High Security Web Servers and Gateways

Nigel Edwards, Owen Rees
Networked Systems Department
HP Laboratories Bristol
HPL-96-163
December, 1996

world wide web, security, objects, CORBA, Java, multi-level security

This paper describes a high security, high performance system for making legacy systems accessible to the web. It combines distributed object technology with a trusted operating system that implements multi-level security. The aim is to satisfy the growing demand for dynamic content generation, while providing a high level of protection against unauthorized access to the service.

HP CORBAweb is a software infrastructure that allows access to CORBA applications from the web. HP VirtualVault is a secure environment for web applications. The paper gives overviews of both VirtualVault and CORBAweb, and describes the object gateway that merges the integration features of CORBAweb with the security of VirtualVault.

The paper describe the authorization model that determines the granularity at which access is granted. It then goes on to explain how the system can be extended to allow remote clients, such as Java applets, to invoke the CORBA-based services directly, using the Internet Inter-ORB Protocol (IIOP).

The object gateway is designed to be used to provide controlled access through a firewall protecting the servers. Some of the issues associated with firewalls around the clients are discussed at the end of the paper.

High Security Web Servers and Gateways

Nigel Edwards, Owen Rees
Hewlett-Packard Laboratories, Bristol

Abstract

This paper describes a high security, high performance system for making legacy systems accessible to the web. It combines distributed object technology with a trusted operating system that implements multi-level security. The aim is to satisfy the growing demand for dynamic content generation, while providing a high level of protection against unauthorized access to the service.

HP CORBAweb is a software infrastructure that allows access to CORBA applications from the web. HP VirtualVault is a secure environment for web applications. The paper gives overviews of both VirtualVault and CORBAweb, and describes the object gateway that merges the integration features of CORBAweb with the security of VirtualVault.

The paper describes the authorization model that determines the granularity at which access is granted. It then goes on to explain how the system can be extended to allow remote clients, such as Java applets, to invoke the CORBA-based services directly, using the Internet Inter-ORB Protocol (IIOP).

The object gateway is designed to be used to provide controlled access through a firewall protecting the servers. Some of the issues associated with firewalls around the clients are discussed at the end of the paper.

Introduction

This paper is about how to make legacy systems accessible to the web in a performant and secure way by using a combination of distributed objects and a trusted multi-level operating system. The aim is to support dynamic generation of content tailored to the client by services connected to the back of the web server, and to support applications like electronic business in which customer order processing will be handled by systems behind the web server.

We begin by describing HP VirtualVault; the application of Compartmented Mode Workstation (CMW) to current web server technology using CGI for legacy system integration and dynamic content generation. Next we give an overview of HP CORBAweb which uses distributed objects to overcome a number of CGI's disadvantages.

These introduce the main contribution of the paper: the architecture of a secure object gateway for providing controlled access through a firewall, and an instantiation of the architecture on a CMW. The gateway is generic: it handles both HP CORBAweb and also general IIOP [OMG 95] invocations made by applets downloaded to client machines. This also demonstrates that IIOP provides the necessary indirection for building IIOP proxies into firewalls.

The guiding principle for the object security architecture is that the service (the interface exposed by an object) is the unit of authorization; we argue that this places no restriction on applicability.

The Compartmented Mode Workstation and Virtual Vault

The HP-UX Compartmented Mode Workstation (CMW) is an enhanced security version of the HP-UX operating system. The key features of CMW that distinguish it from standard HP-UX are

Mandatory Access Control, and the division of the superuser power into a set of privileges. HP-UX CMW is being evaluated for B1 rating under the Trusted Computer Systems Evaluation Criteria (TCSEC) [DOD 85] (the "Orange Book"); an earlier release which passed the B1 rating is also available. CMW [DIA 91] is a different, but related set of criteria to TCSEC; it includes the B1 criteria, and many of the features from the B2 and B3 TCSEC classes. Standard HP-UX offers the security features of TCSEC class C2.

VirtualVault [HP 96c] is an HP product that builds on CMW, and exploits its security mechanisms, to provide a high security web service that offers controlled access to internal applications from an external web.

Sensitivity labels and Mandatory Access Control

The typical VirtualVault configuration runs on a host with two network interfaces. One interface is connected to an external, potentially hostile, network - the Internet. The other interface is connected to an internal network - a corporate intranet. These networks must be kept separate with only the most carefully controlled communication between them.

The trusted operating system labels all processes, files, network interfaces and other resources with sensitivity labels. These labels are used by the system's Mandatory Access Control mechanisms to restrict how processes may access resources. VirtualVault uses these sensitivity labels to separate the Internet on the "outside" from an intranet on the "inside", as shown in Figure 1 below.

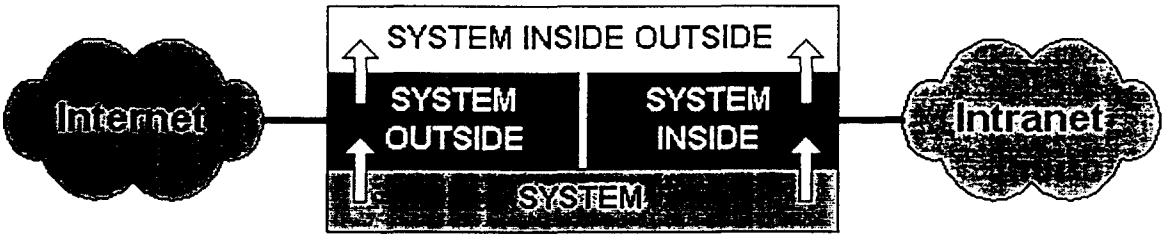


Figure 1: Virtual Vault sensitivity labels

In essence, Mandatory Access Control (MAC) ensures that information may only flow "upwards" in a partially ordered set of labels. This means that information coming from the network interface labeled "SYSTEM INSIDE" cannot be accessed from the network interface labeled "SYSTEM OUTSIDE", and vice versa. Since the labeling is applied to processes and all other system resources, Mandatory Access Control ensures that the networks are separated.

