

The GSI plug-in for gSOAP: Enhanced Security, Performance, and Reliability

Giovanni Aloisio, Massimo Cafaro, Italo Epicoco and Daniele Lezzi
Center for Advanced Computational Technologies
University of Lecce, Italy
{giovanni.aloisio, massimo.cafaro, italo.epicoco, daniele.lezzi}@unile.it

Robert van Engelen
Computer Science Department
Florida State University, USA
engelen@cs.fsu.edu

Abstract

In this paper we report on the current status of the GSI plug-in for gSOAP, an open source solution to the problem of securing web services in grid environments.

1 Introduction

Many scientific endeavours rely on a distributed computing infrastructure; recently, the scientific community has started the transition from a general-purpose distributed computing infrastructure to grid environments capable of tighter security, scalability, manageability, availability and reliability. A number of technologies have been proposed and used for distributed computing, e.g., RPC (Remote Procedure Call), Java RMI (Remote Method Invocation), CORBA (Common Object Request Broker Architecture) and so on. While many of these technologies are widely used today, distributed computing has started to embrace the key idea of SOA (Service Oriented Architecture), and the consensus of both scientific community and ISV (Independent Software Vendors) has converged on the recent development of the Web services framework.

The W3C Web Services Architecture Working Group defines a Web service as “a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.” [1]. It is worth noting here that, conceptually, the Web services framework functionality is very similar to CORBA, and that a key point is in *how* the functional-

ity is achieved, i.e. with open protocols. These protocols are SOAP (simple Object Access Protocol) for XML based messaging, WSDL (Web Services Description Language) for description of services, UDDI (Universal Description Discovery and Integration) and finally HTTP (Hyper Text Transport Protocol) and HTTPS (HTTP over Secure socket Layer) as transport layer protocols.

In the past few years the grid community contributed several middleware and environments for developing, building and deploying large-scale distributed applications integrating legacy code, scientific instruments, sensors, visualization, computational and information resources belonging to diverse organizations, collectively referred to as Virtual Organizations. Well-known grid environments include the Globus Toolkit [2] and Unicore [3]. The Globus Toolkit can be considered as the *de facto* standard middleware for grid computing, since it has been deployed worldwide at universities and research centers.

In the previous vision of the grid the attention was focused on the *protocols* needed to provide interoperability among Virtual Organizations components [4]. The current vision is captured by the recent Open Grid Services Architecture (OGSA) [5], that shifts the attention to services as follows: the grid is viewed as an extensible set of *Grid Services* that may be aggregated to provide new capabilities [6]. Grid services, as envisioned, retain several features of the Web Services framework; for instance it is highly desirable to retain service description and discovery, and binding of service descriptions to wire protocols.

OGSA thus leverages WSDL and SOAP, but gives its own definition of a Grid service, which is “a (potentially transient) stateful service instance supporting reliable and secure invocation (when required), lifetime management, notification, policy management, credential management, and virtualization”. The definition clearly states the need

for *transient* services in the grid environment besides *persistent* services as provided by the Web Services framework; this entails the need for interfaces able to manage service lifetime, policies and credentials, and to provide support for notification. Virtualization of resources is a natural consequence of the adoption of Service-Oriented computing: computational resources, storage, networks, applications, archives and so on are all presented as Grid Services.

The current version of the Globus Toolkit (3.2.1) is now rapidly evolving to support OGSA grid services, adding support for the most common Web services standards. In particular, the following are considered core security standards:

- WS Secure Conversation;
- WS Security;
- XML Encryption;
- XML Signature.

Given that the standardization process of these and many other Web services proposed standards is currently work in progress, it is reasonable to expect many changes to their specifications in the near future. While these standards are of course desirable, the lack of stability hinders application development and as a consequence, delays the adoption of the current Globus Toolkit for day by day production. Indeed, many projects are still based on the pre-web services version of the Globus Toolkit. The situation is likely to persist until one of the key Web services new standards will be available as part of the next version of the Globus Toolkit, namely the Web Services Resource Framework (WSRF).

