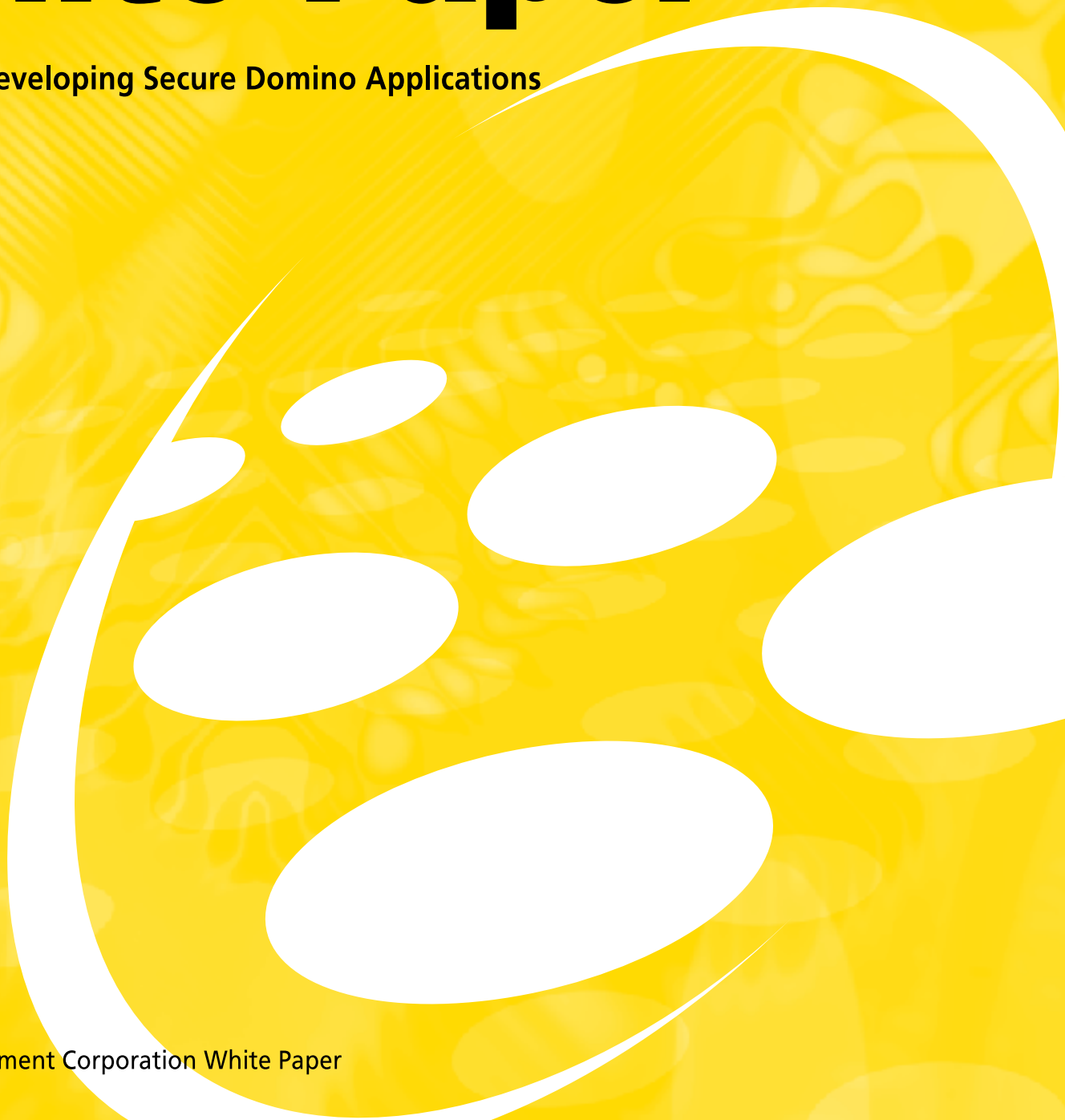
Lotus

White Paper

A Guide to Developing Secure Domino Applications

December 1999



A Lotus Development Corporation White Paper

Contents

Introduction

As the popularity of Lotus® Domino™ as a Web application development platform grows, we can expect to see an increased awareness about techniques for properly implementing the Domino Security Model. Domino contains a vast array of design features and capabilities that allow developers to create innovative Domino Web applications. However, if security is an objective, then the features that are used must be selected specifically to provide the desired security. The only way to ensure a secure Domino Web application is to adhere to the Domino Security Model during the development process. Controlling access to documents using field-based access controls is necessary for every Domino Web application.

