Interested in learning more about security?

SANS Institute
InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

Securing End User Active Server Page Applications on an Intranet

End user computing solutions have evolved from the mainframe to the personal computer and now to the web. Business partners must be empowered to create their own applications, but the enterprise must still ensure that adequate control and security is in place. This can be especially challenging in a Microsoft IIS intranet environment, as Microsoft has made it easy for end user developers to create powerful, but unsecure, applications. Creation of highly secure applications can require more skills than are possessed by ...

Copyright SANS Institute

Author Retains Full Rights

AD

Securing End User Active Server Page Applications on an Intranet

GSEC Practical Version 1.4a Option 1

Bob Bohn
August 4, 2003

Abstract

End user computing solutions have evolved from the mainframe to the personal computer and now to the web. Business partners must be empowered to create their own applications, but the enterprise must still ensure that adequate control and security is in place. This can be especially challenging in a Microsoft IIS intranet environment, as Microsoft has made it easy for end user developers to create powerful, but unsecure, applications. Creation of highly secure applications can require more skills than are possessed by the average end user developer and compromises may need to be made. This paper discusses the evolution of end user computing as well as the issues involved, and explores a number of techniques which can be used to secure end user applications in a Microsoft IIS 4.0 intranet environment.

The Evolution of End User Computing

Since their first introduction into business a half century ago, computers have become essential, but complex, business tools. Specialized skills were required to manage both the hardware and software aspects of computing. Most large companies created separate departments called Information Systems (IS), Information Technology (IT), or something similar, and staffed them with specialists who managed the enterprise computing environment.

In an era of centralized mainframe computing, this arrangement made a lot of sense. The hardware was an enterprise asset which was shared by multiple business departments. Most of the software applications also addressed functions at an enterprise level. Individual business departments had no more desire to maintain this enterprise environment than they did to maintain their own telephone system.

As computing became more pervasive, business departments began to become unsatisfied with this arrangement. Relying on a centralized IT department to develop software had several disadvantages. While the business department

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

was the source of the business requirements and logic, the job of translating this logic into software needed to be performed by IT programmers. Much like the children's game of telephone, this additional communication between the players often resulted in misinterpretation of the business logic. Involving more people also increased the amount of time required to complete the task. A second problem was prioritizing the work. Since IT handled the programming for all business departments, one business department's request was competing with requests from other business departments. Less important requests had little chance of being done at all. Individual business departments no longer had control over their destiny.

While time-sharing systems such as IBM's TSO provided some capabilities for the business departments to directly interact with the enterprise computing environment, there were limited tools available to the business end user. Reporting and query tools allowed the end user to tap into the existing IT databases, but didn't provide a friendly interface to enter new data. And even these limited tools required some IT skills.

All this changed in the early 1980's with the introduction of the personal computer into business. The success of the personal computer was largely due to its ability to empower the end user with ground-breaking personal productivity tools such as VisiCalc and WordStar [13]. Single user applications could now be created without IT involvement. Over the years, these productivity tools have evolved into powerful packages such as Microsoft Office, which are quite capable of creating small workgroup solutions. Microsoft Access databases are generally used to house the data, and the application code may be completely generated by the wizard or supplemented by VBA code.

As powerful as these Microsoft Access applications are, they do have many limitations. Microsoft Access is not a database manager with server code, so network traffic, performance, and scalability are concerns. Security options are limited and often not well implemented by the end user. Most often the database is just placed on a file server with little regard to access control. Managing the application can be very difficult. Additionally, the enterprise applications at many companies are now web-based, preventing a seamless integration of these non-web end user applications.

The natural evolution would be to allow the end user business partners to create web-based applications on their company's intranet [14]. But intranets are now the enterprise production environments for most companies, and are managed by the IT department, much like the mainframe environment. Because this is a mission critical environment, it needs to be highly secure and carefully, perhaps overly, managed. Applications running on the production enterprise intranet must be carefully tested for stability, availability, scalability, and performance to ensure that they do not adversely effect other critical applications running in this

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

shared environment. In short, enterprise intranet applications must be industrial-strength.

Like the earlier mainframe environment, building a multi-tiered industrial-strength intranet application requires a high level of professional programming expertise, tools, and processes that are not usually available outside the IT department. They are also very expensive and take a long time to develop. Clearly this is not a viable solution for end user computing. It is overkill.