Program files, and data, such as HTML pages, that are to be visible from both inside and outside are labeled "SYSTEM". These can be read by processes running at either "SYSTEM INSIDE" or "SYSTEM OUTSIDE" and thus are available to both networks. The MAC rules do not allow such processes to write into files labeled "SYSTEM", so program files and public data are protected against modification by processes connected to either network. If the public data includes applets, or other code that is to be executed by clients, this protection against modification is especially important.

The system has extensive auditing facilities, and the audit data is labeled "SYSTEM INSIDE OUTSIDE" rendering it invisible to both networks. An administrator authorized to act as the "Information System Security Officer" has the necessary rights to examine the audit data, and to establish alarms to be triggered by particular audit events.

Sensitivity labels and MAC are pervasive throughout the system, and enforced by the system, regardless of the apparent ownership of data. Conventional Discretionary Access Control (DAC) provided by file mode, and user and group ids, as available in standard HP-UX, provides some

degree of protection, but this can be undermined by programs executing on behalf of the data owners. With DAC, programs run by the user may grant inappropriate rights, or copy data to unprotected files; unsuspecting users may run programs that have been maliciously altered to violate security in this way. With MAC, users cannot be tricked into compromising the data by "trojan horse" code inserted into the programs that they run.

Privileges and Authorizations

The trusted operating system does not have a superuser, rather, it divides that power into a number of privileges. For example, the `netprivaddr` privilege allows a process to bind to privileged local addresses such as Internet protocol ports in the range 0 to 1023, without granting any other rights.

The system goes further by separating the rights granted to a user, from the privileges associated with a process. Users can be granted "command authorizations", and commands can check if the user has an authorization before making a potential privilege, associated with the command, effective. This requires explicit action by the program, known as "raising the privilege", and good practice dictates that privileges are lowered as soon as possible after use. For example, the `chroot` command tests whether the user has the `chroot` authorization before it raises the `filesysops` privilege that is needed to perform the `chroot` system call successfully. Attaching the privileges to commands, rather than to users, means that there are no privileges to be exploited by other programs run by the user. This means that trojan horse programs do not gain access to the privileges that users can exercise through authorizations and trusted commands.

In addition to the privileges which replace the power of the superuser, there are privileges that can be used to allow trusted programs to override the strict rules of the security policy. It is these privileges that make it possible to provide a controlled path between the outside and inside networks.

Web Servers and the Trusted Gateway Agent

Off-the-shelf web servers contain too many complex functions, and, in particular, are too extensible to be trusted. Therefore the VirtualVault does not attach any privileges to the processes that run the web servers. Instead, it runs two web servers, one for the outside network, and one for administration from the inside; and uses a separate Trusted Gateway Agent (TGA) to mediate communication between outside and inside.

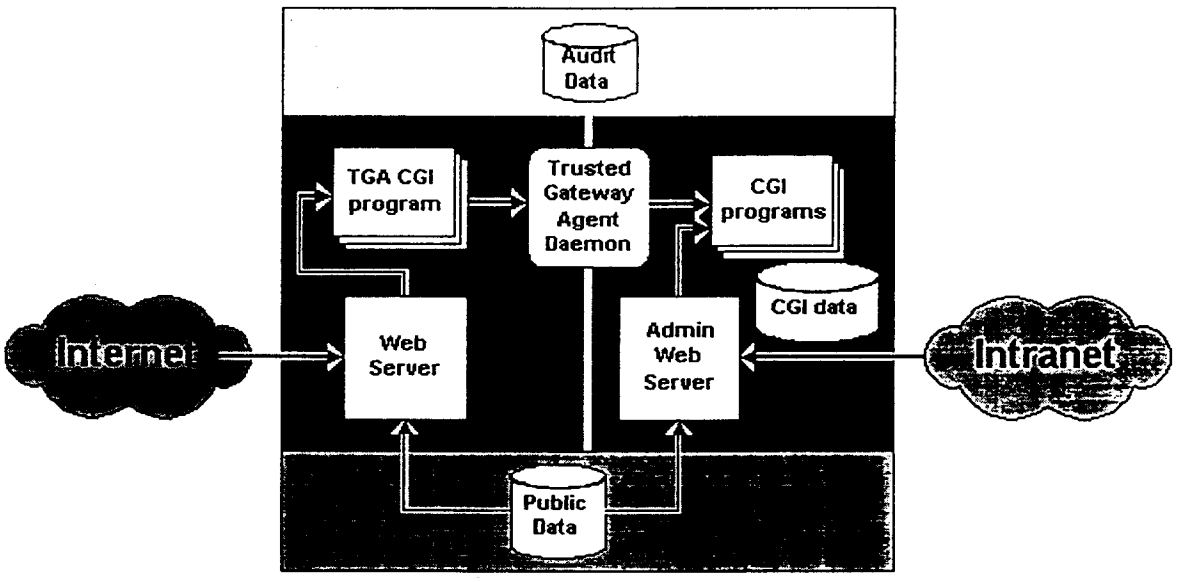


Figure 2: Virtual Vault Web Server Configuration

Static data, including HTML and applets to be delivered directly to client browsers, is held in files labeled "SYSTEM". These files are readable by both servers, but writable by neither. Even if an intruder manages to make the web server execute malicious code, it cannot modify the data, and, in particular, it cannot insert trojan horse code into applets that will be served to clients.

