ITL Bulletin: WebSubmit

Ryan P. McCormack, John E. Koontz, Judith Devaney

January 4, 1999

1 Introduction

Effective use of high-performance computing (HPC) systems can be a daunting task. Users must deal with an array of constantly changing hardware fronted by diverse operating systems. The tools for similar tasks – for example, job queuing – can vary from system to system, even when the operating systems (UNIX variants) are nominally the same. One barrier to using HPC systems can be overcome by providing users with a single, easy-to-use interface that insulates them from direct contact with the operating systems, tools, and applications on the HPC systems. **WebSubmit**, a browser-based gateway to remote applications, is one way to do this. It provides a friendly, system-neutral environment in which trusted users can access application software on HPC systems.

Trusted is the key word here. The Web is used mainly to transmit documents and images; the introduction of Java has made it also a way to distribute client-side executables safely. Apart from this, programs can be run on remote Web server systems using the Common Gateway Interface (CGI). CGI programming tasks have been restricted to those that can be accomplished anonymously. They are executed as the server user and have only the limited privileges of this account. In most cases, this is appropriate: allowing random users to execute any command on the server system would be giving away the keys to the store. Still, it would be nice to be able to identify valid, trusted users and give them the same privileges they would get with a regular login.

The contribution of WebSubmit is that it provides a novel framework for establishing just this sort of trust relationship in a CGI environment. In this way WebSubmit adds a telnet-like functionality to the ftp-like functionality of the existing Web. The client side execution facility of Java is supplemented by a remote execution facility that can run user-owned jobs on existing, unmodified legacy systems, including HPC systems, the application discussed here. The familiar, pleasant user interface used in Web browsing is extended from document retrieval to remote execution of user programs using their own files.

The primary goal of the WebSubmit project is to provide users with seamless access to a collection of HPC resources. The ideal has been to create an environment in which, from the user's viewpoint, the distributed nature and heterogeneity of the resources disappear. The system is anticipated to have several impacts on the user community: (1) HPC resources should be accessible to a wider class of users, (2) customized execution environments should simplify and speed tasks, and (3) users should be insulated from changes in operating or queuing systems assocaited with various HPC resources. WebSubmit is not intended to be a distributed computing system, although it is extensible in that direction. In this sense, the scope of WebSubmit is not as large as that of metacomputing projects that create and provide access to a distributed computer. At present WebSubmit provides access to batch queues and to a range of interactive utilities including command execution, file editing and file transfer on several HPC systems at the National Institute for Standards and Technology (NIST). The currently supported systems are an IBM SP2 running LoadLeveler, two SGI Origin 2000 systems running SGI's Network Queuing System (NQS), and a Linux-based Pentium array running the Load Sharing Facility (LSF). We hope it will be obvious that WebSubmit is not limited to these HPC systems, or to HPC applications generally, or, indeed, to any particular kind of application at all. In this report, a general discussion of WebSubmit is given, along with a more detailed analysis of its security infrastructure.

2 An Overview of WebSubmit

WebSubmit operates over a set of networked systems, linked by common Web and Internet protocols in a simple transaction model. The user interface, which appears in the user's browser, is composed of a group of application modules, each of which is implemented by a pair of CGI scripts. These CGI scripts reference some shared library code. The software is modular, flexible and extensible, with hooks for including existing CGI code, and for developing and adding new applications. The code is portable and can be modified to suit the needs of a given site quickly and easily.

2.1 The WebSubmit Transaction Model

The WebSubmit user interface is set up to help users accomplish specific tasks on one or more HPC (or other) systems. Each of these tasks is accomplished within a basic transaction model with three parties:

1. **Clients:** the systems of the users requesting performance of tasks

2. **WebSubmit Server:** the system that does user authentication and formats and routes client task requests

3. **Targets:** the (remote) systems on which the tasks are performed

The WebSubmit server is configured to interact with a specific group of one or more target systems (hereafter referred to as the WebSubmit *cluster*) specified by the WebSubmit administrator. For security reasons, a particular WebSubmit server can interact only with systems within its configured cluster.

The user uses a Web browser on the client system to obtain a secure connection with the WebSubmit server's master page, then follows a link on that page to the *application module* page for the task of interest. The module page is the user interface to a task. It is an HTML form that the user fills out and submits to the WebSubmit server. Modules can be in generic format, requiring the user to specify the target system in the form, or in specific format, restricted to a particular target.

