PRE-PRINT: The Trellis Security Infrastructure for Overlay Metacomputers and Bridged Distributed File Systems

Paul Lu, Michael Closson, Cam Macdonell, Paul Nalos, Danny Ngo, Morgan Kan, and Mark Lee

Department of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8, Canada
Email: paullu@cs.ualberta.ca

Abstract

Researchers often have non-privileged access to a variety of high-performance computer (HPC) systems in different administrative domains, possibly across a wide-area network. [1] Consequently, the security infrastructure becomes an important component of an *overlay metacomputer*: a user-level aggregation of HPC systems.

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 TSI is in demonstrating that a single sign-on (SSO) system, for a variety of use-case scenarios, can be implemented without requiring a completely new security infrastructure. We describe the use of TSI for a Canada-wide overlay metacomputer, for computational workloads (i.e., CISS-3) that spanned 22 administrative domains, at its peak had over 4,000 concurrent jobs, and included a new distributed file system (i.e., Trellis NFS).

Key words: security, single sign-on, metacomputing, computational science, capacity computing, global job scheduler, distributed file system

[1] Please see `http://dx.doi.org/10.1016/j.jpdc.2006.04.005` for the final, corrected, camera-ready version. This version of the paper is provided for reference only.

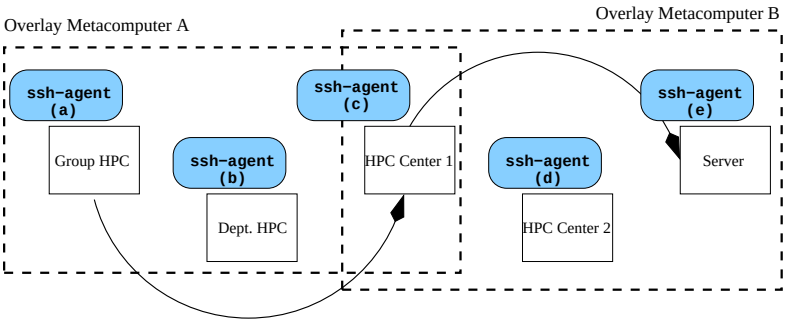

Fig. 1. Overlay Metacomputers and the Trellis Security Infrastructure

1 Introduction

Some workloads and experiments in computational science require large amounts of resources, both in terms of capability and capacity. 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.

For example, Researcher A (Figure 1) 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 may 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.

Some of the main requirements of the security infrastructure for a cross-administrative domain situation include:

(1) *Single Sign-On (SSO) across Multiple Administrative Domains:* The user wishes to authenticate (i.e., prove his identity) to the system only once, and not once per-domain. The well-known Secure Shell (SSH) [1] system can support SSO if the user properly sets up his private and public keys and uses the `ssh-agent` for automated authentication.
(2) *SSO Support for Background Jobs, Servers, and Multiple Users:* Jobs or servers left in the background need SSO (e.g., to get a unit of work, return a result, move data).
(3) *Security and Mitigation of Attacks:* Secure Shell is already considered to be reasonably secure. The challenge for this work is in maintaining that security while not opening up new, significant avenues of attack.

At a high level, the Trellis Security Infrastructure (TSI) addresses some of the

main issues in security as follows:

(1) *Basic Authentication and Authorization:* TSI relies on the existing ability to use `ssh-agent` for automatic, non-interactive *authentication*. The problems are: How can *all* processes find a valid agent? How can a server support different users and their per-user agents?

    Layered on top of Secure Shell, TSI makes it possible and convenient to create a set of `ssh-agent`s (Figure 1), such that for any node that is the source of an *outgoing* `ssh` connection (e.g., `Group HPC` and `HPC Center 1`), there will be an agent available at a *well-known location* (i.e., indirectly find the agent's Unix domain socket and process identity) (Figure 1, agents `(a)` for Researcher A and `(c)` for Researcher B, respectively).

    At the *incoming* end of the connection (e.g., Figure 1, `HPC Center 1` and `Server`), the user (or a TSI tool) must have the public keys properly set up in the `authorized_keys` configuration file of Secure Shell. The *forced command* feature of the OpenSSH implementation of SSH (i.e., the current de facto implementation) supports *authorization* and is useful to mitigate certain classes of attacks.