Incoming requests from outside for pages with dynamically generated content, conventionally served by CGI, must be forwarded to the "inside" context, since that is where the data and services used to generate the content are available.

CGI programs are installed in the usual way on the inside server, and these can then selectively be made available from the outside server. The outside server uses CGI to invoke a privileged TGA program. Copies of the TGA program are installed on the outside server, with the same names as the inside CGI programs. The TGA program has potential privileges that that it can raise to communicate with the Trusted Gateway Agent Daemon (TGAD). The TGAD checks that the request is permitted, and if it is, uses its privileges to invoke the inside program at the "SYSTEM INSIDE" level, using CGI, as if TGAD were the inside server.

The CGI programs will run at "SYSTEM INSIDE", launched either by the TGAD for requests from the Internet, or by the inside admin web server for requests from the intranet. This means that any data they write will be labeled "SYSTEM INSIDE" and not directly accessible to the outside server. The CGI programs are also at the level necessary to open connections to systems on the inside network, as would be required if they are to access legacy systems on an intranet.

The VirtualVault approach is effective and secure, but suffers from the limitations of CGI. In particular, the performance problems of CGI will be made worse by the need to launch a copy of the TGA program, as well as a copy of the CGI program. The overhead of fork/exec suffered by conventional CGI is doubled.

HP CORBAweb offers a higher performance replacement for CGI, and the components needed to build a secure gateway to object systems.

Overview of HP CORBAweb

HP CORBAweb uses distributed object technology (CORBA [OMG 95]) to support dynamic content generation and legacy system integration into the web. The aim is to support applications like electronic business where customer order processing will be handled by systems behind the web server. In this approach legacy systems are wrapped or encapsulated in an object. For a detailed discussion of the use of CORBA in legacy system integration and migration the reader is referred to [BS 95].

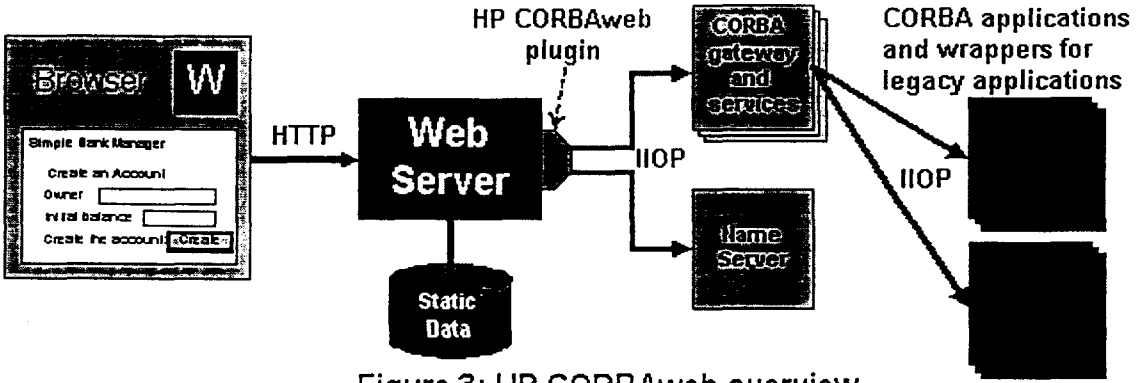


Figure 3: HP CORBAweb overview

How a request is handled

CORBAweb is a plugin replacement for CGI: to access a CORBAweb application, browsers use ordinary HTML forms. The web server is configured to hand HTTP requests (GET or POST) to the HP CORBAweb plugin. The plugin parses the URL, extracts the name of a gateway object, then uses the name server to resolve the name to an object reference. The plugin then invokes the gateway object. The gateway object decodes the parameters; using library routines provided as part of HP CORBAweb to decode URL-encoded data received from the browser. The typical gateway then invokes an application object which may be co-resident with the gateway or remote. The application object may be a CORBA wrapper for a legacy system, or it may be a "pure" CORBA service. Results are passed back via the gateway and the plugin to the browser. The gateway renders the results which can be in any media type that the browser understands.

Using CORBA means that we have much choice over placement of the gateway object: it can be on a remote machine, and can be in a dedicated process or share a process with many other gateway objects. Communication between all CORBA objects including the plugin uses the Internet Inter-ORB Protocol (IIOP) [OMG 95] and so different gateways may utilize different implementations of CORBA (provided they support IIOP).

All gateway objects implement the same interface, which is a mapping of CGI to CORBA IDL (Interface Definition Language). They isolate the plugin from the variety of application interfaces.

Advantages of HP CORBAweb

HP CORBAweb has several advantages over conventional CGI:

- Performance and efficiency: there is no fork/exec overhead to serve each incoming request
- Stateful interaction: since CORBAweb objects persist from one HTTP request to the next they can store state internally
- Support for distribution: the gateway object can be on a separate machine from the web server, with communication via a standardized protocol (IIOP)

CORBAweb also has several advantages over installing the gateway as a plugin directly into the server using the server API (e.g. NSAPI):

- Fault Confinement: since the gateway is not run inside the web server, if the gateway develops a fault and crashes, it will not affect the web server
- Debugging environment: it is easier to attach a debugger to the gateway object (especially if the web server forks multiple copies of itself to service multiple requests concurrently)
- Manageability: to shutdown or install a new gateway it is not necessary to restart the web server; a gateway makes itself visible by registering itself with the name server and removes itself from the name server when it is shutdown

The main disadvantage of CORBAweb with respect to CGI is language dependence: the language used to implement the gateway must be supported by an implementation of CORBA which in turn supports IIOP. To obviate this, at the time of writing we are experimenting with using CORBAweb to provide a remote CGI facility. This uses CORBAweb gateways on remote machines to run CGI scripts on those machines and pass the results back to the web server via the plugin. This has the advantage of moving the fork/exec overhead off the web server machine and allows us to site the CGI script on the most appropriate machine. The use of CORBAweb is hidden from the CGI programmer.