This paper discusses some advanced Domino development techniques, and therefore assumes that readers have a solid working knowledge of Domino application development. Each technique described includes an explanation of the Domino Security and deployment implications as well as a recommendation of the proper development method to ensure security.

The main topics covered in this paper include secure Domino Web application design, database access control lists, view level security and document level security. For a good understanding of these topics, read the following titles, available at http://www.notes.net/today.nsf unless stated otherwise, before proceeding with this paper:

- *Lotus Notes® and Domino R5.0 Security Infrastructure Revealed (http://www.lotus.com/redbooks)*
- *Designing a Secure Domino Application*
- *Securing Your Application*
- *The ABC's of Using the ACL*
- *Filtering Data for Domino Web Users*
- *Technique #3: Readers Fields (sidebar)*

Database Web Launch Properties

Background

The Database Web Launch properties enable a developer to specify what will be displayed when a Web user tries to open a database. Typically, the application's main home page is displayed. This property executes the launch option specified when Domino receives a URL request to open the database.

Implication

This property does not prevent users from reconstructing the URL and accessing a list of non-hidden views in the database. Be aware of the following Domino URL actions that may be used:

http://host/database.nsf/$DefaultNav?OpenNavigator

http://host/database.nsf/?OpenDatabase&3PaneUI

Note: There is no current way to prevent Web users from using the Domino URL action /$DefaultNav?OpenNavigator.

Recommendation

Use the following methods to prevent view names from being displayed to the Web browser when either of the above two Domino URL actions are used.

- Hide the view by surrounding the name of the view with parentheses. **Note:** Just hiding a view name does not ensure that documents within this view are secure. In fact, they are not secure unless document access control (Readers and Authors names fields) lists and various hide-when techniques have been implemented. For example, if a user guesses the name of a hidden view, such as "(All)," they will be able to access the non-secure documents within that view.
- Maintain a view access control list. This method will allow user access only to a controlled list of users. Consider using Roles and Groups in the view access control list for ease of maintenance.
- Apply the Design Document property "Hide design element from Web browsers." Be aware that this method will prevent all Web user access to this view and to the documents referenced within this view.

Tip: The /?OpenDatabase&3PaneUI Domino URL action (e.g. http://host/database.nsf/?OpenDatabase&3PaneUI) can be prevented by adding DominoEnable3PaneUI=0 in the Domino server notes.ini file. Prior to R4.6, this notes.ini setting was not available. From R4.6 and above, this setting is on by default.

View Templates

Background

When views are displayed on the Web, they may not have as many features as in the Notes™ client. For example, by default, a view will display as a full screen with default navigation buttons on the Web. To gain more control on the Web, it is possible to embed a view in a form which allows developers to add features that will maintain the same functionality available in Notes applications, as well as control the size and appearance of a view display. For a form to display an embedded view, the form name must conform to certain rules which specify reserved form names. These forms are called *view templates*.

The following are the reserved form names for views:

- $$ViewTemplate for <viewname>: Where <viewname> is the name of the view with which this form is to be associated. Note that view alias names must be used in place of <viewname> if one exists.
- $$ViewTemplateDefault: A form of this name with an embedded view or $$ViewBody field will be the template for all Web views that aren't associated with another form.

Both of these forms also require an embedded view (R4.6.x and higher) or a $$ViewBody field for the view to be displayed in the form. There are many examples of uses of view templates in the templates that ship with Domino. One example is in the R4.6 Discussion database template (discsw46.ntf). This template contains multiple view templates including $$ViewTemplate for By Category.

When a HTTP request is made to access a Domino view (e.g., http://host/database.nsf/viewname?openview) Domino first looks for a reserved form name to associate with the view. If no reserved form name is found, Domino serves the view up in a default format.

For example, if a user clicks on a URL to open the "By Category" view and the Domino application did not contain either a form called "$$ViewTemplate for By Category" or "$$ViewTemplateDefault," Domino would serve the view in default format.

Tip: Notes Pages may be used to create view templates in Release 5.x.