Fortunately, the typical end user computing application is not mission critical and does not require the high availability and scalability provided by the enterprise intranet. A separate intranet for end user computing can provide an adequate environment for these applications at a fairly low cost. More importantly, the applications themselves can be simpler, capable of being built by the end user developer with minimal assistance from IT.

Constraints of the End User Computing Intranet Environment

To be successful, the end user intranet and the applications running on it need to live within many constraints:

- The skill level required to build this type of intranet application must be within the capability of the typical end user developer. It should be comparable to the skills required to develop a Microsoft Access application. In a Microsoft IIS environment, this means that the developer should be able to create an application using HTML or ASP (Active Server Pages) with VBS (Visual Basic Script) or jscript (Microsoft's version of javascript) programming and enough knowledge of SQL (Structured Query Language) to interact with an SQL Server database. Advanced skills, such as the creation of Visual Basic MTS components, should not be required, as they may be beyond the capability of most end user developers.

- IT involvement should be minimized, both from a cost standpoint and also from a time-to-market perspective. Empowerment of the business departments is what this environment is all about.

- Standardization of both the environment and the solutions is also required to hold down costs. While the end user computing environment is simpler, it should conform as much as possible to the rest of the enterprise technology architecture. This will allow the existing support structure to be leveraged. The application solutions should also conform to a limited set of models. This will allow automation of web site configuration and much of the security.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

- Applications cannot be mission critical. Mission critical applications belong on the enterprise intranet, which is engineered for availability and performance, and the applications themselves must be robust enough to take advantage of that environment. The end user computing environment is engineered to keep costs low. If a server fails after normal business hours, it may not be fixed until support staff report the next day for their normal shift. Rather than spending time and money performance testing applications, any application which consumes too many resources or causes problems with the environment is simply removed until the problem is fixed.

Securing the Application and the Database Password

Securing an end user intranet application has two main objectives. First, and most obvious, use of the application must be restricted to only authorized users. But it is often even more important to restrict access to the application source code, since the source code may contain the algorithm and data that you are trying to protect. If an unauthorized user is able to use the application, at least the damage is limited to only the functionality which can be performed by the application. But if the source code is disclosed, even to a user who is authorized to use the application, an unencrypted password to the database might also be disclosed. With knowledge of the database password, the user will be able to access the database with general query tool or utility, and perform operations such as updates, which would normally have been restricted or audited. Worse yet, the database password could be given to someone else, even an unauthorized user, and they would also have unrestricted access to the data.

This need to restrict access to the end user intranet applications limits the choice of acceptable authentication methods. Anonymous authentication, while common on the Intranet, is not a viable choice because it would allow anyone to access the site. Basic authentication does allow us to identify the user and utilize that information to restrict access. However, basic authentication transmits the user name and password over the network using Base64 encoding. While better than no encryption at all, Base64 is very weak and creates a risk that this information could be captured by a sniffer and decoded. The risk of having a transmitted password discovered can be eliminated by utilizing challenge/response authentication, which transmits the user's credentials as hashed values. Challenge/response also allows the establishment of fairly granular access rights, since it uses individual user accounts. Challenge/response is clearly the best authentication method for an intranet application. [1] [11] [12]

Microsoft has made it very easy to develop a simple database query or reporting program under IIS. An Active Server Page (ASP) only requires a few lines of script invoking ADO to connect to a SQL Server database, retrieve selected rows, and display the results to the user. The budding end user computing

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

developer will find sample applications using this technique in every article and book that introduces ASP development, and will likely incorporate it into their first end user computing application. And why not? The technique is easy to code and performance is adequate for this environment. The developer may have initially been concerned about the ADO connection string, which contains not just the database server name, but also an application database ID and (ominously) the password for that ID. But this connection string is contained within an ASP, and the developer knows that the "View Source" option of the web browser only displays the HTML code which the ASP generated and sent to the browser, not the ASP source. The password should be safe from prying eyes. Unfortunately, in an intranet environment, the files on the IIS server can be accessed from the NT file system by mapping a network drive to the IIS server. And since the ASP files are text files, anyone can use Notepad to view the contents of the ASP, including the password in the connection string.