Dynamic Content Generation with Active HTML

Although CORBAweb supports different MIME types being returned to the browser, usually the

results of invoking a gateway are rendered in HTML. The HTML is either embedded in the program as "print" statements or embedded in a server parsed HTML file (the server parses the HTML with embedded invocations of the CORBAweb gateway). The former approach is inflexible: it is hard to change the look and feel of the results. The latter approach results in poor performance (the web server has to parse the HTML for each request).

In an attempt to address these problems, we have developed "Active HTML". This is a C++ class library which parses template HTML documents with embedded CORBA function calls. When a CORBAweb gateway starts, it loads its HTML templates and parses them to create in-memory HTML objects. The parsing machinery has been designed so that the templates are syntactically correct HTML. Thus they can be created with an HTML editor and can be run through an HTML checker.

On receiving an invocation for one of its active documents, the gateway simply invokes a method of the HTML object to display the document. The HTML object that was created by the parser traverses its internal structure to display each component in turn. Static text parts are returned directly, embedded functions may invoke third party services (e.g. a remote database) to generate the dynamic results. Since the HTML object doesn't have to do any further parsing, this approach is much faster than conventional server-parsed HTML, but provides a great deal of flexibility in being able to change the look and feel of the interface without any coding. It is simply a matter of editing and reloading the HTML template.

The Secure Object Gateway

The purpose of the secure object gateway is to give fine grain access control on the services which are accessible from the back-end of the web server. It allows access only to those services which the administrator has authorized explicitly. In addition the administrator can name which users and groups are allowed to access which services.

This section begins with an overview of the architecture. Next it explains how to instantiate the architecture on CMW. Finally, it discusses the authorization model.

Basic Architecture

As shown below, the secure object gateway intercepts all calls made by the CORBAweb plugin to other CORBA services (including the name server).

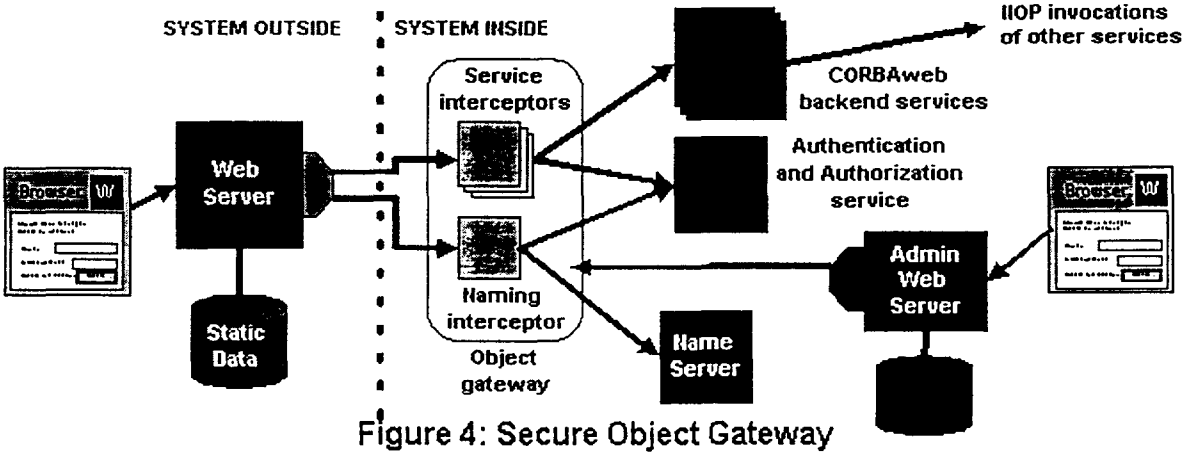


Figure 4: Secure Object Gateway

When the interceptor for the name service receives a request from the CORBAweb plugin to resolve