One Domino development technique exists whereby the required $$ViewBody field or embedded view is omitted from the view template form. This results in no view being displayed at runtime when the view is referenced. Generally, these empty view templates contain simple text messages such as "Access Denied" or "No Documents Found" to prevent users from accessing the view contents. This technique is an excellent deterrent, and it is recommended to be included in Domino Web applications, but is not considered part of the Domino Security Model to protect sensitive data. View templates were made available in Domino to bring additional functionality to the Web, not security.

Implication

With any prior knowledge of the design of the Domino application, a user may potentially obtain unwanted access to the document level. Thus documents without access controls in them are left exposed.

Recommendation

To cut off access to documents and prevent the user from bypassing the empty view template technique, developers need to restrict access to documents by using field-based access controls. Documents containing confidential and sensitive information need to take advantage of field-based access controls (Reader and Author fields) in conjunction with database and view user access controls. For example, one technique might be to give users access only to data that is specific to them. If a Web order form contains users' personal information, such as credit card numbers and expiration dates, this sensitive data could be placed on a computed subform which is displayed only when the document is new for data input purposes. Access to the subform would be limited to those listed in a Readers names field, that is, the user who authored it and those users who will process it.

Hidden Views

Background

The technique of using views is common in Domino Web applications because such views provide programmatic access to documents that would not otherwise be available in an application. Hidden views have their names surrounded by parentheses. For example, a view named “(All)” is considered hidden. Hidden views are not available via the View-Go To… menu in the Notes client, nor do they appear in the list of views presented when the Domino URL syntax is reconstructed using /$DefaultNav?OpenNavigator.

Note: The following Domino URL syntax will display all non-hidden views to the Web browser.

http://host/database.nsf/$DefaultNav?OpenNavigator

Implication

Hidden views are not considered part of the Domino Security model and may result in users accessing sensitive data. Access may be gained to the document level with prior knowledge of the design of the Domino application. An example might be that of a user who guesses that you have developed your site with a view name of (All) and then reconstructs the Domino URL to access documents in that view.

Recommendation

To cut off access to the document level and essentially prevent the user from bypassing the hidden view deterrent, ensure that access to all documents containing sensitive data is restricted using field-based access controls in conjunction with database, view and document user access controls. By doing so, even if the view name is guessed correctly, Domino will prevent access to the documents containing the sensitive data.

Form Formulas

Background

Another misuse of a Domino feature in place of Domino security on the Web is form formulas. When a user tries to read or edit a document from a view, that view's form formula determines which form Domino will use in presenting the document to the user. It also controls which form the user will get when trying to compose a new document.

This formula is part of the view design element. For example, the following is a form formula that will display a document opened from a browser to display an "Access Denied" error.

```
@If(!@IsNewDoc ; "AccessDeniedForm" ; "OrderForm")
```

Once the document is opened, the Domino URL might look like this:

http://host/orders.nsf/d46f1520006b1d1e852567df006d89bd/16437726bc6934f6852567df006df2e8?OpenDocument

Tip: There are three possible Domino URLs to represent the same document URL:

1. http://host/database.nsf/viewname/UNID?action
2. http://host/database.nsf/viewID/UNID?action
3. http://host/database.nsf/0/UNID?action

Note: Any text string may be used to replace the 32-character view ID in a Domino document URL.

Implication

The third example above is valid and shortens the document URL by replacing the 32-character view ID with a zero. Generally, developers are not aware that by using this technique form formulas will not be executed and they will effectively be bypassed. This occurs because the form formula is part of the view design element and by replacing the view ID with a placeholder (in this case a zero), Domino just displays the document without going through the view, which would execute the form formula. In the above example, the URL has been reduced to the following:

http://host/orders.nsf/0/16437726bc6934f6852567df006df2e8?OpenDocument

Domino will proceed to serve the document up to the browser using the form name contained in the Form field stored in the document. This will potentially expose sensitive data, if the documents have not been properly secured by implementing field-based user access controls at the document level.

Recommendation

The correct technique is to use field-based access controls and hide-when formulas. Placing only the usernames, roles and group names in the Readers field will effectively prevent all unwanted access to sensitive information. This technique properly covers all malicious attempts at compromising your data.