The WebSubmit server processes the submitted form, performing any target-side error checking of the input data, and executes the specified task on the proper target system. Execution may consist of submitting a job to the job queue on the target, or of running a command script. Output from whichever is the case is then returned to the user's browser for viewing. If the task is a job queue submission, the output returned is that produced by the act of submitting the job, not the final output of the job itself. As we have formulated our interface, it is up to the user to keep track of the progress of the job and to retrieve the final output and direct it to subsequent jobs.

2.2 Generic Modules

Three generic modules exist at this point: a command execution interface, a simple file editor, and a file transfer utility. The command interface allows users to execute arbitrary commands on remote (UNIX) systems. The file editor (based on HTML text areas) allows users to make quick changes to text files on remote systems and save them. The file transfer utility provides a way to transfer single files (text or binary) between systems.

2.3 Host-Specific Modules

Host specific modules have been developed for three different types of HPC systems and the job queuing software currently used with these systems at NIST:

1. an IBM SP2 with LoadLeveler

2. two SGI Origin 2000 systems with NQS/NQE

3. a Linux Pentium Cluster with LSF

For each of these systems a general module is provided for submitting batch jobs to the queuing system (e.g., LoadLeveler) and another module to monitor the jobs on the system. More specific interfaces have also been built for Gaussian (a quantum chemistry package), and for parallel Message Passing Interface (MPI) jobs. In principle, there are few constraints on the types of modules that can be constructed. The limits are mainly the imagination and needs of the user community and the time of the developers.

2.4 Primary Software Features

Uniform Interfaces for Similar Tasks: Interfaces exist for several different types of systems, but for a specific task, the interfaces between systems look very similar. For example, the interfaces to submit batch computing jobs on the LSF and NQS systems are almost identical.

Modes: In an attempt to support a range of user skill levels, each application module is equipped with basic and advanced modes. These present different users with different levels of detail in the interfaces.

Session Libraries: Some interfaces require fairly lengthy HTML forms; repeatedly filling out these forms is time-consuming and tedious. This problem was addressed by creating the concept of session libraries. Users can fill out the elements in an application module and then save this data as a named library. Any set of data from this collection of libraries can then be loaded at a later time, thus allowing the user to create customized templates for future work.

Automatic Configuration Updates: Batch queuing systems often undergo changes in the structure of the queues (e.g., how much memory they can use). WebSubmit removes the need for users to keep track of such updates, since configuration information is updated automatically or maintained by the software administrator.

3 Authentication and Security

One of the primary concerns in a system like WebSubmit is security. Indeed, this is a primary concern in seamless and metacomputing systems in general. The definition of security varies with the context, but it usually encompasses authentication, authorization, and encryption. In the context of WebSubmit, the primary concern from a security standpoint is authentication. Authentication is a part of everyday life in modern society, from ATM cards to driver's licenses to passports; it is essential in many instances to be able to demonstrate identity. The same is true in electronic environments like the Internet, where it is often desirable to provide access to electronic resources for a limited set of authorized users. However, in these environments, there is no recourse to physical means of identification such as photographs. One must resort to other technologies to establish identity. In the past, electronic authentication was most commonly done using login-password identification, but for most, if not all, of today's applications, this method no longer provides the needed level of security.

WebSubmit utilizes a combination of existing secure protocols to accomplish authentication and to allow users to execute commands on remote systems. The basic transaction begins when the client requests access to remote resources with their browser. The client provides authentication to the server, and the server then propagates this authentication to connect to the remote resource. This authentication process can be broken into three stages:

Stage I Client-to-Server authentication

Stage II Identity establishment and authentication translation

Stage III Server-to-Remote execution of client requests

The client provides authentication to the browser once at the outset of a session (usually by giving a password for a local certificate database). This single authentication then offers access to any one of the remote resources on which the client has privileges. This is in distinct contrast to the standard model of login-password authentication with applications such as `telnet`, `ftp`, and `rlogin`. In these systems, a login and password are normally presented for each resource accessed. In the present scheme, the server is responsible for establishing the client's identity, translating this identity into a login name on the remote system, and then executing the client's request. A single password usually suffices to access all systems with a security superior to login-password authentication. At present, it does not appear that any other systems use this novel form of authentication.