(2) *Authentication and Delegation of Actions for Servers and Multiple Users:* As with, for example, the Trellis NFS distributed file system [2] (see also Section 4) it is also possible for the server to access different agents, belonging to different users, such that the user can *delegate* an action (e.g., data movement) to the server.

(3) *Key Management: Creation, Dissemination, Loading:* With public-key authentication systems, such as SSH and TSI, public and private key pairs must be created. Neither SSH nor (currently) TSI *enforces* good key creation methodology, but TSI makes it convenient to follow "best practices".

Understandably, systems administrators desire modern and effective cross-domain mechanisms for authentication, authorization, and data management. For example, grid computing and the Globus Project [3] 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) [4, 5] 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. However, for cross-domain situations (e.g., virtual organizations) GSI requires a substantial amount of privilege to install, configure, and it requires human-negotiated organization-to-organization security agreements at the administrator-level (and up) to decide on the trusted certification authorities (CA) and how to map global user identities to local user identities.

But, 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 [1]. SSH supports public-key authentication, secure channels using strong encryption, and can use familiar local protection mechanisms for data sharing and other aspects of authorization. Consequently, the Trellis Project 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 thousands of computers in many administrative domains (Section 4). Working 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 with an earlier paper on the Trellis Security Infrastructure (TSI) [8], we do *not* claim that TSI is a general-purpose security infrastructure. Explicitly, TSI is designed to support capacity-oriented HPC workloads and their related needs, such as the need for distributed file systems.

2 Trellis System Overview and Background

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, 9]) across multiple HPC systems while also allowing the jobs to access their data (i.e., via a distributed file system [2,10,11]). 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 first two Canadian Internetworked Scientific Supercomputer (CISS) experiments in 2002 and 2003 [7], with 18 different administrative domains at 16 different institutions. Since the publication of the original TSI paper [8], the security infrastructure has evolved, the CISS-3 experiment of 2004 has been completed, and we include extensive discussion of TSI and that experiment in Section 4.

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. Some well-known application-oriented examples in this area include SETI@home and Project RC5/distributed.net. But, the contemporary challenge is in supporting arbitrary applications across administrative domains. Related examples of middleware infrastructure include Condor [12], and the projects associated with the Globus Alliance and the Open Grid Service Architecture (OGSA) [3]. 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 lap-

top), 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 and the University of Victoria Grid Testbed).

Of the recent systems and security infrastructure for grid computing, Globus and GSI are the best known examples. Authentication in GSI is based on the cryptographic signing of credentials and keys by a CA [4]. CA-signed keys can be used to securely identify a user or a process acting on behalf of a user. 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.

However, GSI does have pragmatic, inter-related weaknesses, including: (1) the *a priori* need for GSI software on all the systems, 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. GSI is a technically strong system, but there are significant social factors that affect the adoption of *any* new system.

As overlay systems, Trellis and the TSI are designed to run on top of any platform, including GSI itself. 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. The burden on the user is roughly comparable to the current need for users to manage their different identities for different Web sites, and also solvable with similar tools and techniques. Consequently, it might be argued, the basic SSH-based architecture may not scale to large numbers of systems. But, with the right tools and "best practices", as provided by TSI, we have shown that it is possible to scale up to thousands of processors and tens of domains. For truly large user bases (e.g., multi-thousands of users) with regular and dynamic resource sharing, the more comprehensive approach of GSI may be better, despite the initial complexity curve. However, we believe that there will be an on-going need for smaller overlay metacomputers, which is the niche targeted by TSI.

