Implementing Policy-based Content Filtering for Web Servers

Tony White[1], Eugen Bacic[2]
[1]School of Computer Science, Carleton University
[2]Cinnabar Networks
{arpwhite@scs.carleton.ca, ebacic@cinnabar.ca}

Abstract: *Web servers dominate our view of the Web today. Security provided by them has been implemented with varying degrees of success. Web servers are frequently successfully attacked, with subsequent loss of corporate loss of face or revenue. Recent legislation has increased the importance of ensuring that only approved users gain access to information, which often implies filtering content served by applications. While content filtering can be implemented at the application level, this paper describes an innovative architecture for policy-based filtering that can be integrated with existing web applications.*

Keywords: web server, policy, content filtering

1. Introduction

Servers dominate the Web today. We rely on search engines, meta-search engines, portals and a wide range of other services hosted off of web servers accessed using HTTP or its secure variant. Business-to-business (B2B) interactions involve web servers and other modes of access. In a time when knowledge and information are increasingly the measurable assets of a corporation, information security is becoming more and more important. Recent legislation concerning the privacy of health care records (HIPAA) has increased the importance of secure web-based information access. While access and content control has been addressed in a number of ways, solutions have been implemented on a product-by-product basis. A consistent solution for access and content control has yet to be implemented for web servers, although applicable criteria and models exist [1], [2], [3], [4]. Heterogeneous implementation of access and content control has lead to incoherent security solutions being placed in service, access control being determined fully, or in part, by the path through which information is accessed. Here, we mean access control to be the decision to process a given HTTP request. By content control we mean the filtering of information generated by a web application based upon the identity of the user and state of a workflow process.

Even within the 3-tier web application architecture that is most commonly employed, where web, application and database servers have been combined, access control has been built into all three components. Clearly, when distributing the security responsibility across multiple components, creating a consistent view of security is difficult using such architectural schemes. The weakest link is not always obvious. For example, a user with web and database access might find that their ability to access information varies depending upon whether they access the information in the database directly or via the Web. This could easily occur if access control is not harmonized between the database and Web applications that retrieve and process the data. Clearly it is very

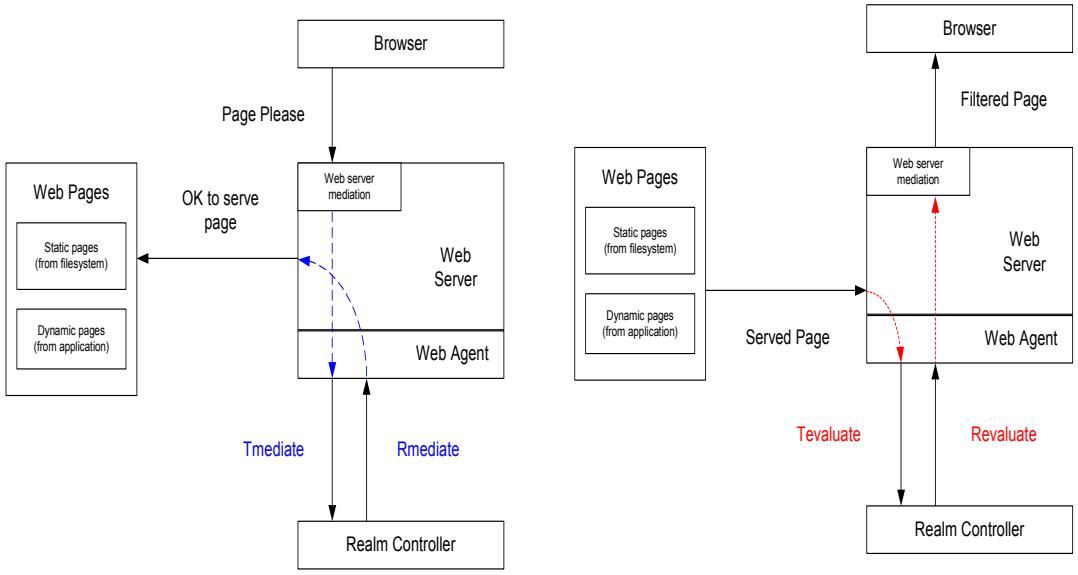

Figure 1: Web Server Content Filtering

difficult to decide what content should be provided to an end user if filtering takes place at multiple points in the n-tier business application as no single component has a view of the entire workflow process. This motivates the design of the centralized, web-based content filtering solution described in this paper.