3.1 Client-to-Server Authentication

In the first stage of the authentication process, the client must authenticate itself to the server using a Web browser. This transaction is to be mediated by a Web server running on the server machine. At present, there are two standard methods for performing this type of authentication: (1) basic HTTP authentication using a login-password combination, and (2) client authentication based on public-key cryptography. In basic HTTP authentication, the server requests a login-password combination from the client when resources are requested; the login-password combination is encoded (not encrypted) and returned to the server. The server then compares the information presented against a database of registered users. Basic HTTP authentication is insecure since cleartext passwords are transmitted across the network, and in fact may be worse than standard login-password methods (depending on how closely HTTP traffic is monitored on the server). It is subject to password sniffing and dictionary attack (repeated login attempts using a known login name with passwords taken from a carefully-chosen dictionary); it also does not provide the possibility of protecting the user's data during the transaction. Based on these two concerns, basic HTTP authentication was not deemed to be robust enough for the desired system.

Client authentication based on public-key cryptography can be implemented using a Web server that implements the Secure Sockets Layer (SSL) protocol. This protocol allows for strong authentication (superior to traditional methods) and also provides data encryption over the duration of the session. It has become the *de facto* standard for secure communication on the Internet, and is in the process of being upgraded to an Internet standard (TLS - Transport Layer Security). Finally, all recent versions of the two dominant Web browsers support SSL. Based on these facts, SSL-based client authentication was chosen to perform the Client-to-Server stage of the authentication process.

3.1.1 SSL and Digital Certificates

SSL uses a combination of public- and symmetric-key cryptography to perform authentication and encryption. Public-key authentication is performed using digital certificates, and allows for the exchange of a shared secret, which is then used as an encryption key with a symmetric algorithm (e.g., DES). In the present work, we require that authentication occurs in both directions: the client authenticates itself to the server and vice-versa. SSL also supports server-only authentication and anonymous sessions, although these protocols are not of interest in the present application.

Digital certificates are basically containers for public keys, and they act as a means of electronic identification. The certificate and public key are public documents that, in principle, anyone can possess. An associated private key, only possessed by the entity to whom the certificate was issued, is used as a means of binding the certificate to that entity. Users not in possession of this private key cannot use the certificate as a means of authentication. Entities can prove their possession of the private key by digitally signing known data, or by demonstrating knowledge of a secret exchanged using public-key cryptographic methods.

In practice, anyone can generate public-private key pairs and digital certificates, hence it is necessary to determine whether the holder of a certificate is to be trusted. History has demonstrated that trusting clients is often ill-advised, and centralizing trust simplifies matters greatly. Hence, a trusted-third-party model is utilized with digital certificates. The trusted third party used in the realm of digital certificates is a Certificate Authority (CA). A CA can either issue certificates using public keys provided by clients, or it can generate a public-private key pair for the client and then issue the certificate along with the key pair. In either case, the client must demonstrate their identity to the CA by some trusted means. For example, the client could arrange a face-to-face meeting with the CA and present proof of identity. The CA can then issue a certificate with its digital signature that contains this client's public key, as well as information about the identity of the client. This digital signature can be verified by people who have the public key of the CA, thus establishing the chain of trust from client to CA to server.

3.1.2 Establishing a Secure Web Connection

Once the client has a digital certificate, they can attempt to access the SSL-enabled Web server. The client browser and Web server software enter into a handshake protocol when a connection is requested. Certificates are exchanged and verified, and the client generates a shared secret (encrypted under the public key of the server). This shared secret is used to generate the symmetric encryption key, which then provides secrecy for the session. Once the handshake protocol is completed and encryption keys have been established, client and server are authenticated, and secret data can be exchanged safely. The process of establishing an SSL session does **not** provide the Web server with the identity of the client. It merely demonstrates that the client has a valid, signed certificate that the server trusts, and that the client has the associated private key.

3.2 Identity Establishment and Authentication Translation

Once a client has been authenticated by the server (i.e., they have presented a valid certificate and a verifiable signature), the second stage of the process occurs: identity establishment and authentication translation. These processes occur on the Web server host itself, and allow the client's request for remote resources to proceed to the proper target host.

3.2.1 Identity Establishment

After the formation of an SSL connection, additional action must be taken to obtain the certificate data and to map that to a unique identity (a userID to be used with the authentication framework). Obtaining this userID is crucial, because it allows the server to propagate the client's request to the remote system. The userID can be derived from the certificate in a variety of ways, and it should be associated with a single client. This does not preclude a single client from having multiple certificates (and hence multiple, valid userIDs); the mapping may be many-to-one from userIDs to clients.