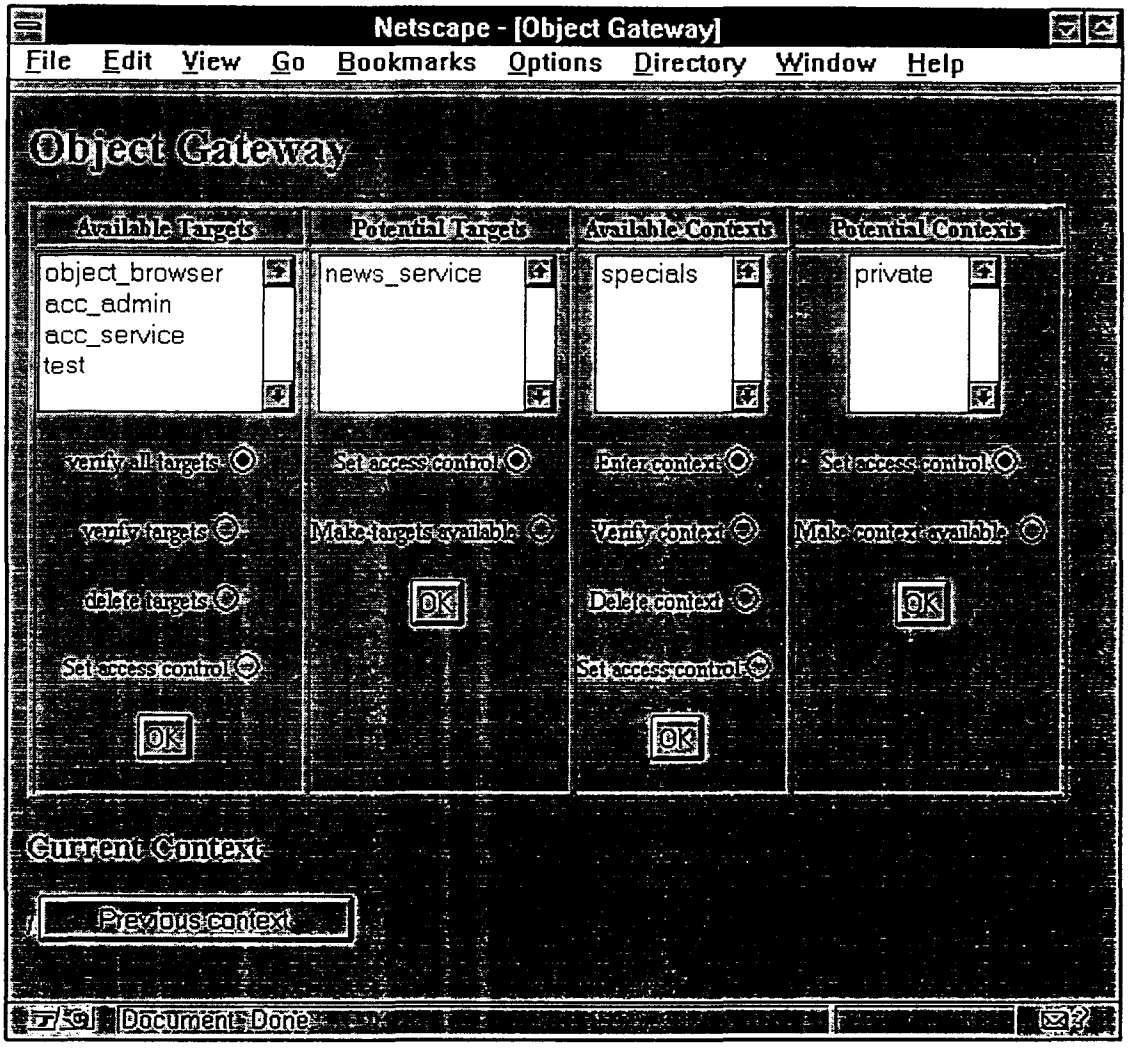
a name, it will return a reference to a service interceptor instead of the actual service. When the plugin attempts to invoke the service, it invokes the corresponding service interceptor. Although the service interceptor is generic, each separate back-end service has its own instance of a service interceptor which is multi-threaded. This allows separate access controls for each back-end service to be set up in the interceptor for that service.

When a service interceptor receives a request for an invocation it checks that the user is authorized to invoke the service. If authorization fails, the request is aborted and an error message is returned. If the request is authorized, the data is passed on to the real service without any further processing.

As shown in the diagram, the object gateway uses a separate authentication and authorization service, built on the Secure Session facility described in [DG 97]. The the Secure Session facility authenticates the user at the start of a session: the user supplies a user-id and password, and is given an unforgeable ticket, the session ID, in a cookie which is attached to all subsequent requests from that user. SSL [FKK 96] is used to provide confidentiality and integrity over the communication links. When the authorization service receives a request for authorization from a service interceptor or a naming interceptor, it checks for a valid session ID, and that the user (or a group to which the user belongs) has been authorized to access the service being requested. Options and possible extensions to authentication and authorization which could be supported by the above architecture are discussed later in the paper.

Administering the Object Gateway

The object gateway is managed using a CORBAweb interface only accessible to an internal web server running in the inside compartment. The administration screen is shown below; back-end services are called "targets".



The first column, "available targets", lists those services for which the object gateway has created service interceptors. When it receives an authorized request to resolve a name corresponding to one of these targets, it will hand out an object reference to the service interceptor for that target, only if the name is in the list of available targets. Any attempt to resolve a name which is not in the list of available targets will result in an "object not found" error.

There are three basic actions which an administrator can apply to the available targets.

1. Verify that the target services still exist by checking that the service is still running and still in the name server
2. Remove the target from the list of available targets
3. Set access controls for the named target

The second column lists the "potential targets". These are the services which are visible in the naming server, but have not yet been made available by the administrator. The administrator can select one or more targets from the list to make them available; this action creates a service interceptor for each back-end target selected. In addition the administrator can also set access controls for targets before they are made available. Although it is possible to change and set access controls once a target is added to the list of available targets, it is advisable to set in place some access controls before the object is made available, unless it is to be accessible by everybody.

The two columns on the right of the table are for managing naming contexts. Naming contexts serve the same purpose as directories in a file system. Starting at the root naming context "/" it is possible to navigate around the hierarchy to other contexts. Each naming context is an object supporting

operations such as "bind" for binding a name to an object reference in that context, and "resolve" for resolving a name to an object reference.

The object gateway reflects the recursive nature of the name server: a naming interceptor may contain other naming interceptors as well as service interceptors. The administrative actions which can be taken on potential and available contexts are similar to those which can be taken on potential and available targets. For example, making a context available causes a naming interceptor to be created. This is also an "enter" action on available contexts. This changes the current naming context of the administration utility.