2. Web Server with SecureRealms

The SecureRealms architecture was introduced in [8]. The essential characteristics of the architecture are that all security objects are represented as entities, and all entities have associated security policies that are stored in a repository called the Virtual Resource Attribute Database (VRAD). The security policies associated with entities are evaluated using a Generic Policy Engine [10] built into the Realm Controller. Mediation of access to resources is achieved by the evaluation of the security policy associated with the resource requested in the context of the resource access. For example, in the case of a web page access, the two entities involved are the user requesting the page and the page itself. The context of the access would include the Apache permissions associated with the page or directory.

SecureRealms defines a small, functional, security-aware meta-language, called Idyllic [8], which is a Policy Meta Language. This language is capable of codifying any business rule and resembles LISP, which has well known properties [7]. It is based on s-expressions, which are becoming an important component of XML as X-expressions (XEXPR) [5]. There is a straightforward mapping from XML's XEXPR to Idyllic's s-expression. It should also be noted that authorization capabilities for XML are only now emerging, with the SAML specification [6] still under discussion. SAML represents a familiar access control solution for authorization; one we feel will prove insufficient for the dynamic security needs of c-commerce.

The Web server instantiation of the SecureRealms architecture (hereafter referred to as SR-Web) operates by introducing a plug-in to the web server that interrupts the usual flow of content delivery. More specifically, we wait for the server to decide if a page can be served, and then we use the services of the Realm Controller to determine if the web server should still be allowed to serve the page. Once page content is available, we intercept it so that we can perform sub-page level filtering before giving the page

back to the server for delivery to the requesting browser, or application.

The mediation flow of control is shown in Figure 1. A browser requests a page ('Page Please'). If the web server determines (during web server mediation) that the page request may be honored, the web agent then gains access to the request. The request is packaged into a *Tmediate* message for transmission to the Realm Controller. The Realm Controller responds with an *Rmediate* message. If the mediation indicates that the request is allowable, the web agent returns control to the web server with an indication that the request may continue. The web server then causes the page to be returned (static) or generated (dynamic). The interaction between the Web Server and the remaining layers of the web application are unchanged.

Once the requested page content is available, the web server delivers the page to the Web Agent. If filtering is enabled, the web agent scans the page contents for special tags. If any are found, the information is packaged up into a *Tevaluate* message for delivery to the Realm Controller. The Realm Controller evaluates the incoming data and returns information to the Web Agent that will allow it to filter the served page. Once the page is filtered, it is passed back to the web server for final delivery to the requesting browser.

2.1 The Web Agent

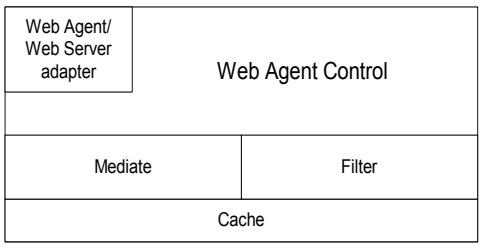

Figure 2: Web Agent Architecture

The Web Agent is constructed to maximize the amount of server independent code. It does this by providing an adapter component that hides the details of the server implementation. The following sections discuss the adapter and filter modules shown in Figure 2.

2.1.1 Web Server – Web Agent Adapter

A web server goes through a number of steps between receiving a page request from a browser and returning the page for viewing. Web servers have been designed so that external programs can bind to the web server at any or all of these points to replace or augment the server's normal behaviour. While the processing stages are similar for the web servers we have investigated (Apache, IIS), the actual method of binding to the server is different for each server. The Web Agent therefore has two server specific pieces of code (the adapters) that bind the common portion of the Web Agent to the server.

In practice, the adapters set callbacks that the server will use to notify that a server processing stage is complete. The callback routine will get session and transaction parameters directly through the callback, or separately as globally available server environment variables. For example, one server environment variable records the user name associated with the current page request, another records the fully remapped page path, another the server version, and so on.