There is information in the certificate about the client's identity (Name, Organization, Email), but this information may not necessarily be unique. One would like to construct a userID that is based not only upon this information, but also on the public key of the client. One simple solution that presents itself is to require clients to possess specially-formatted certificates that contain information about their `userID` on the system. This does not correlate the `userID` and public-key, however, and creates logistical difficulties with issuing certificates in the required format. The entire certificate itself cannot be used, since this would be cumbersome, but there is another alternative: construct a fingerprint (message digest) unique to a given certificate. Fortunately, cryptographers and mathematicians have devised and analyzed one-way (or hash) functions that accomplish precisely this task.

Message digests are used widely in cryptography for digital signature verification and for ensuring data integrity. A hash function is a many-to-one function that takes an arbitrary-length input message M and constructs a fixed-length output digest or *hash* $h = H(M)$. In the present context, a unique userID is determined by constructing the hash of the client's certificate using a trusted algorithm (SHA-1 or MD5, for example). In order for the userID to be unique, one must have reasonable certainty that another client's certificate will not hash to the same value. This requirement is satisfied as long as the hash function is sufficiently collision resistant. In order to determine the userID in a web environment, code on the server must have access to the client's certificate. This can be accomplished by directing the Web server to place the client's certificate in the environment when needed. Server software constructs a hash of the certificate, at which point the hash (userID) can be used for authentication translation.

3.2.2 Authentication Translation

Once a userID has been established for a client, an authentication database is used to translate this user's ID into login information on the remote hosts. This authentication database utilizes the userID as the key for each record. Attributes of the database should include, but are not limited to, the following: user name, user e-mail address, user status within the system (active or inactive), and the user's login names on the collection of machines that can be accessed by the system.

When a registered client makes a request to access a remote system, the user's active status is first verified. If they are not active within the system, they are not allowed access to resources. This essentially amounts to the possibility for revocation of access privileges, in addition to those provided by the client certificate's validity period and any CA revocation lists in use. Once the user's active status is verified, the userID-remote host combination is used to index into the database, which determines the login of the user on the remote system. At this point, the request can be propagated to the remote system by the server software.

3.3 Server-to-Remote Execution

In the present architecture, the web server host acts as a proxy for handling client requests. The web server is the agent that accomplishes remote execution, performed by running a command on the server that in turn spawns the remote command. The commands on the server and remote system run under (possibly distinct) usernames. Regardless of what server-side username is used to initiate remote command execution, there needs to be a mechanism for this execution. The authentication system developed should utilize existing technologies where possible, since this minimizes the amount of specialized, and possibly untrusted, software running on the remote hosts. One common means for executing commands remotely on UNIX systems is via the remote shell (`rsh`) command. Using appropriately configured user accounts, commands can be executed from the server host in a client's account on the remote system. However, `rsh` does not protect against the possibility of unauthorized clients masquerading as the server host. A method of executing commands remotely that is not subject to this attack, and that provides encryption, is the Secure Shell (SSH) protocol.

SSH has grown in popularity since its introduction, and is in the process of being considered for an Internet standard. The software has been ported to a wide variety of UNIX platforms; both commercial and non-commercial versions are available. SSH has several features that make it attractive in the present context: (1) Strong authentication methods prevent identity spoofing, trojan horses, and similar means of attack, (2) Encryption and compression of data, and (3) Secure means for file transfer. These qualities precisely meet the needs of the problem being addressed, hence SSH was chosen as the means to execute commands on the remote system. SSH uses a hybrid cryptosystem similar to SSL; a shared secret is exchanged using public-key cryptography, and then data is encrypted using a symmetric cipher based on the shared secret. Server authentication is performed using public key cryptographic methods, whereas several possibilities are provided for client authentication. In the present approach, secure host-based authentication (called `RhostsRSAAuthentication`) is used, since this allows the Web server proxy to execute commands on the remote systems *as the user*, without the need for password exchanges.

4 Discussion: Policy Issues

4.1 Certificate Authorities

Certificate authorities are a means for centralizing trust, so that the server need not trust each individual client. The server, however, must trust the CA to vouch for the identity of clients. As mentioned earlier, numerous commercial CAs exist that can issue certificates to clients. However, it may be that there is no reason to trust a commercial CA more than one would trust clients. In such a case, it will be necessary to use and maintain a CA dedicated to the system in use. The use of a CA for digital certificates raises two other concerns: the protocols used for issuing and distributing certificates to clients. Resolution of these twin concerns depends largely on the site under consideration and on the CA ultimately chosen to perform the task. Needless to say, however, the difficulty in choosing and using a CA should not be underestimated, since the CA is one very crucial link in the entire authentication process.