In this context, many projects decided to continue developments using the pre-web services version of the Globus Toolkit but leveraging the Web services framework to provide a clean, established and widely available interface to their applications and legacy codes. Some Web services require of course tighter security, and the need to support the established Globus Security Infrastructure (GSI), led us to the development of a GSI framework for the gSOAP Web services toolkit [7]. This enables a seamless implementation of GSI enabled web services, based on the plug-in architecture of the open source gSOAP toolkit. Our package has been developed in the context of the GridLab project [10], an EU funded development effort to provide high level services and grid aware libraries (the GridLab Application Toolkit). It is worth noting here that, while many of the features provided by the GSI are also available using Transport Layer Security (TLS), there are still important features extremely relevant in grid computing applications that the GSI package deals with, e.g. support for delegation of credentials. The GSI provides an implementation of the Generic

Security Services (GSS) API, enhanced to work in grid environments. These extensions are now being standardized through the Global Grid Forum (GGF).

This paper is organized as follows. We describe the gSOAP Web services toolkit in Section 2, and report on the current status of our GSI plug-in in Section 3. Related work is recalled in section 4 and finally, Section 5 concludes the paper.

2 gSOAP, an Open Source Web Services Toolkit

The gSOAP toolkit project [8] is a US federally funded research and development project that aims to simplify the implementation of performance-critical C and C++ Web services applications by leveraging compiler techniques to automate the mapping of C/C++ operations and data types to distributed SOAP/XML Web service operations. The toolkit supports industry-standard Web services protocols. The gSOAP toolkit WSDL importer translates WSDL service definitions into human-readable C/C++ interface definitions utilizing familiar C/C++ header file syntax. The gSOAP's front-end C/C++ compiler translates these interface definitions of the service operations into proxies and service skeletons for SOAP/XML messaging. One of the key features that distinguishes the toolkit from other approaches is the ability of the gSOAP compiler to generate XML serialization routines for encoding native C and C++ data structures in XML for SOAP/XML messaging. To this end, new compiler-based schema-specific recursive descent parsing techniques were developed to enhance the performance of XML parsing and data serialization [9].

3 Securing Web Services in Grid Environments

The previous versions of our GSI implementation for gSOAP exploited the plug-in architecture of the gSOAP toolkit, utilizing an extension mechanism of gSOAP capabilities through its flexible transport layer supporting both synchronous and asynchronous messaging. GSI support was added by overriding gSOAP callback functions that can be used to replace TCP/IP socket and/or HTTP communications. In particular, we used the Globus I/O module. Our choice was dictated by the ease of use of the Globus I/O APIs that provided a uniform I/O interface to stream and datagram communications, with the addition of GSI security. The clear resemblance to standard TCP and UDP APIs made development of the package faster, easier and less error-prone.

However, two reasons motivated us to redevelop the software starting from scratch using a different Globus API.

First, the Globus I/O API was plagued by a severe bug in its internal event handling mechanism that led to code hanging in a `select()` system call indefinitely. The bug was reported to the Globus team but was never corrected in version 2.4.3 of the Globus Toolkit because the Globus developers had already decided to move from the Globus I/O to a newer API that is currently available in the Globus Toolkit v3.2.1 and is called eXtensible Input Output library (XIO). Even though a compatibility Globus I/O library has been written on top of Globus XIO, some of the original functions in the API were discarded in the new compatibility library, thus making impossible for our software to be compiled and linked using the new Globus I/O package.

The second reason for rewriting the GSI plug-in originated from our desire to improve the performance of our software. To this end, we decided to exploit the Globus GSS libraries. While this approach requires dealing with the intricacies of the GSS library (which is considerably harder than the Globus I/O library) and a thorough knowledge of security aspects, it provides of course better performances, since there are no other intermediate Globus software layers. Another important consequence of our design choice to limit the number of software layers is that fewer bugs can arise due to faulty intermediate layers (as was the case for the Globus I/O package).

Our latest version of the software provides the following features:

- based on the GSS API for improved performance and reliability;
- extensive error reporting related to GSS functions;
- debugging framework;
- support for IPv4 and IPv6;
- support for development of both service and client applications;
- support for mutual authentication;
- support for authorization;
- support for delegation of credentials;
- support for connection caching.

3.1. GSI plug-in implementation

We now describe the current implementation, highlighting relevant changes. The GSI layer is added to the gSOAP stack using a plug-in. A web service application developer adds the GSI layer by simply registering the plug-in. Internally, the plug-in accesses the gSOAP run-time environment and overrides its transport layer by redefining its transport function callbacks:

- *fopen* is overridden by *gsi_connect*;
- *fsend* is overridden by *gsi_send*;
- *frecv* is overridden by *gsi_recv*;

There is no need, as in previous versions of the plug-in, to override the *fpost* and *fposthdr* callbacks, which were meant to change the HTTP POST handling operations, since the latest gSOAP releases handles the HTTPG (HTTP over GSI) protocol by providing automatic support for HTTPG headers. The initialization function determines the debug level requested by the user retrieving the value of the environmental variable `GSI_PLUGIN_DEBUG_LEVEL`. The new debug framework is quite flexible, allowing the developer to choose among multiple debug levels. A debug level of zero fully disables debug, level one provides information related to entering and exiting plug-in functions, level two adds execution summary information and finally, level four adds display of internal state, i.e., information related to the plug-in local variables.