2.1.1.1 Apache

Under Apache, server extensions are called modules. Modules are usually compiled into the server, but can also be linked in dynamically. We implemented the dynamic link approach. The Apache web agent must be multithreaded to work correctly with the server. Apache performs the following functions when processing a page request:

1. URL -> Filename translation
2. Authentication ID checking [is the user who they say they are?]
3. Authentication access checking [is the user authorized *here*?]
4. Access checking other than authentication

5. Determining MIME type of the object requested
6. `Fixups' --- there aren't any of these yet.
7. Actually sending a response back to the client.
8. Logging the request

Steps 1-4 are where Apache resolves the information necessary to perform mediation and carries out its own mediation. In step 5 Apache determines what sort of handler should be used to process the page request. Step 7 is where the server actually either retrieves a static page or invokes an application to create a dynamic page.

For us to do mediation, we need access to fully resolved request information at a point where we can stop further request processing. Step 6 gives us this access point – here we have access to full pathnames and authenticated user names and we can instruct Apache to abort the request depending on the results of our mediation.

On the filtering side, we need to capture the output of step 7. Unfortunately, the modules doing the work in step 7 return data directly to the web server without offering us a glimpse at it on the way past. To resolve this, the Apache adapter wraps the step 7 modules in a handler of our own that itself invokes the original step 7 modules in a way that allows us access to returned data.

2.1.1.2 IIS

Under IIS, server extensions are called either extensions or filters, depending on what functionality they implement. For the Web Agent, we will be creating a filter to gain the most complete access to the server. IIS filters are built as DLLs. The IIS web agent must be multithreaded to work correctly with the server.

The IIS adapter has to do the same sort of things as the Apache adapter – allow us to catch resolved user name and file path information for mediation purposes and then allow us to intercept output prior to delivery to the requesting browser. The IIS API offers us the following hooks:

1. SF_NOTIFY_READ_RAW_DATA: When a client sends a request, one or more SF_NOTIFY_READ_RAW_DATA notifications will occur.
2. SF_NOTIFY_PREPROC_HEADERS: This notification indicates that the server has completed pre-processing of the headers associated with the request, but has not yet begun to process the information contained within the headers.
3. SF_NOTIFY_URL_MAP: An SF_NOTIFY_URL_MAP notification occurs whenever the server is converting a URL into a physical path.
4. SF_NOTIFY_AUTHENTICATION: An SF_NOTIFY_AUTHENTICATION notification occurs just before IIS attempts to authenticate the client.
5. SF_NOTIFY_AUTH_COMPLETE: This notification fires after the client's identity has been negotiated with the client.
6. SF_NOTIFY_READ_RAW_DATA: As mentioned in step 1, if the client has more data to send, one or more SF_NOTIFY_READ_RAW_DATA notifications will occur here.
7. At this point in the request, IIS will begin to process the substance of the request. This may be done by an ISAPI extension, a CGI application, a script engine (such as ASP, PERL, and so on), or by IIS itself for static files.
8. SF_NOTIFY_SEND_RESPONSE: The SF_NOTIFY_SEND_RESPONSE event occurs after the request is processed and before headers are sent back to the client.
9. SF_NOTIFY_SEND_RAW_DATA: As the request handler returns data to the client, one or more SF_NOTIFY_SEND_RAW_DATA notifications will occur.
10. SF_NOTIFY_END_OF_REQUEST: At the end of each request, the SF_NOTIFY_END_OF_REQUEST notification occurs.
11. SF_NOTIFY_LOG: After the HTTP request has been completed, the SF_NOTIFY_LOG notification occurs just before IIS writes the request to the IIS log.
12. SF_NOTIFY_END_OF_NET_SESSION: When the connection between the client and server is closed, the

SF_NOTIFY_END_OF_NET_SESSION notification occurs.

Our mediation is triggered by the SF_NOTIFY_AUTH_COMPLETE in step 5. Unlike Apache, IIS does offer us a look at returned data. Filtering then is carried out in response to the SF_NOTIFY_SEND_RAW_DATA event (or events) by passing this data to the Web Agent for possible modification. The flow of control shown corresponds to IIS V5. IIS V4 has a slightly smaller set of hooks and requires a slightly different flow of control. The principles, however, are the same for both versions.