3 The Trellis Security Infrastructure

The Trellis Security Infrastructure (TSI) relies on the existing SSH mechanisms of public-key authentication and agents. 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.

```
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

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 (Figure 1).

Configuring, Launching, and Testing the SSH Overlay: 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 this example, we have created four hosts in multiple administrative domains. 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.

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 [1]. 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. Fig-

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

ure 3 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 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.

The common use-case scenario is to set up the overlay, launch all the agents using `launchAgents`, and then to begin using the SSO and other capabilities of TSI for a computational workload.

Empirical Results: Basic Overheads: The common 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: (1) TSI with the unmodified SSH implementation from OpenSSH (version 3.6p1), which uses public-key authentication, and (2) GSI via 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` on the WestGrid network. Using GSI-enabled SSH, 100 connections to run the `date` command takes 100 seconds. Using TSI and 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 wide-area network (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. Independently of TSI, version 4 of OpenSSH has newly added support for persistent SSH connections.

4 TSI In Practice

CISS Before TSI: As discussed above, TSI is targeted at HPC applications. In a series of on-line, Canada-wide, and production-oriented experiments, the software and techniques developed by the Trellis Project have been used to complete large (i.e., multiple CPU-year, thousands of jobs) computational science workloads. Using well-known scientific applications such as MOLPRO [13], GROMACS [14], and CHARMM [15], these experiments have been dubbed the Canadian Internetworked Scientific Supercomputer (CISS) experiments. Although the CISS-1, CISS-2, and CISS-3 experiments have been summarized previously from the point of view of throughput, scale, and the file system mechanisms [2, 7], this is the first discussion of the security infrastructure and TSI in practice. For both CISS-1 and CISS-2, in 2002 and 2003, cross-domain security was handled by SSH, but many functions were handled manually. Key management was done manually and, in some cases, private keys were placed on remote nodes without passphrase protection for reasons of expediency. It was clear, even at that time, that a better solution was required.

TSI, CISS-3, and the Trellis Metascheduler: The TSI described in this paper was developed between the CISS-2 and CISS-3 experiments, in direct response to the limitations discussed above and the desire to support SSO and distributed file systems. The support for multiple users was added after CISS-3; although CISS-3 involved many administrative domains and two major workloads, there was only a single user identity in the domains that Trellis NFS was deployed.

In broad terms, CISS-3 demonstrated the functionality and scalability of the Trellis Project designs, including TSI, the metascheduler, and the distributed file system. From September 15 to 17, 2004, in a 48-hour production run, over 15 CPU-years of computation were completed, using over 4,100 processors, from 19 universities and institutions, representing 22 administrative domains [2]. At its peak, over 4,000 jobs were running concurrently.

It is beyond the scope of this paper to detail the design and implementation of the metascheduler itself. The basic architecture and an earlier implementation were covered elsewhere [6]. However, we sketch out the basic operations of the

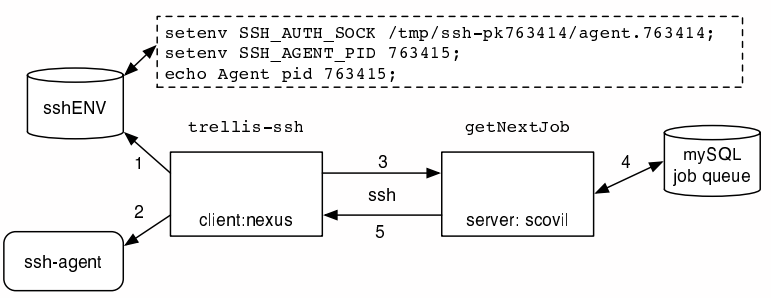

Fig. 4. Example of Metascheduler and TSI (Job Queue is on `scovil`)

metascheduler in terms of how it interacts with TSI.

The basic idea of the Trellis metascheduler is that there is a background process, called a *placeholder*, that is running on computational nodes (Figure 4, `client:nexus`). The placeholder is a special job script submitted to a batch scheduler (e.g., PBS, LoadLeveler, LSF) that, when it starts executing, queries the global job queue (e.g., `server:scovil`) and *pulls* work to its nodes. Initially, the placeholder is started using TSI and Trellis tools.