Instantiating the Gateway on CMW

In a Virtual Vault configuration, the CORBAweb gateways and services, and their naming service will be running on the "inside" network. Some or all of these could also be run at "SYSTEM INSIDE" on the CMW host. The CORBAweb plugin on the externally visible server will be running at "SYSTEM OUTSIDE".

The object gateway must be the only route to services running on the inside network. The object gateway must provide a way to cross the compartment boundary, and it must mediate all accesses that cross the boundary to ensure that they are authorized. These two functions are implemented by separate components, a proxy that forwards interactions across the boundary, and interceptors that ensure that the interactions are authorized.

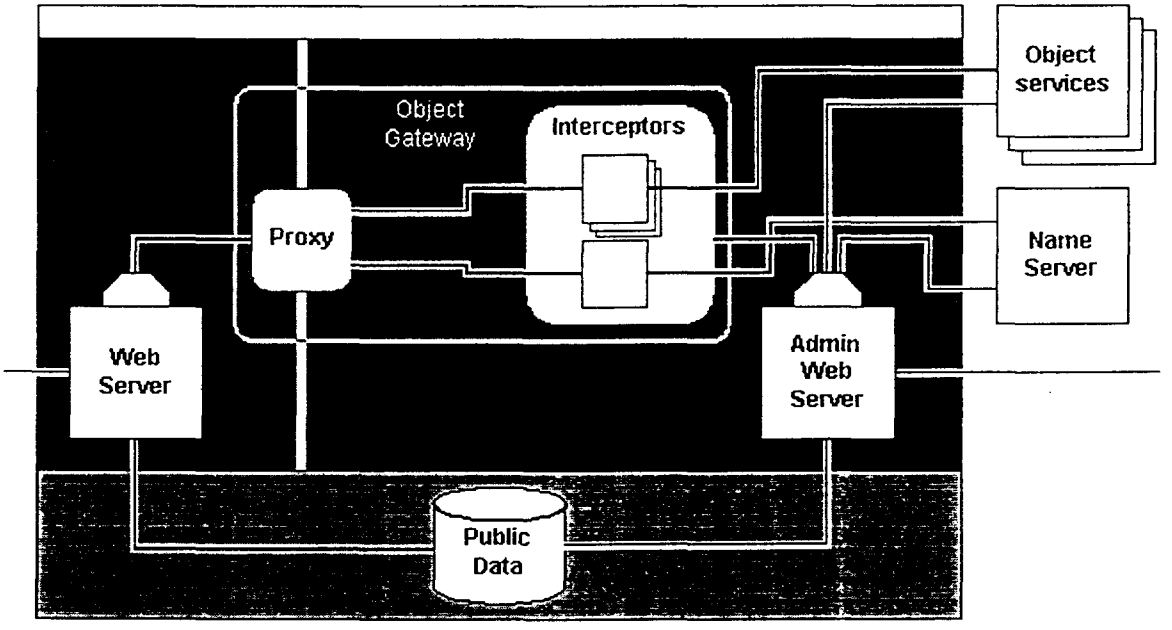


Figure 5: Object Gateway on VirtualVault

In order to ensure that the outside server plugin cannot access the real services, including the naming service, these services must not be exposed to unprivileged processes running at "SYSTEM OUTSIDE". This can be achieved by running the interceptors at "SYSTEM INSIDE", and exposing only the interceptor interfaces to the outside.

The interfaces are exposed through a proxy, which is the only part of the object gateway that needs to be privileged. This arrangement means that the interceptors run unprivileged at "SYSTEM INSIDE", and, in particular, the administration interface of the object gateway is exposed only at that level.

The outside web server runs a plugin that has been configured to forward all object service requests to the proxy. Responsibility for looking up the name in the naming service now rests with the proxy

rather than with the plugin. Since the proxy is configured to use the naming interceptor as its naming service, the only object references it will receive will be for the service interceptors that perform the authorization checks.

The proxy is currently implemented as a very simple ORBPlus server and client. Very little code is required to accept the incoming invocation, look up the required service, and invoke it. This is partly because a great deal of the work is done in the ORBPlus library code and in mechanically generated stub code.

The proxy is larger than the quantity of application code would suggest. This raises the question of whether or not it is appropriate to grant privileges to this proxy. Is it safe to trust the libraries and stub code? For the fullest confidence, a security inspection of the ORBPlus code will need to be conducted.

For most purposes, sufficient confidence can be gained by observing that the proxy performs one simple and well-defined task, and with such simplicity, there is little scope for unexplored paths in the code where security holes may lurk. Also, the libraries are designed to deal with generic requests involving arbitrary quantities of incoming data, so the common problem of overrunning fixed limits is unlikely to occur.

Authorization Model

The object gateway controls access to the services in two ways: the naming interceptor controls whether or not a service is available to any external clients, the service interceptor for each service then imposes an authorization check for each invocation of the service. The service interceptor invokes an authorization service through a well-defined interface; this means that the interceptor can be configured to use whatever specific authorization mechanism has been chosen to match the system requirements.

In a distributed object system, the service - the interface exposed by an object - is a natural unit for granting access to resources. The object references that may be passed as parameters in CORBA-based systems point to these services. It is up to the system designer to choose what operations are offered in the interfaces of the objects. Objects, or collections of objects, can be designed so as to offer whatever sets of operations are meaningful for the application. Given this flexibility, there is no advantage in controlling access at the level of individual operations.

One of the key elements of the commercial security policy advocated by Clark and Wilson [CW 87] is that all manipulations should be performed by "well-formed transactions", with users granted rights to perform specific transactions, rather than having read and write access to the underlying data items. The data abstraction and encapsulation of object based systems corresponds to the "well-formed transactions", and the right to perform a transaction corresponds to the right to invoke an operation in an interface.