2.1.2 Adapter – Web Agent interface

The previous two sections have documented where and how the adapter portion of the Web Agent hooks into the web servers. To ensure that the Web Agent code is common to all web servers, the adapters present a common interface between the web server and the Web Agent. The interface is implemented as a set of 5 callbacks instantiated in the Web Agent code. The callbacks are:

- Boolean canAccessPage (authenticatedUserName, filePath)
- Boolean filterOn (filePath)
- Boolean filterStart (authenticatedUserName)
- Integer filterData (authenticatedUserName, pageContent, size)
- Boolean filterEnd (authenticatedUserName)

The first callback invokes the mediation portion of the Web Agent and its outcome determines whether the adapter will allow the web server to continue processing or not. The remaining four callbacks relate to Web Agent filtering. The first (filterOn) allows the adapter to determine if Web Agent filtering is enabled. If filtering is disabled, the adapter can speed page processing by not passing page data to the Web Agent. The 'filterStart' and 'filterEnd' callbacks allow the Web Agent to do any page setup and teardown activities that may be necessary. The 'filterData' callback may be invoked multiple times to pass page data to the Web Agent for filtering.

2.1.3 Page Request Mediation

If SR-Web determines that a user cannot access a page, the page that will be returned to the requesting user will be identical to one the web server would have returned if it had blocked the page access. This has the advantage of making the authorization engine transparent to the user, identical errors being returned with or without SR-Web.

SR-Web mediation occurs after any mediation done by the web server. At that point, the adapter will invoke the 'canAccessPage' callback with the authenticated user name and the requested file name.

2.1.4 Page Content Filtering

Filtering of web pages is a significant new security function. A filtered page is one that may have had portions of the page content removed as determined by the privileges of the requesting user and the policies attached to that portion of the page. A page to be filtered, whether it is a static or dynamic web page, must properly enclose the block that is to be filtered in a pair of 'srf' start and end tags. An example is shown below, with the content filtering tags shown in bold text. The 'srf' start tag must have a 'filter' attribute. The value of the filter attribute is one or more name/value pairs. The attribute name corresponds to an entity called a filter entity. A web page, modified to allow for sub-page level filtering now looks like:

```
<html>
<head>
<meta name="srf" content="FilterEntityA,
FilterEntityB">
</head>
<body>
<p>Some text.</p>
<srf filter="(FilterEntityA userid),
(FilterEntityB userid)">
<p>Text to be secured.</p>
```

</srf>
</body>
</html>

As the Web Agent filters the original page, the second paragraph (and its enclosing 'srf' tag) may be removed from the final output if the Web Agent, working with the Realm Controller, determines that the requesting user cannot access the material.

2.1.4.1 Filter Entities

Filter entities are ways of naming content that share similar characteristics. The characteristics are identified by end-user analysis of web page data. A filter entity is a regular VRAD entity to which a policy may be attached. The indirection allows different policies to be attached to a filter entity (and by extension to a fragment of a web page) without having to alter the source page data. Filter entities are explicitly created by management activity.

2.1.4.2 Filter Operation

Filtering proceeds in several steps. For each 'srf' tag the filter entity names and requesting user name are bundled into a Plan Nein *Tevaluate* message for transmission to the Realm Controller. The Realm Controller evaluates the policies bound to each filter entity in the context of the user name and packages the results into an *Revaluate* for return to the Web Agent.

The filtering software then matches the values returned for each filter entity with the required value from the page. If all the returned filter entity values evaluate to true, the secured content will be passed on to the requesting user, otherwise it will disappear from the output.

Should a web page contain a filter entity name that does not have a corresponding entity in the VRAD, a false value will be returned to the Web Agent for that filter entity. This will have the effect of suppressing the affected data.

Filtering is potentially an expensive operation, as every output page has to be checked for the presence of 'srf' tags. To increase performance, a 'srf' meta-tag must be present in the head portion of a web page. The contents of the meta-tag will be a list of all filter entities referenced in the page body. The filter entity names will be bundled up with the requesting user name to be sent to the Realm Controller via the Plan Nein *Tevaluate* message. This can be done even before the balance of the web page is available. If no filtering meta-tags are found in the page head, the rest of the page does not have to be screened.