Agents

Background

Securing your Domino Web applications is not all about data security. Agents are less likely to result in unwanted access to your data; however, they are prone to attack. Typically this results in a denial of service type attacks.

Any attack that prevents a Web client from being served by a Web server is termed a Denial of Service (DoS) attack. The attack could disable either a client workstation by taking up excessive CPU cycles, or by hogging memory or other critical resources. It could also bog down a server with large numbers of time- and CPU-intensive URL requests, making the server unable to respond to legitimate requests. Agents that are time- and CPU-intensive are prone to DoS attacks.

Implication

One implication of denial of service attacks is that an agent may be repeatedly invoked using the ?OpenAgent Domino URL syntax, a Domino server could be made to run several long agent threads, unnecessarily tying up resources needed for legitimate URL requests. Another is that an agent could be invoked outside of the scope of its original intent, possibly causing unwanted modifications to your data.

Users can discover agent names quite easily using the same technique described in the “Hidden Views” section above.

Recommendation

To protect your agents from potential DoS attacks, you need to prevent the bulk of the agent code from executing if the agent is being invoked maliciously. A typical practice is to establish prerequisites to run. These prerequisites, verified at the start of an agent, determine whether or not the agent is being invoked within an intended context. Only if the context is valid is the agent allowed to continue; otherwise it is terminated.

You want to be able to prevent users from using the ?OpenAgent Domino action by themselves. There is no need for any Web user to manually use this Domino URL action. Select your prerequisites by choosing conditions for the context that cannot be guessed or simulated easily. Described below are two methods to prevent Web users from maliciously invoking an agent:

Method 1: Ensure the referrer page is from the same Web server.

When a URL is entered directly in a browser, the referrer indicated by the HTTP_REFERRER Common Gateway Interface (CGI) variable will be blank. The following piece of LotusScript code at the start of the agent will prevent users from invoking the ?OpenAgent action URL directly.

```
Set session = New NotesSession

Set docContext = session.DocumentContext

REM ---------------------------------------------------------

REM  Don't allow invocation of this agent from just any HTTP

REM  Referrer. The referrer must be from our server.

REM ---------------------------------------------------------

If Not(Instr(1, Ucase(docContext.HTTP_Referer(0)),
Ucase(docContext.Server_Name(0))) > 0) And _

Not(Instr(1, Ucase(docContext.HTTP_Referer(0)),
Ucase(docContext.HTTP_HOST(0))) > 0) Then

   Print {<HTML><HEAD><TITLE>Error</TITLE></HEAD><BODY>}

   Print {<H1>Error</H1>Unauthorized Exception<P><HR>}

   Print {</BODY></HTML>}

   Exit Sub

End If
```

When a user tries to run the agent by directly entering an ?OpenAgent Domino URL action, it will result in an HTML page stating "Unauthorized Exception." The key result is that only a few lines of code are executed before the agent terminates and most importantly, there is no harm done to your data and Domino environment.

Method 2: Use a hidden computed for display field on a form to trigger an agent.

There is another technique to prevent invocation of agents from any HTTP client. This one uses the HTTP_REFERRER, SERVER_NAME and HTTP_HOST CGI variables. The idea behind this technique is to ensure that a hidden computed-for-display field computes to a predetermined value.

This method involves placing a hidden computed-for-display field on forms that trigger an agent. The value of the field should compute to a predetermined fixed value. The following code illustrates such a field, named "ValidContextProof," and its value computes to "domino." You can choose any field name and value.

When the agent is triggered, the document context should contain this hidden field and its value should be the predetermined fixed value. If this is not the case, the agent should terminate, since it has not been invoked using an intended context. When a URL is entered directly in a browser, it is very unlikely that the context will contain a field with that same name and value. The following LotusScript code placed at the start of the agent will ensure that the agent continues execution only if the context is valid.

```
Set session = New NotesSession

Set docContext = session.DocumentContext

REM ------------------------------------------------------

REM  Don't allow invocation of this agent from just any

REM  context. The intended context must have a field called

REM  ValidContextProof with a value of "domino".

REM ------------------------------------------------------

If (docContext.ValidContextProof(0) <> "domino") Then

     Print {<HTML><HEAD><TITLE>Error</TITLE></HEAD><BODY>}

     Print {<H1>Error</H1>Unauthorized Exception<P><HR>}

     Print {</BODY></HTML>}

     Exit Sub

End If
```

The field is hidden using text paragraph hide-when formulas (as opposed to being a hidden using the HIDDEN HTML attribute), so Web users cannot see it. It does not appear in the HTML source for the page. Also, a Web user with malicious intentions would have to first figure out that such a technique is being used, then guess the name of the field and additionally the value of the field. This is very unlikely.