4.2 Firewalls

Firewalls protect one network from another by filtering traffic, and their use is becoming more widespread, especially for large organizations. Many firewalls are configured to block Web server traffic (`HTTP` or `HTTPS`). In addition, many firewalls block `rsh` requests, and may consider `ssh` requests equally unreliable. For these reasons, firewall policy for the server host and the remote systems must be considered. If the server host is behind a firewall, then one must consider whether clients outside the firewall will be using the Web server to access remote hosts. If this is the case, then the firewall must pass at least HTTPS traffic to allow SSL connections to the Web server. If all clients of the system are within the firewall, then this is of no concern. In order for clients to have access to remote (target) systems, these systems must be open to SSH traffic from the server host. Any remote system with a firewall that does not pass SSH packets from the server host will be unusable in the present scheme.

4.3 SSH

Some systems discourage the use of `rsh` with no-password access because of the danger this poses to the system through identity spoofing. SSH can be similarly configured (i.e., to provide access without passwords), but the means through which this is achieved are totally different. SSH strong authentication essentially prevents identity spoofing. Hence, the only concern with the present approach is whether the client and remote host are comfortable with allowing the server host Web user to execute commands on behalf of the client. By accepting the server host's public SSH key, the remote system acknowledges trusting the server host. A client's trust in the server host is equivalent to the trust they place in any system administrator. The administrator of the Web server would be the only one who could act in their stead (barring root compromises of the server host). Concern regarding the trustworthiness of the server is thus in the hands of the remote system and the client; either can disable trust at any time with little effort.

4.4 Remote System Policy

One pivotal issue involving cluster systems is obtaining usernames from these systems for each valid WebSubmit user. This information is required in order to properly propagate the chain of trust established during authentication and authorization at the WebSubmit server, or, in short, to execute the user's task in the user's own account. If a policy is in place that prevents distribution of this information, then less reliable methods must be used (e.g., getting the username from the user). Hopefully, the WebSubmit and target system administrators will know each other, or overlap, in which case the distribution of login information will not be a major problem.

5 Conclusions

WebSubmit is a flexible, modular framework for accessing and using remote computing resources across the World Wide Web. Though it has been developed at NIST for use as an interface to high-performance computing systems, it is certainly not limited to this field of endeavor. The system impacts the user community by making resources more accessible, simplifying and speeding task execution, and insulating users from changes in the way remote resources are controlled. WebSubmit should be useful in any circumstance where a user community needs authenticated individual access to applications on remote systems (assuming a certification authority is available). It is designed to be portable and can be installed at most sites with a minimum of effort. It can support an existing body of CGI code, as well as providing a framework for developing new applications. The security framework implemented in WebSubmit is novel and robust; it provides both strong user authentication and data encryption, although it produces some policy issues that may need to be addressed before it can be adopted. In summary, WebSubmit extends the basic conception of the Web as a data archive and retrieval system to one of a general computing environment.

6 Acknowledgments

The authors would like to acknowledge Robert Lipman and Katherine Pagoaga for their previous work on the WebSubmit project. We would also like to thank Don Libes for useful discussions of `cgi.tcl`. James Dray of the NIST security division provided useful insights into the WebSubmit security architecture. John Wack was also very helpful in discussions related to Certificate Authorities.

7 Further Reading

- Ryan McCormack, John Koontz, and Judity Devaney, Seamless Computing with WebSubmit, *Concurrency: Practice and Experience* (in press)
- Bruce Schneier, *Applied Cryptography*, 2nd Edition, John Wiley & Sons (New York, 1996)
- Netscape SSL Overview, `http://home.netscape.com/eng/ssl3/ssl-toc.html`
- T. Ylönen, T. Kivinen, M. Saarinen, T. Rinne, S. Lehtinen, SSH Internet Draft (work in progress), `http://search.ietf.org/internet-drafts/draft-secsh-architecture-02.txt`
- cert R. Housley, W. Ford, W. Polk, D. Solo, PKIX Working Group Internet Draft (work in progress), `http://search.ietf.org/internet-drafts/draft-ietf-pkix-ipki-part1-11.txt`
- S. Garfinkel and G. Spafford, *Practical UNIX & Internet Security*, 2nd Edition, O'Reilly & Assoc. (Sebastopol, 1996)