In terms of TSI, the placeholder on `nexus` wishes to perform a `trellis-ssh user@scovil getNextJob ...`, which will return the actual job that the placeholder should run. `getNextJob` retrieves the job to bind to the placeholder via a job queue database and returns the work specification via the Unix Standard Out portion of the SSH connection. `trellis-ssh` is primarily a wrapper for the real `ssh` executable. The placeholder needs to find the previously initialized `ssh-agent` to be used to authenticate as `user` on the server `scovil`. Specifically, `trellis-ssh` sources (i.e., reads) the file on `nexus` (i.e., the source of the SSH connection) at `~/.trellis/hosts/user@scovil/sshENV` (Figure 4, Step 1). The `sshENV` file contains the appropriate `SSH_AUTH_SOCK` and `SSH_AGENT_PID` environment variables needed to find the agent itself (Step 2). Previously, TSI tools (e.g., `launchAgents`) had been used to launch the agents and to set up the contents of `sshENV`. Once the appropriate agent has been found, the actual `ssh` command is executed (Step 3), the authentication succeeds thanks to the `ssh-agent` on `nexus`, and the `getNextJob` command is executed on `scovil` (Step 4). The job specification is returned over the same `ssh` connection (Step 5) and the placeholder begins the job.

TSI, CISS-3, and Trellis NFS: If a job can be scheduled on a node that is different than where its data is stored, then the job will need a method to access its input and output data. In HPC, stage-in/stage-out (i.e., explicit specification of the files required) is the de facto technique. However, with the proper security mechanisms, a distributed file system would be highly desirable.

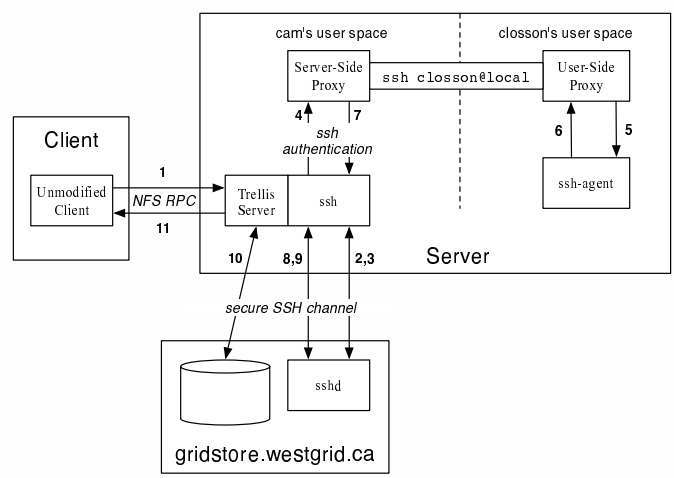

Fig. 5. Trellis NFS Supporting Multiple Users with TSI

Trellis NFS [11] is a new distributed file system based on the notion of bridging unmodified clients (and local area network (LAN) protocols) and unmodified storage nodes (and WAN protocols) [2]. The key benefit of bridging is that the LAN clients do not have to be modified, except for the creation of mount points. On the LAN, the NFS security model (for better or worse) is used. On the WAN, TSI is the security system. For CISS-3, Trellis NFS was deployed across a subset of the CISS-3 nodes totaling four administrative domains, to support two different clusters, two storage nodes, hundreds of jobs, and dozens of simultaneous clients. Basically, Trellis NFS works as follows: unmodified applications use a Secure Copy-like naming convention (e.g., `/tfs/scp:gridstore.westgrid.ca:/data/file1`) to name their files. Using the unmodified NFS RPC protocol, the client contacts the Trellis NFS server. The Trellis NFS server then uses the name of the file to determine which remote, home node (i.e., `gridstore.westgrid.ca`) to access using Secure Copy (i.e., using `trellis-scp`, the TSI wrapper script for `scp`) and copies the file to a local disk-based cache. Subsequently, Trellis NFS services all LAN NFS RPCs out of the cache. The cache is flushed at an appropriate time.

Most relevant to this discussion, Trellis NFS uses TSI. For Trellis NFS to successfully invoke `trellis-scp` to copy the files from the home nodes to the local cache, Trellis NFS needs access to a valid agent. The TSI solution to the problem is (Figure 5): Each user (e.g., `closson`) that wishes to use Trellis NFS must have a valid `ssh-agent` running on the same node as the Trellis NFS server. That agent is expected to have been created using TSI. Each user is expected to trust the Trellis NFS server enough to put the public key of the server in his `authorized_keys` file, which allows the server to (in loopback fashion to `local`) SSH into the server node as the user and access his `ssh-agent`.

When a process belonging to `closson` requires a file to be transferred from

`gridstore.westgrid.ca`, the NFS RPC goes to the server (Figure 5, Step 1). Before the server, running as user `cam`, can invoke `trellis-scp`, the server sets up a server-side proxy since `cam` is not likely to have a valid `ssh-agent` for accessing `closson@gridstore.westgrid.ca`. But, `cam`'s server-side proxy can SSH into `closson@local`, as per the discussion above. In fact, the server-side proxy starts up a user-side proxy, which finds the appropriate `ssh-agent` using the well-known TSI location at ~`closson/.trellis/hosts/closson@gridstore.westgrid.ca`.

When the server invokes `trellis-scp` (Step 2), the SSH daemon (`sshd`) at `gridstore` will start the public-key authentication process (Step 3). All of the challenge-response messages are passed verbatim to the server-side proxy (Step 4), then to the client-side proxy, and finally to `closson`'s agent (Step 5). As far as the `sshd` and `closson`'s agent are concerned, they are speaking directly to each other (Steps 6, 7, 8). And, assuming that `closson`'s agent has been loaded with the right key, the authentication at `gridstore` will succeed, then the file can be moved (Steps 9 and 10) and stored on the local disk of the server (not shown). Note that the private keys of one user are never revealed to other users. With the server-side proxy, the Trellis NFS server can support multiple, different users and still have TSI-based secure access across administrative domains.