Two additional callbacks (*fcopy* and *fdelete*) must be overridden, so that the gSOAP environment can respectively copy the plug-in's local data and deallocate or cleanup them when de-registering the plug-in. It is worth noting here that gSOAP requires a deep copy to avoid sharing the plug-in local data by two or more gSOAP run-time environments. The plug-in's local data include a pointer to a callback function for customized authorization, listening and connected file descriptors, file pointers for reading and writing, the distinguished names that identify a client or server when mutual authentication is performed, the established GSS security context and the credential used, possibly a delegated credential, the distinguished name of the target Web service if a client requests delegation of credentials (as required by the latest Globus GSI package) etc. The package now provides extensive error reporting related to GSS functions used. Previous versions of the plug-in did not fully explain GSS errors, making sometimes difficult to understand the exact nature of problems experienced. We are also fully supporting protocol independent (IPv4 and IPv6) networking in the latest releases. Connection caching is also supported, through the HTTP keep-alive mechanism. Once the plug-in is registered, a GSI enabled Web service or client can be easily implemented.

3.2. Developing GSI Enabled Clients

We begin describing shortly how the plug-in functions utilize the Globus Toolkit GSS API to provide gSOAP with automatic, transparent transport-level security. Before starting a connection, clients need to activate relevant Globus modules (common and `gsi_gssapi`), to acquire the credential to be used and to specify the behav-

ior of the GSI communication. Credential must be acquired using *gsi_acquire_credential*. This function utilizes *gss_acquire_cred* to retrieve the user's credential from her proxy. The GSI communication properties can be fully specified by means of the functions *gsi_set_delegation*, *gsi_set_confidentiality*, *gsi_set_integrity*, *gsi_set_replay* and *gsi_set_sequence*. These functions can be used optionally to request respectively delegation of credential, message confidentiality (encryption), message integrity (anti tampering), replay attack detection and out of sequence message detection.

Clients connect to remote GSI enabled Web services taking advantage of the *gsi_connect* function. This function, invoked transparently by gSOAP, tries a protocol independent connection (IPv4 and IPv6 are both supported) to a target Web service. If the connection succeeds, an attempt is made to establish a GSS security context by calling *gsi_init_security_context*. Establishing a security context implies a mutual authentication process and the exchange of GSS tokens. Client-side, we exploit *gss_import_name* to import the Web service target name (if provided by the user, as requested for delegation of credential by the latest GSI package) and then we try to establish a security context by calling in a loop *gss_init_sec_context* and the functions *gsi_send_token*, *gsi_rcv_token* to send and receive GSS tokens (these are based on the Globus functions *globus_gss_assist_token_send_fd* and *globus_gss_assist_token_get_fd*, modified to use a Unix file descriptor instead of a FILE * argument).

The security context is deleted with *gss_delete_sec_context* in case of error, otherwise context establishment has been successful and the remote Web service *actual* target name (it may differ from the one supplied by the user) is retrieved from the context using *gss_inquire_context* and *gss_display_name*.

If no errors were detected, the next step is to perform an authorization process. This is done invoking the authorization callback, if one has been provided by the user. If the authorization callback succeeds, then the client proceeds invoking one of the Web service methods. Data is sent to and received from a Web service transparently, utilizing the *gsi_send* and *gsi_rcv* functions. To send a message, we simply wrap it with *gss_wrap* and send it using *gsi_send_token*. Receiving a message is more complicated, since we need to preserve the semantics of a standard *rcv* function. If *gsi_rcv_token* receives from the network a number of bytes that is less than or equal to the length of the buffer supplied by the caller, then we can unwrap the message with *gss_unwrap* but upon reception of a number of bytes greater than the length of the buffer supplied by the caller, we need to carefully manage the bytes in excess that must be safely stored in a new buffer and returned to the user in successive calls to the function.