When a user tries to run the agent by directly entering an ?OpenAgent Domino URL action, it results in the same page shown in the previous technique. Again, the key result is that only a few lines of code are executed and the agent terminates.

CGI Variables

Background

CGI is a standard for interfacing external applications with HTTP servers. When a Web user saves or opens a document, the Domino Web server uses CGI variables to collect information about the user, including the user's name, the browser, and the user's Internet Protocol (IP) address. To capture this information in a Web application, you have two options:

1. Use a field with the same name as CGI variables.
2. Use LotusScript agents.

For a complete listing of the CGI variables supported by Domino, refer to Notes Help (help4.nsf) or Domino 5 Designer Help (help5_designer.nsf).

Implication

Domino can capture CGI variables through a field or a LotusScript agent and also capture any CGI variable preceded by HTTP_ or HTTPS. For example, cookies are sent to the server by the browser as HTTP_COOKIE. However, there are ways to manipulate CGI variables on the client side and push them to any HTTP server. It cannot be assumed that CGI variables are secure or that they cannot be altered by Web users.

Recommendation

There are other, more secure alternatives that can be used in place of CGI variables. For example, instead of using the REMOTE_USER CGI variable, you can use the @UserName function instead, which is secure and cannot be tampered with.

Also, the @BrowserInfo function (new in Release 5) can be used to determine the capabilities of a Web client. This information was previously only available via CGI variables.

The bottom line is that you should not rely exlusively on CGI variables for protection in your security implementation.

Custom Authentication Using @Password

Background

A common reason for building a custom login function is the need to use an already existing user-unique entity such as employee number, social security number, etc., instead of the username.

Typical implementations involve having users register with the site. Registration requires the user to supply a username and a password. The entered password is then encoded or hashed using the @Password function. A document containing the username and the encoded password is stored for the authentication process. The actual password before encoding is not stored.

A login interface that prompts the user for the username and password is provided. The password entered is encoded using @Password and this encoded password is compared to that stored during the registration process. If encoded passwords match for a given user name, the user is allowed further access by being routed to another server or database.

Implication

The @Password function simply encodes a string. @Password is especially useful in an input translation formula to protect a user's password from being seen by others. There is no way to decode the original string once it has been encoded by @Password.

The encoding function is a one-to-one mapping; for example, a password of "lotusnotes" will always result in "(DE9CA9CD7BD212362B6D312A33E10FB2)" when passed through @Password.

Developers incorrectly assume that they don't need to protect the encoded passwords from being seen by others. Passing the encoded version of a password through @Password has no effect but returns the input string.

For example,

@Password("lotusnotes")

will always return

(DE9CA9CD7BD212362B6D312A33E10FB2)

and

@Password("(DE9CA9CD7BD212362B6D312A33E10FB2)")

will always return

(DE9CA9CD7BD212362B6D312A33E10FB2)

If you don’t protect the user names and corresponding encoded passwords of your registered users, and you don’t accommodate for passwords that could be encoded passwords in your login implementation, your login function will not be secure. Users may be able to authenticate using encoded passwords.

Recommendation

Since trustworthy user identities are essential for security, use of the standard Lotus Notes login functions rather than a custom login function is recommended. However, if you must use a custom authentication function, take the following precautions to create a secure implementation:

- Be sure that the user IDs and hashed passwords of registered users aren’t accessible by any of the techniques described in this paper.
- Don’t allow passwords that are 34 characters long and that start with “(” and end with “)”. These could represent a hashed password.
- Use the “Upgrade to More Secure Internet Password Format” action in the “People” view of the Public NAB to upgrade user passwords to a more secure Internet password format. This action runs an agent entitled “(SetNewPasswordFormat).” **Note:** By doing so, clients will only be able to access 4.6 servers and above.