When combined with the proper "best practices", TSI provides the same overall level of security as SSH, mainly because TSI is closely coupled with SSH's design and mechanisms.

Basic Attacks: An attacker can gain access to a TSI-based overlay metacomputer by either stealing the legitimate user's password or the private key itself. Of course, since both OpenSSH and TSI always transmit passwords and keys over encrypted channels, the main points of the attack are where the passwords and keys are stored (if any) and used. TSI through its automatic ability to start and load remote `ssh-agent`s with private keys, makes it easier to manage the keys, as per SSO, without repeated entries of keywords and passphrases at the keyboard, and without ever requiring the private key be stored on the remote node's disk. To ameliorate attacks based on *privilege escalation*, the forced command feature of SSH can be used to provide only restricted shells and restricted command authorization.

Windows of Vulnerability and Revocation: A well-known technique (e.g., GSI, Kerberos [16]) to limit the window of vulnerability is to limit the lifetime of passwords, keys, or certificates. Admittedly, although OpenSSH provides the ability to set a lifetime for a private key stored in an agent, it is not currently utilized by TSI. As future work, we recognize the potential value of limited-lifetime keys and will consider techniques to both provide this feature and make it convenient to the user. In a related matter, *simple* key revocation

already exists in SSH and, therefore, TSI. Access can be denied by deleting the relevant public key in the authorized keys file of a given account.

High-Value Target: The Trellis NFS Server: Compromising the file server gives the attacker immediate access to a lot of data in the Trellis NFS cache on the server-local disk. There are no easy answers. The first line of defense is to make it difficult to compromise the Trellis NFS server by protecting the password and key of the server. Note that TSI makes it possible for the Trellis NFS server account to not contain any private keys at all. The second line of defense is to limit the damage, even if the server is compromised. Since the server runs purely at the user-level, an attacker can have a great impact on a single user, but unlike root-installed file systems, the attacker does not have automatic access to other users. A third line of defense, which is *not* yet implemented, is to use the forced command feature to automatically shutdown the NFS server, kill its `ssh-agent`, and clear out the cache whenever someone Secure Shells *into* the server account or node. A legitimate user can easily restart the server and agent.

Admittedly, there are other forms of attack and other limitations of TSI not discussed here and not (yet) directly addressed by the system. For example, there are risks associated with using TSI on a system that is compromised. With root access, it is possible to extract private keys from the address space of an `ssh-agent`. We note that the same risk exists with SSH itself and a proper defense, possibly through limited-lifetime keys, is a matter for future work.

5 Other Related Work

Kerberos is a production-quality and well-known security infrastructure that uses key servers (known as, in contemporary versions, Key Distribution Centers (KDC)) within realms to authenticate users with symmetric key cryptographic techniques [16]. Kerberos uses limited-life "tickets" to perform authentication and to limit the window of vulnerability to certain kinds of attacks. Variations of Kerberos are used by the Windows, Mac OS X, and various Unix operating systems. In the context of overlay metacomputing, cross-realm (or cross-administrative domain) authentication is also possible if all domains are running Kerberos and if other Kerberos key servers are trusted (and configured as such) by a local realm or domain. However, it is not always the case that all domains are running Kerberos in an overlay metacomputer, which motivated our design decision to use SSH. Kerberos is a tested and powerful system, but its need for homogeneous security systems (i.e., Kerberos) in all domains or realms makes it an impractical choice for our application.