The following code snippet shows how easy it is to use the GSI plug-in inside a client. The developer activates the required Globus modules, initializes the gSOAP environment, registers the GSI plug-in and sets GSI channel properties before calling a remote web service method using the HTTPG (HTTP over GSI) binding. Before exiting, the gSOAP environment is deallocated.

```
...
globus_module_activate(GLOBUS_COMMON_MODULE);
globus_module_activate(GLOBUS_GSI_GSSAPI_MODULE);

/* We init the gsoap runtime environment
 * using soap_init2()
 * instead of soap_init()
 * this is needed to support connection caching
 */
soap_init2(&soap,
           SOAP_IO_KEEPALIVE,
           SOAP_IO_KEEPALIVE);

/* register the GSI plug-in */
if (soap_register_plugin(&soap, globus_gsi)) {
    soap_print_fault(&soap, stderr);
    exit(1);
}

/* setup of authorization callback */
data = (struct gsi_plugin_data *)
        soap_lookup_plugin(&soap, GSI_PLUGIN_ID);
data->gsi_authorization_callback =
my_gsi_authorization_callback;

/* acquire our credential */
rc = gsi_acquire_credential(&soap);
if (rc < 0){
    soap_destroy(&soap);
    soap_end(&soap);
    soap_done(&soap);
    exit(-1);
}

/* setup of Web Service target name
 * required for delegation of credential
 */
target_name =
strdup("/O=Grid/OU=unile.it/OU=CACT/CN=Massimo Cafaro");

/* setup of GSI channel */
gsi_set_delegation(&soap, GLOBUS_TRUE, target_name);
gsi_set_replay(&soap, GLOBUS_TRUE);
gsi_set_sequence(&soap, GLOBUS_TRUE);
gsi_set_confidentiality(&soap, GLOBUS_TRUE);
gsi_set_integrity(&soap, GLOBUS_TRUE);
...

```

3.3 Developing GSI Enabled Web Services

Implementing a Web service requires activation of Globus modules, registration of our plug-in, acquisition of credential and optionally setting an authorization callback before listening for incoming connections. The *gsi_listen* function provides a protocol independent (IPv4/IPv6) network listener that must be in place before entering the Web service main loop, where the *gsi_accept* function accepts incoming connections retrieving the peer IP address and port. Then, (usually) a new thread is spawn to serve the client request.

This thread begins execution of a user-defined function that works as follows. The *gsi_accept_security_context* is called to establish a security context with the client (performing mutual authentication) and, in case of success, the authorization callback is invoked if provided. Once the user has been authenticated and authorized, the thread serves the client request. Establishing a secure context server-side requires receiving in a loop the client GSS tokens, processing them with *gss_accept_sec_context* and sending back to the client the Web service GSS tokens until the context is not fully established. The token exchange is also needed to negotiate the GSI communication properties requested by the client. If context establishment is not successful, the current context is deleted, otherwise the distinguished name of the client is retrieved and stored in the plug-in local data.

The following code snippet shows the use of GSI plug-in to implement a Web service. The initial steps are the same performed in the client code, and we omit these identical statements.

```

...
/* listen for incoming connections */
data->listening_fd =
    gsi_listen(&soap, NULL, port, backlog);
if(data->listening_fd == -1){
    fprintf(stderr, "Failing in gsi_listen()\n");
    soap_destroy(&soap);
    soap_end(&soap);
    soap_done(&soap);
    exit(-1);
}

/* server's main loop */
for (;;) {

    /* accepting incoming connections */
    data->connected_fd = gsi_accept(&soap);
    if(data->connected_fd == -1){
        fprintf(stderr, "Failing in gsi_accept()\n");
        soap_destroy(&soap);
        soap_end(&soap);
        soap_done(&soap);
        exit(-1);
    }

    /* spawning a new thread
     * to serve the client's request
     */

    tsoap = soap_copy(&soap);
    globus_thread_create(&tid,
        NULL,
        &process_request,
        (void *)tsoap);
}

/* here is an example process_request function */
void process_request(void *arg)
{
    ...
    rc = gsi_accept_security_context(soap);

    if (rc == 0) {
        rc = data->gsi_authorization_callback(soap);
        if (rc == 0)
            soap_serve(soap);
        else {
            fprintf(stderr,
                "User %s is not authorized\n",
                data->client_identity);

```

```

        soap_receiver_fault(soap,
            "Authorization error\n",
            NULL);
        soap_send_fault(soap);
    }
}
...
}
...

```

4 Related work

The Globus team developed a gSOAP GSI plug-in for the initial C bindings to the Grid services provided in early version 3 of the Globus Toolkit. This plug-in was similar to our previous implementation, exploiting the Globus I/O API. It is worth mentioning here that this software was never made available as production quality software. Moreover, the Globus plug-in could only be used to develop clients, so that this software provided only a partial solution for developers.