Be sure that any field validation involving usernames and passwords will run in a secure fashion; i.e., it should not be implemented using client-side JavaScript or any other client-side technique. Typical field and form validation on Web clients is done for convenience and to reduce server load. Since in these cases there aren’t any security concerns, client-side JavaScript is typically used. In the case of a login function, you should implement the validation on the server side to protect it from being manipulated at the client.

Ultimately, avoid building your own login. Use the standard Domino authentication, which is secure. For a complete explanation of Domino authentication,see the *Lotus Notes and Domino R5.0 Security Infrastructure Revealed* redbook, which is available at http://www.lotus.com/redbooks.

Hide-When Techniques

Background

There are many hide-when techniques in use today designed to prevent Web users from seeing certain pieces of information. Typically, insecure documents contain confidential pieces of the information that should be hidden or removed prior to displaying to a Web user. The most common of these hide-when techniques are:

1. Hide-when… formulas
2. WebQueryOpen agents

WebQueryOpen agents, for example, run before Domino converts a document to HTML and sends it to the browser. This gives the developer the opportunity to make changes to the document before it is displayed to the Web user. Clearing fields containing confidential information has become a common use for these agents. The following is an example of a LotusScript WebQueryOpen agent that clears the CardType, CardNumber, and Expiry fields before displaying a document to a Web user.

```
Sub Initialize

     Dim session As New NotesSession

     Dim docContext As NotesDocument

     Set docContext = session.DocumentContext

     REM  Clear Credit Card fields so that they will not

     REM  be displayed to Web users.

     Call docContext.ReplaceItemValue( “CardType”, “” )

     Call docContext.ReplaceItemValue( “CardNumber”, “” )

     Call docContext.ReplaceItemValue( “Expiry”, “” )

End Sub
```

Implication

Developers assume that Web users have no way to get to the contents of the documents without going through the hide-when logic. This is usually true unless the hosting server supports the Internet Inter-ORB Protocol (IIOP) protocol, which enables remote Java invocation of Domino Object Model. Be aware that enabling the DIIOP server task in combination with allowing for server browsing may permit a malicious attack on your Domino server to occur.

Recommendation

The simple solution in this case is to not enable the DIIOP server task in combination with allowing for server browsing. The majority of situations will fall into this scenario. However, this will not address any other topics discussed in this paper. Ultimately, the secure technique is to use field-based access controls in conjunction with database, view and document user access controls. Placing only the usernames, roles and group names in the Readers field will effectively prevent all unwanted access to sensitive information. This technique properly covers all malicious attempts at compromising your data.

Disable Server Browsing

Background

Databases must be in the \Data directory or a subdirectory off the \Data directory of your Domino server in order to be accessed by a URL command, except in the case of server commands such as ?OpenServer.

Implication

Enabling server browsing provides users with links to all databases on the Domino server. Any Web browser can view a list of every database in the \Data directory or a subdirectory off the \Data directory if the server is configured to allow this action. In most cases, server browsing is undesirable.

Note: Server browsing only provides a list of links to databases. It does not provide any additional access rights to the user. For a user to access any of the databases listed, the database ACL must be configured to allow access.

Recommendation

Server browsing is preventable by setting the "Allow HTTP clients to browse databases" field in the server document to 'No.'

Note: By default, the "Allow HTTP clients to browse databases" field in the server document is set to 'Yes.' Now, Web users cannot see a list of databases, although they can still open individual databases to which they have access. Hiding the list of databases is useful if you have virtual servers on one machine or if some databases aren't for Web use.

CORBA/IIOP

Background

With Release 5.0 we have introduced an architecture called CORBA (**C**ommon **O**bject **R**equest **B**roker **A**rchitecture) into Domino. This is an open standard defined by the OMG (Object Management Group). The OMG is an industry standards body with over 800 worldwide participants, including IBM.

CORBA serves as middleware and facilitates the design and implementation of distributed systems by providing a transport through which distributed objects can locate and exchange data with each other. It also provides language, operating system, hardware platform and networking interoperability. The transport protocol for CORBA communication over a TCP/IP network is IIOP (Internet Inter-ORB Protocol).

CORBA is supported by IBM, Netscape, Oracle, and Sun, and promoted as an alternative to Microsoft's DCOM (Distributed Component Object Model).

With respect to Domino, this allows Java programs on remote clients such as applets in browsers and stand-alone Java applications to access the Domino objects on the Domino server. From an implementation standpoint, a remote client instantiates and references DOM objects as if they were resident at the client. In fact, these objects are instantiated at the Domino server. When the client is referencing these objects, it is actually communicating with the objects on the server. This is seamless to the programmer.