The authorization check must be performed when the service is invoked. In the generic gateway described below, it will be possible for object references to be retained and passed around among clients. It will not be sufficient to check authorization when the name is looked up; the reference may be reused in a different session, perhaps by a different user. The right to look up the name is distinct from the right to invoke the named service. Each needs to be authorized separately.

If a finer granularity of access control is required, it should be provided in the business logic implemented in the application. If rights depend upon parameter values, they should be enforced where those values are interpreted. The significance of the parameter values is a matter for the application that implements the business logic, rather than a generic security mechanism. The security mechanisms can give a guarantee of identity - they can identify the user who initiated the

session - but the security mechanism is not the appropriate place to implement the business logic that sets limits on the range of parameters for the user.

The application designer may choose to use a sophisticated authorization service, such as the Praesidium Authorization Server [HP 96b], to implement value based checks, but this is not imposed by our architecture.

Generic IIOP Gateways

The use of Java to implement CORBA means that it is possible to download a Java applet which then uses IIOP to talk to services inside the internal (corporate) network. The secure object gateway described above can be generalized to support this; the modified architecture is shown in Figure 6 below.

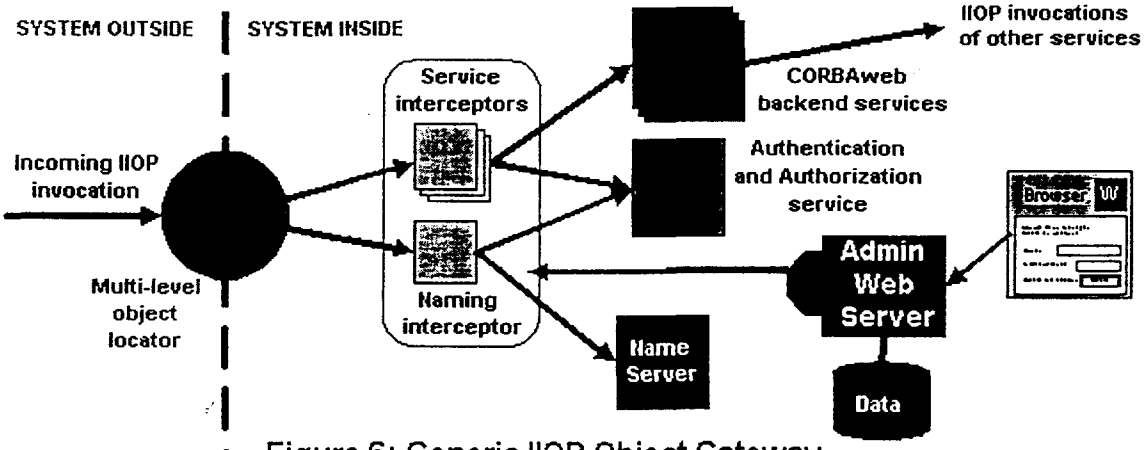


Figure 6: Generic IIOP Object Gateway

The service interceptors are generalized: for CORBAweb they only needed to intercept calls to services of type "CGI"; now they must intercept calls to any service type by using the Dynamic Invocation Interface (DII) and Dynamic Skeleton Interface (DSI) (see [OMG 95]).

The proxy is replaced by a multi-level object locator. This acts an object locator for requests coming from "inside", and as a proxy for requests coming from "outside". This component has the privileges needed to act as a multi-level server, able to accept connections from clients at more than one level. It is configured so that requests from outside clients are always forwarded to one of the interceptors, ensuring that the authorization check cannot be avoided.

The Multi-level Object Locator

The Multi-level Object Locator exploits the flexibility in the IIOP specification (in [OMG 95]) that provides for several different ways to ensure that requests reach the service for which they are intended. A request is sent to a specific host and port, but also contains an object key that uniquely identifies the target object. The object that receives the request may be the target object, it may be a proxy that invokes the target object, or it may respond with a "location forward" instructing the client to re-send the request to a new destination. The re-send is done by the infrastructure at the client, and is not visible to the application.

For requests from "inside" clients, the Multi-level Object Locator acts like the standard HP ORB Plus object locator. It keeps track of persistent objects using their object keys. Whenever an object starts, its underlying infrastructure informs the object locator of the host and port at which it can be called. When the Object Locator receives a request from an IIOP client it responds with a

"location_forward" informing the client of the current location of the target object. The client then interacts directly with the target object, and the object locator takes no further part in the interaction.

For requests from "outside" clients, the Multi-level Object Locator acts as a proxy. It checks that the target object is an interceptor, and, if it is, uses its privileges to forward the request at the "SYSTEM INSIDE" level. If the target object is not an interceptor, an "invalid object reference" exception is returned to the client.

Confidentiality and integrity between external clients and the object locator is achieved by running IIOP over SSL. The access control mechanisms are identical to those used for the CORBAweb Object Gateway. At the time of writing we have not found a technique for the applet to reuse the browser's cookie (and hence the session ID), so users must re-authenticate themselves when the applet starts to run. The session ID is passed as a "principal" parameter [OMG 95].

Object references returned to the client applet as the the result of an invocation will not be usable by the client unless they are references to service interceptors. This prevents services in the internal network accidently subverting security by handing object references to a client in the outside network. The only object references which are usable by these clients are those which are obtained from the naming service interceptor. For the naming interceptor service to make them available an administrator must have explicitly set up access controls and authorized the service to be available (as part of the action of creating the interceptor).