Another GSI plug-in, CGSI, was developed by Ben Coururier of CERN using the GSS APIs. This software only allows development of clients or servers, again providing a partial solution (only the traditional client-server paradigm is supported). Moreover, the software is not freely available. Our plug-in allows development of clients and servers (thus supporting the client-server paradigm) and also allows mixed mode: servers can be simultaneously clients of GSI enabled Web services if needed. This is the case in many projects exploiting the peer-to-peer paradigm.

Recently, the Globus team has started development of a GSI enabled soap engine, using the Apache Axis C++ soap implementation. The project, called ScOAP, is in its early stage, since the software has been released in a call for community testing, thus it can not be used for production. Moreover, gSOAP currently outperforms Axis C++ [9], provides better interoperability, and offers more features including document/literal operations. Both the Globus gSOAP plug-in and the new ScOAP engine support message-level security besides transport-level security.

Our plug-in provides an effective solution for building and deploying production level Web services and clients, the API is quite simple but, to date, the software does not support message-level security. However, we plan to add this feature in the near future, since a new gSOAP release is expected to support message-level security natively. The current GSI plug-in is fully interoperable with Grid services built using the latest Globus Toolkit v3.2.1 release (if these services use transport-level security). Thus, our package is fully interoperable with the Globus Java CoG (that provides a Java implementation of GSI) and Java Axis (the current Globus soap engine).

5 Conclusion

In this paper we described the evolution of the GSI plug-in for gSOAP, an open source solution to the problem of securing Web services in grid environments. We have reported the current status of the plug-in, highlighting new features and key points related to the implementation. We have also compared our solution to existing and planned soap engines for the Globus Toolkit. Our plug-in enables seamless integration of both legacy and new applications in grid environments wrapping them as GSI enabled Web services, is distributed under the GNU Public License and is freely available. We plan to continue its development in the near future, by adding message-level security in order to make it fully interoperable with the Globus Toolkit.

Acknowledgment

The authors would like to thank all of the users of the plug-in for their valuable feedback, and gratefully acknowledge support of the European Commission 5th Framework program, grant IST-2001-32133, which is the primary source of funding for the GridLab project. The gSOAP project is supported in part by National Science Foundation grants CCR-0105422, CCR-0208892, and US Department of Energy grant DEFG02-02ER25543.

References

- [1] W3C. Web services architecture requirements. <http://www.w3.org/TR/wsa-reqs>
- [2] I. Foster and C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit*, Intl J. Supercomputer Applications, Vol. 11, 1997, No. 2, pp. 115–128
- [3] J. Almond and D. Snelling, *UNICORE: uniform access to supercomputing as an element of electronic commerce*, Future Generation Computer Systems, Vol. 15, 1999, No. 5-6, pp. 539–548
- [4] I. Foster, C. Kesselman and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, International Journal Supercomputer Applications, Vol.15, 2001, No. 3, pp. 200–222
- [5] I. Foster, C. Kesselman, J. Nick and S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed System Integration*, Technical Report for the Globus project. <http://www.globus.org/research/papers/ogsa.pdf>
- [6] I. Foster, C. Kesselman, J. Nick and S. Tuecke, *Grid Services for Distributed System Integration*, Computer, Vol. 35, 2002, No. 6, pp. 37–46
- [7] G. Aloisio, M. Cafaro, D. Lezzi and R. van Engelen, *Secure Web Services with Globus GSI and gSOAP*, Proceedings of Euro-Par 2003, Lecture Notes in Computer Science, Springer-Verlag, N. 2790, pp. 421–426, 2003
- [8] R. van Engelen et al. *Developing Web Services for C and C++*, in IEEE Internet Computing Journal, March, 2003, pp. 53–61
- [9] M. Govindaraju, A. Slominski, K. Chiu, P. Liu, R. van Engelen, and M. Lewis, *Toward Characterizing the Performance of SOAP Toolkits*, to appear in Proceedings of 5th IEEE/ACM International Workshop on Grid Computing.
- [10] G. Aloisio, M. Cafaro, I. Epicoco and J. Nabrzyski, *The EU GridLab Project: A Grid Application Toolkit and Testbed*, to appear in "Engineering the Grid: Status and Perspective", L. T. Yang, Jack Dongarra, Adolfo Hoisie, Beniamino Di Martino and Hans Zima (Eds), Nova Science Publisher, 2005