Implication

Be aware that when a Domino server has the DIIOP task enabled, unwanted attempts to compromise data security are possible under certain conditions. For example, data could be compromised by using a stand-alone Java application with CORBA/IIOP to access a Domino server, create a DBDirectory object and proceed to browse all of the databases left open, assuming browsing is allowed.

Note: When server browsing is disabled by setting the "Allow HTTP clients to browse databases" field in the server document to 'No,' any unwanted attempts to compromise data security using a stand-alone CORBA application are not possible.

Recommendation

The simple solution in this case is to not enable the DIIOP server task in combination with allowing for server browsing. The majority of situations will fall into this scenario. However, this will not address any other topics discussed in this paper. Ultimately, the secure technique is to use field-based access controls in conjunction with database, view and document user access controls. Placing only the usernames, roles and group names in the Readers field will effectively prevent all unwanted access to sensitive information. This technique properly covers all malicious attempts at compromising your data.

“No Access” Rights to Site Databases

Background

Every Domino server comes with a number of templates and configuration databases for installation. These databases are crucial to controlling the Domino environment. Database default ACLs (Access Control Lists) are set with “Designer” privileges. Also, the default setting for the ‘Max Internet Access’ is set to “Editor” privileges.

Implication

These databases have ACLs that may not be set correctly for your environment. Thus, if this issue is not addressed prior to deployment, Web users potentially have access to site databases.

Recommendation

Review the ACL settings for each .NSF file on your Domino server. In addition to checking the ACL for your application databases, be sure to also check the ACL’s of system databases such as NAMES.NSF, LOG.NSF, ADMIN4.NSF, DOMCFG.NSF, etc. Be sure to review every database including MAIL.BOX, AGENTRUNNER.NSF, BILLING.NSF, BOOKMARK.NSF. In addition, ask yourself these questions:

What is the “-Default-” access control set to?

What is “Anonymous” set to?

What is the Maximum Internet name & password access set to in the Advanced ACL settings?

Note: If the Web user is being authenticated with Basic authentication (that is, name and password), the Advanced section of the ACL lets you specify a maximum access setting for Web users. Even if you explicitly give Web users higher access, they never have an access level greater than what you specify as the “Maximum Internet name & password access.”

If the Web user is being authenticated using SSL Client certificates, then the “Maximum Internet name & password access” field does NOT apply to them. These users have the full access granted to them in the ACL. In other words, if the access list says that they have “Manager” access, they truly have Manager access to the database and not the access specified in the “Maximum Internet name & password access” field.

Tip: It may be easier to write a simple agent that will loop through every database on the Domino server, add a group name to each database and also set proper rights to “-Default-” and “Anonymous.”

Unwanted Searches

Background

Domino provides search-related URLs for performing view, multiple-database, and domain searches. Typically, either a customized search form or the default search form is presented, which allows users to define their own searches. Developers can then take the user input and programmatically build Domino Search URL's to perform text searches from a Web browser. An example might be:

http://host/db.nsf/view?SearchView&Query=FIELD+Manager+contains+Bob

Implication

It's possible to reconstruct a Domino URL in order to conduct a search and have the results returned without using a designated search result form. This circumvents the control of the developer to contain searches and ultimately makes confidential documents available to the user.

Recommendation

When a database is full-text indexed and a search page is requested, Domino will first look for a form in the database named $$Search to display. If this form is not available, Domino will look for and use a form in the database named $$SearchTemplateDefault to display. If neither are present, the default search page is used.

Note: The default search form is the search.htm file located in the \Data\Domino\Icons directory of the Domino server.

Create a form named $$SearchTemplateDefault in your applications. The search results created outside of the controlled search form will be published using the $$SearchTemplateDefault form. If the form contains nothing but the text "Search Access Denied," the user will see only the text "Search Access Denied," and documents will remain secure. This method does not hinder regular search methods; rather it is an addition for other purposes including security.

The bottom line is that to prevent misuse of database searching, simply include a $$SearchTemplateDefault with no $$ViewBody field.

