to make 100 connections between `nexus` and `lattice`. Both of these hosts are on the WestGrid network. Using GSI-enabled SSH, 100 connections to run the `date` command takes 100 seconds. Using the unmodified SSH, the same 100 connections takes 120 seconds, which represents a 20% additional overhead for the TSI approach. Of course, both 1.0 and 1.2 seconds of overhead per connection is somewhat high compared to connections on a local network, but cross-domain authentication across a WAN is typically more expensive.

We suspect that TSI's additional overheads, which are not onerous, may be due to the more-complicated baseline SSH protocol for authenticating the host (i.e., `nexus` authenticating `lattice` and vice versa, using public keys). Fortunately, we have also developed a simple SSH proxy that creates per-host SSH connections on demand and leaves the connection open until it times out due to inactivity. The same 100 connections experiment took 9 seconds, again running the `date` command, between `nexus` and `lattice` using the unmodified OpenSSH binary, with our proxy. Therefore, even the 20% overhead can be avoided in the common case of Trellis's operation.

V. Concluding Remarks

The interest in aggregating HPC resources across a WAN has been growing as the capacity needs of computational scientists have increased. Cross-domain tools and security infrastructure are an important part of metacomputing and grid computing. The Grid Security Infrastructure, or GSI, is a sophisticated and elegant system. However, GSI requires a substantial amount of installation support and it requires human-negotiated organization-to-organization security agreements.

We have proposed an alternate security infrastructure based on the widely-deployed SSH, called the Trellis Security Infrastructure, or TSI. By layering TSI on top of SSH, it is possible to deploy an SSH overlay and an overlay metacomputer entirely at the user-level, without compromising security. SSH is a well-known and trusted system with public-key authentication and secure channels. The only human-negotiated agreement required for a TSI-based overlay metacomputer is the user's unprivileged account on the various hosts. In contrast to the GSI approach, the contribution of the TSI approach is in demonstrating that a single sign-on system can be implemented without requiring a completely new security infrastructure.

ACKNOWLEDGMENTS

Thank you to the Alberta Science and Research Authority (ASRA), SGI, C3.ca Association Inc, Alberta's Informatics Circle of Research Excellence (iCORE), Sun Microsystems, Inc., the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Canada Foundation for Innovation (CFI) for their research support.

We also gratefully acknowledge the contributions of Jonathan Schaeffer, Chris Pinchak, Paul Masiar, Rob Lake, Aaron Davidson, George Ma, Victor Salamon, Jeff Siegel, Jeremy Handcock, Vanessa Chung, and Yaling Pei to the Trellis Project.

REFERENCES

[1] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," 2002, Open Grid Service Infrastructure WG, Global Grid Forum, http://www.globus.org/.

[2] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch, "A National-Scale Authentication Infrastructure," *IEEE Computer*, vol. 33, no. 12, pp. 60–66, 2000.

[3] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, "Security for Grid Services," in *Proc. 12th Int'l Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.

[4] D. Barrett and R. Silverman, *SSH, the Secure Shell: The Definitive Guide*. O'Reilly and Associates, 2001.

[5] Trellis Project, [Online 2004], Available: http://www.cs.ualberta.ca/~paullu/Trellis/.

[6] C. Pinchak, P. Lu, and M. Goldenberg, "Practical Heterogeneous Placeholder Scheduling in Overlay Metacomputers: Early Experiences," in *Proc. 8th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, Edinburgh, Scotland, UK, July 24, 2002, pp. 85–105.

[7] C. Pinchak, P. Lu, J. Schaeffer, and M. Goldenberg, "The Canadian Internetworked Scientific Supercomputer," in *17th Annual Int'l Symposium on High Performance Computing Systems and Applications (HPCS)*, Sherbrooke, Quebec, Canada, May 11–14, 2003.

[8] M. Goldenberg, P. Lu, and J. Schaeffer, "TrellisDAG: A System for Structured DAG Scheduling," in *9th Workshop on Job Scheduling Strategies for Parallel Processing*, Seattle, June 2003, pp. 21–35.

[9] J. Siegel and P. Lu, "User-Level Remote Data Access in Overlay Metacomputers," in *Proceedings of the 4th IEEE Int'l Conference on Cluster Computing*, Sept. 2002, pp. 480–483.

[10] Y. Xu, A. Huckauf, W. Jäger, P. Lu, J. Schaeffer, and C. Pinchak, "The CISS-1 Experiment: *ab initio* Study of Chiral Interactions," in *39th Int'l Union of Pure and Applied Chemistry (IUPAC) Congress and 86th Conference of The Canadian Society for Chemistry*, Ottawa, Ontario, Canada, August 10–15, 2003.

[11] SETI@home, [Online 2004], Available: http://setiathome.ssl.berkeley.edu/.

[12] RC5 Project, [Online 2004], Available: http://www.distributed.net/rc5/.

[13] Condor, [Online 2004], Available: http://www.cs.wisc.edu/condor/.

[14] Grid Canada, [Online 2004], Available: http://www.gridcanada.ca/.

[15] UVic Grid Testbed, [Online 2004], Available: http://grid.phys.uvic.ca/.

[16] R. Cox, E. Grosse, R. Pike, D. Presotto, and S. Quinlan, "Security in Plan 9," in *Proc. 11th USENIX Security Symposium*, San Francisco, California, U.S.A., August 5–9, 2002, pp. 3–16.