The consequence of this is that any server handing an object reference to an external client needs to obtain that object reference from the naming interceptor.

Issues

The above does not address client side firewall issues: how does the applet trying to use IIOP make a connection through its firewall? There are at least two possible approaches:

- Run a dedicated IIOP proxy in the client firewall.
- Tunnel IIOP through an existing proxy server (e.g. an HTTP proxy)

A dedicated proxy (or application level gateway [CB 94]) offers the maximum potential for security. It understands and exploits the semantics of IIOP to restrict communication to that which has been approved by the administrator who configures the proxy. Examples of restrictions which could be enforced by such a proxy would be: to restrict the sites to which IIOP invocations can be made; to restrict which users can make these invocations.

The secure object gateway described above could be run in reverse as a dedicated client proxy. This would prevent users invoking services which had not been previously authorized by an administrator - some may consider this too restrictive!

One problem which would need to be solved for any dedicated proxy is: how to map the object references embedded in the applet to point to the client proxy, rather than the one at the server? Some options are discussed below.

- The applet asks the user for the host and port number of the IIOP proxy
- Modify the web browser so the IIOP proxy is set as part of its configuration
- Assume the IIOP proxy is the same as the HTTP proxy
- Assume the IIOP proxy is on the same host as the HTTP proxy, but at a different, well known port.

Tunneling through an existing (HTTP) proxy offers less security, unless that proxy also understands

IIOP: it turns the proxy into a circuit level gateway [CB 94] which is not able to use the semantics of IIOP to restrict communication. This has the advantage of working with existing firewall technology (many HTTP proxies are already deployed), but one might also argue that it punches a hole in it. This hole is even larger if the proxy needs to allow applets to act as servers. A full Java implementation of CORBA allows applets to provide services, so an applet might provide a callback interface.

Conclusions

CMW exploits the concepts of least privilege to protect against root attacks, and data partitioning to protect sensitive data. In addition we exploit distributed object technology, so that services never have to be installed on the gateway machine. Management of the object gateways is automated via an HP CORBAweb interface to minimize the chances of misconfiguration which could compromise the security of the gateway machine.

The secure object gateways described in this paper try to strike a balance between a bespoke application level gateway and a circuit level gateway. They are generic in that we only ever instantiate one type of gateway to make an application available. They are bespoke in that we instantiate one gateway for each separate back-end service and can then configure the behaviour of that gateway to say who is authorized to access the service.

Application level gateways [CB 94] offer the most security, because they understand the semantics of what the gateway user is trying to do and can therefore provide detailed security checks. Circuit level gateways are generic, but offer less security because they cannot understand what the user is trying to do and so can only provide very general checks. One approach to making legacy systems available via WWW is to write an application level gateway for each application. This would be costly and also bugs in the application level gateway itself may well compromise security of the gateway machine and the application.

The guiding principle for the architecture is that the service (the interface exposed by the object) is the unit for granting access to resources. What precisely constitutes a service is entirely under the control of the system designer. The object reference (or service handle) may be to a service which wraps an entire database, or it may be to a wrapper for a field in a file. This supports both fine and coarse grain access control, depending on the unit of service provision. Very fine grain authorization such as parameter range checking cannot be supported, as it is difficult to extract such information in the object gateway. This is more easily supported in business logic residing in the service or service wrapper.

The results of this paper also demonstrate that the standard features of IIOP [OMG 95] adequately support the incorporation of IIOP proxies into firewalls. In particular the multi-level object locator can act as a proxy for invocations received from external clients, but but can instruct internal clients to use the service interceptor directly (for access control) so that the object locator does not partake directly in the interaction.

A number of issues have been identified for client side proxies which require further investigation.

Acknowlegements

The authors would like to thank their colleagues Chris R. Dalton, Dirk Kuhlmann and Dave Clarke for valuable feedback on this work.

References

[BS 95] "Migrating Legacy Systems: gateways interfaces & the incremental approach", M.L. Brodie

& M. Stonebraker, Morgan Kaufmann, 1995.

[CB 94] "Firewalls and Internet Security - Repelling the Wily Hacker", W.R. Cheswick, S.M. Bellovin, Addison-Wesley, 1994.

[CW 87] "A Comparison of Commercial and Military Computer Security Policies", D.D. Clark, D.R. Wilson, Proceedings of the 1987 IEEE Symposium on Security and Privacy, pp 184-194, IEEE 1987.

[DG 97] "Applying Military Grade Security to the Internet", C.I. Dalton and J.F. Griffin, submitted to 8th Joint European Networking Conference (JENC8), Edinburgh, Scotland, 12-15 May 1997.

[DIA 91] "Compartmented Mode Workstation Evaluation Criteria VERSION 1 (Final)", J.P.L. Woodward, DDS-2600-6243-91, 1991.

[DOD 85] "DOD Trusted Computer System Evaluation Criteria", Department of Defense Standard, DoD 5200.28-STD, December 1985; http://www.disa.mil/MLS/info/orange/index.html

[FKK 96] "The SSL protocol Version 3.0, A.O. Freier, P. Karlton and P.C. Kocher, http://home.netscape.com/eng/ssl3/ssl-toc.html

[HP 96a] "HP ORB Plus", Hewlett Packard, June 1996.

[HP 96b] "HP Praesidium/Authorization Server", Hewlett Packard, June 1996.

[HP 96c] "HP Praesidium/VirtualVault Internet Security Solution", Hewlett Packard, June 1996.

[OMG 95] "The Common Object Request Broker: Architecture and Specification", Revision 2.0, July 1995, The Object Management Group.