Conclusion

This paper has described techniques for using the Domino Security Model when developing an application intended for Web browsers. To ensure data is accessible only by those users it was intended for, appropriate concern should be given to security. If you are developing an application that will conduct business on the Web, you must ask yourself, “Have I done everything necessary to protect this data?” As an example, if you are developing an order form to be completed by users who will ultimately enter their credit card number and expiration date on this form, do you have a Readers names field on the form? Is this data only accessible to the person who created it and for whom it was intended?

All of the techniques described above were put in place not as part of the Domino Security Model, but primarily as part of a set of features intended as a benefit to Domino developers. Even though they are not considered part of the Domino Security Model, they do act as an extra layer of security for those who either inadvertently or deliberately attack a Domino Web application. Developers are encouraged to use techniques such as hidden views, view templates and form formulas, but only in conjunction with implementing the fundamentals of the Domino Security Model.

To ensure that your sensitive data is properly protected, follow the summarized lists below before deploying your Domino Web application.

Fundamentals

1. Fully understand the Domino Security Model and ensure that it is implemented in your Web applications.
2. Fully understand the Access Control List (ACL) feature.
3. Fully understand field-based access controls AND use them for your Web applications.
4. Fully understand Domino server, database, view, and document access control lists, and use them throughout your Web applications.

Development Considerations

1. Protect all views you don't want accessed, because there are methods by which Web users can see all your database views.
2. Do not rely solely on View Templates as a secure way to protect your data.
3. Do not rely solely on Database Web Launch properties as a secure way to protect your data.
4. Do not rely solely on hidden views as a secure way to protect your data.
5. Do not rely solely on form formulas as a secure way to protect your data.
6. Protect your agents from being invoked by Web users by implementing the techniques described above.
7. Do not rely solely on CGI variables in your security implementation.
8. Be aware that Custom Authentication using @Password may be compromised.
9. Be aware that using an agent as a technique to hide data may be compromised by using a CORBA/IIOP application.
10. Review the Access Control List (ACL) settings for each data file, including .NSF, .NTF, .BOX, .NS2, .NS3, .NS4, .NSG, and .NSH files on your Domino server and set rights accordingly. Set all databases to have Anonymous and -Default- access privileges set to 'No Access.' In this way only validated users can get into databases. **Note:** This may not be appropriate for a home page, in which case use either a separate .NSF file or .HTML file as your home page.
11. To prevent misuse of database searching simply include a $$SearchTemplateDefault with no $$ViewBody field.

Server Environment Considerations

1. For Domino servers that are located on the Internet, extranet, or outside your firewall system, put the external Domino server in a separate organization in case some of your databases have */Org in the ACL, and then cross-certify the external Domino server with internal Domino servers.
2. Especially for external Domino servers, put a difficult-to-guess password on the Notes server ID, in case someone manages to steal it off your site. This will help protect any encrypted databases. **Note:** The Domino server ID must be manually unlocked by an administrator when the server starts. In other words, the server will not restart if unattended.
3. Encrypt all databases with the Notes server ID in case someone manages to steal your databases. Choose between Simple, Medium and Strong encryption, keeping in mind that stronger encryption will have an effect on performance.
4. Set all databases to enforce local security, in case someone manages to get the database and the server ID.
5. Turn off database browsing for Web clients so people can't reach databases you didn't mean to publish.

6. Activate SSL (whether using a self-certified certificate or a certificate from a Certificate Authority (CA) such as Verisign) to secure your network traffic with Web clients.
7. Turn on network encryption when Notes clients talk to the Domino server over the Internet.
8. If you are using the Domino server as a Notes server, put files you don't want Web users to access in a directory with a DirLink, and disable Domino's DirLink support to allow Notes users to use the DirLinks but not Web users.
9. Be careful with the naming of your database and design elements. Be aware that your naming convention may be picked up by someone maliciously trying to access your data.
10. Be aware that enabling the DIIOP server task in combination with allowing for server browsing may permit a malicious attack on your Domino server to occur.

Security Assessment

After reading this document, should you require a security assessment of your Domino Web application, IBM Global Services offers Domino-specific Application Security Assessment and Ethical Hacking services. For more information, please contact Eriks Taube (etaube@ca.ibm.com) at 416-490-5131.

Lotus®

An IBM Company

SUPERHUMANSOFTWARE™

www.lotus.com

Part No. CT6L9NA