3. Future Work

Policies have begun to be of greater and greater interest. The recent work on the eXtensible Access Control Markup Language (XACML) at OASIS [9] is a case in point. The SecureRealms architecture and Idyllic in particular were an outgrowth of years of R&D effort. Idyllic was designed to be syntactically correct and provable via denotational logic [10] and reflected existing technologies of the time.

XACML specifies a "subject-target-action-condition" oriented policy for XML documents. A subject is a unique identity, group, or role while a target is what is typically referred to as a resource or object. XACML includes conditional authorization policies, as well as policies with external post-conditions to specify actions that must be executed prior to permitting access.

With XACML being both an access control policy language and a request/response language it appears similar in scope and intent to Idyllic. Hence, the XACML policy language is used to express access control policies while the request/response language expresses queries as to whether a particular access request should be allowed and provides the appropriate response.

For example, in the case where a subject wants to take some action on a particular object, or resource, the subject submits its query to the component protecting the resource (e.g., file system, web server). This component is called a Policy Enforcement Point (PEP). The PEP forms a request (using the request language) based on the attributes of the subject, action, resource, and any other relevant information. The PEP then sends this request to a Policy Decision Point

(PDP), which examines the request, retrieves the relevant policies, and determines whether access should be granted. That answer (expressed in the response language) is returned to the PEP, which can then allow or deny access.

With XML becoming a lingua franca for communication of logic between disparate components it only stands to reason that efforts should be made to see whether or not the lessons learned from SecureRealms, Idyllic, and the Realm Controller can be migrated to a full XML-based implementation.

4. Conclusion

As computer networks grow, security is becoming more of a concern with each passing day. Organizations view and relate to information differently and have differing requirements for the protection, dissemination, and modification of their resources. There are now important legal considerations in granting individuals access to information. Content filtering as well as access management become important considerations when designing a web-based information system.

To date, many organizations have met their security concerns by implementing access prevention mechanisms such as firewalls, cryptography, and virtual private networks. General access to host systems is provided based on the premise that once authenticated, users can be given full freedom to perform their duties. Existing security products protect only the perimeter creating islands of security and although each performs their individual tasks very well, interoperability and workflow-related issues constantly arise. Solutions to the interoperability problems include special servers accessible to external partners and the use of web servers to store restricted views of information. Such duplication of information often leads to errors due to inconsistency and is expensive to maintain.

This paper has described an architecture where content can be modified after creation, based upon policy-based filtering. Legacy n-tier web applications require minimal modification in order to take advantage of the enhanced security – new srf tags need only be added to page content. Content filtering and access control is delegated to a centralized security server that is capable of understanding workflow. Security, independent of access path, is clearly provided by the design. We believe that it represents a significant step forward in providing technology-independent authorization.

References

[1] Department of Defense Trusted Computer System Evaluation Criteria. DoD 5200.28-STD, December 1985.

[2] Communications Security Establishment, *The Canadian Trusted Computer Product Evaluation Criteria.* Version 3.0e, January 1993. The Communications Security Establishment, Government of Canada.

[3] *Information Technology Security Evaluation Criteria*. Harmonised Criteria of France - Germany - the Netherlands - the United Kingdom. Version 1, May 2, 1990.

[4] Bell, David E. and L.J. LaPadula. *Secure Computer Systems: Mathematical Foundations,* ESD-TR-73-278, Volumes I, II, and III. The MITRE Corporation, March, May, and December 1973.

[5] http://www.w3.org/TR/xexpr/

[6] See SAML references on http://www.oasis-open.org/committees/security/

[7] Lee, P. and Pleban, U.F., "On the Use of LISP in Implementing Denotational Semantics", *Proceedings of 1986 ACM Conference on LISP and Functional Programming,* Cambridge, Mass., 1986. pp. 233 - 248.

[8] White T. and Bacic E. Authorization as a Service provided by a Generic Policy Engine. In Proceedings of the 2002 International Conference on Security and Management, Las Vegas, June 24-27 2002.

[9] http://www.oasis-open.org/committees/xacml/.

[10] Bacic, E. *The Generic Policy Engine*. Master of Computer Science Thesis, Carleton University, May 1998.