The Trellis NFS server is similar to the Self-certifying File Systems (SFS) [17] in the use of a well-known LAN-based protocol (e.g., NFS) for the file system, but a different security and WAN-based data movement protocol. In SFS, the self-certifying hostname strategy is a certificate-less technique that cryptographically encodes the identity of the server into the user-visible name of the host. The advantages of SFS, from a security point of view, is that there is no need for certificates nor an extensive security infrastructure. The disadvantages of SFS include the fact that SFS client and server software (i.e., their modified NFS server and wrapper scripts) must be installed as root on *all* local client and remote nodes. The need to avoid installing software as root and the need to layer and support underlying software, such as SSH, results in different design decisions for both TSI and the Trellis file systems. The Andrew File System [18], typically, uses Kerberos for its cross-domain authentication, with the same trade-offs discussed above.

Finally, it should be noted that Plan 9's factotum [19] 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 and TSI, factotums are designed to be used directly by a variety of programs that require authentication.

6 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. We have proposed a security infrastructure for HPC 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.

The main contributions of this work are:

(1) Presented a proof-of-concept that TSI is a useful, user-level, and (reasonably) secure security infrastructure.

Specifically, TSI provides single-sign on and support for background processes (e.g., servers). A user-level system based on Secure Shell maximizes the number of systems that can be aggregated and minimizes the deployment effort.

(2) Described an experiment that tested the effectiveness and scalability of TSI. For the CISS-3 overlay metacomputer, TSI scaled to over 4,000 processors across 22 administrative domains and the Trellis metascheduler used TSI to complete thousands of jobs in two computational science workloads.
(3) Described a strategy that allows the user-level Trellis NFS distributed file system server to have secure access to the data for multiple users on home nodes in different administrative domains.

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, Edmund Sumbar, Ron Senda, 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] D. Barrett, R. Silverman, SSH, the Secure Shell: The Definitive Guide, O'Reilly and Associates, 2001.

[2] M. Closson, P. Lu, Bridging Local and Wide Area Networks for Overlay Distributed File Systems, in: Proc. 2nd Workshop on Real, Large Distributed Systems (WORLDS '05), San Francisco, California, U.S.A., 2005, pp. 49–54.

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

[4] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, V. Welch, A National-Scale Authentication Infrastructure, IEEE Computer 33 (12) (2000) 60–66.

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

[6] C. Pinchak, P. Lu, 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, 2002, pp. 85–105.

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

[8] M. Kan, D. Ngo, M. Lee, P. Lu, N. Bard, M. Closson, M. Ding, M. Goldenberg, N. Lamb, R. Senda, E. Sumbar, Y. Wang, The Trellis Security Infrastructure: A Layered Approach to Overlay Metacomputers, in: The 18th International Symposium on High Performance Computing Systems and Applications, 2004, pp. 109–117.

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

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

[11] M. Closson, The Trellis Network File System, Master's thesis, Department of Computing Science, University of Alberta (2004).

[12] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the Condor experience., Concurrency - Practice and Experience 17 (2-4) (2005) 323–356.

[13] MOLPRO Quantum Chemistry Package, `http://www.molpro.net/`.

[14] H. J. C. Berendsen, D. van der Spoel, R. van Drunen, GROMACS: A message-passing parallel molecular dynamics implementation, Comp. Phys. Comm. 91 (1995)) 43–56.

[15] A. D. MacKerell, Jr., B. Brooks, C. L. Brooks, III, L. Nilsson, B. Roux, Y. Won, M. Karplus, CHARMM: The Energy Function and Its Parameterization with an Overview of the Program, The Encyclopedia of Computational Chemistry 1 (1998) 271–277.

[16] J. G. Steiner, B. C. Neuman, J. I. Schiller, Kerberos: An authentication service for open network systems, in: Proceedings of the USENIX Winter Technical Conference, USENIX Association, Berkeley, CA, 1988, pp. 191–202.

[17] M. Kaminsky, G. Savvides, D. Mazières, M. F. Kaashoek, Decentralized user authentication in a global file system., in: Proc. Symposium on Operating System Principles (SOSP), 2003, pp. 60–73.

[18] J. Morris, M. Satyanarayanan, M. Conner, J. Howard, D. Rosenthal, F. Smith, Andrew: A Distributed Personal Computing Environment, Communications of the ACM 29 (4) (1986) 184–201.

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