How can we prevent someone from mapping a network drive and viewing the ASP file? The first reaction is to use the standard method for restricting access to a file, namely NTFS file permissions, to prevent the file from being read. But since IIS also uses NTFS file permissions to control a user's access to the ASP page, the user will also no longer be able to access the intranet application. (We might try to allow execute-only access, but will quickly find out that the ASP interpreter requires read access). So while NTFS file/folder permissions can be used to prevent unauthorized users from accessing the application from both a browser and a network drive, we need to look for another solution to prevent the application's authorized users from discovering the password.

If we can't prevent the ASP file from being viewed, perhaps we can solve the problem by removing the password from the ASP file so that at least the password cannot be viewed. Do we actually need to use a separate application ID, or could each user authenticate to the database with his or her own ID, allowing the database to manage security? There are several problems with this approach. For starters, this will negatively impact scalability and performance, since you can not take advantage of database connection pooling. To pool connections requires that the connection strings be identical, so the ID cannot vary by user [2]. Also, in most cases the SQL database will not be located on the web server or domain controller. Challenge/response authentication will not work in this environment; basic authentication just be used instead, leaving the username and password vulnerable to network sniffers. [10]

We've established that an application database ID and password will be required. Perhaps we could store this information outside of the ASP file so that it would not be disclosed if the ASP source was viewed by network drive mapping. ADO connection string information can also be place in a UDL (Universal Data Link) file. The ADO database connect then specifies the name of the UDL file instead of the sensitive connection string information. While promising, this is an even worse solution than leaving the password in the ASP, as the UDL file needs the

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

same NTFS file permissions as the ASP page which references it, so it offers no additional protection against mapped network drive access. The fixed format of the UDL file requires that the password be clearly visible in plain text; the password in an ASP can at least be obscured from casual viewing by programmatically constructing it with the scripting language, perhaps using ASCII numeric values instead of the actual character codes making up the password string [5]. Because the ASP is running under the identity of the web browser user, any file directly accessible by the ASP will need permissions which would allow the user to access the file via a mapped network drive. This restriction also applies to server side include files incorporated via the #include directive. Depending upon the how IIS is configured to handle the file extensions, storing the password in an external file may even allow the file to be viewed by the browser as well.

Storing the password in a file external from the ASP code, such as web.config or global.asa, might be a viable solution if the password could be encrypted. If the password is stored in encrypted form, restricting access to the external file via NTFS file permissions becomes less important. Someone viewing the file via a mapped network drive would be able to see only the encrypted password, not the password itself. Because the password must be in plain text when used in the ADO database connection string, the ASP script must contain the code which performs the decryption. Since we are unable protect the plain text ASP script from being accessed via a mapped network drive, the decryption technique will be disclosed. With access to both the encrypted password and the decryption technique, the password can be decrypted. There is some benefit to encryption, as it prevents casual discovery of the password and increases the skill level required to obtain the password.

Another option is to store the database password in the Windows registry on the web server. This has promise, as it eliminates the vulnerability of file mapping. Remote access to the registry can be restricted by setting the \CurrentControlSet\Control\SecurePipeServers registry key [9], but this method still has other vulnerabilities. Since the ASP source is still unprotected from network drive mapping, the application user can view the source to determine the registry key under which the password is stored, the method of reading the registry, and any decryption technique used. Another end user computing developer could use this information in their application to extract the password for someone else's database. Additionally, utilizing the registry requires some additional developer and web administrator work, as a script to place the password into the registry will need to be developed and executed on each web server. This installation script will also need to be secured, as it contains the database password we are trying so hard to protect. While not perfect, storing an encrypted password in the registry does further limit the number of people who have both the skill level and access privileges to decrypt the password.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

The Connect-As Method

All of the approaches we've examined so far have failed because IIS uses the identity of the requesting user to determine NTFS file permissions when accessing either an Active Server Page or a file accessed from the ASP. To grant the user the necessary read permission on the web means that we must grant them read permission when they map a network drive, which allows them to view the file containing the password or the logic by which the ASP accesses the password. If we could force IIS to use a different identity than that of the requesting user, we could use simple NTFS permissions to allow access via the web but deny access via a mapped network drive.

As it turns out, while not obvious, it is relatively simple to configure an IIS virtual directory to do this. From the IIS console, right click on the website and choose Properties. Select the Virtual Directory tab. At the top of this screen, there is a configuration option for "When connecting to this resource, the content should come from:" with several choices. The normal choice is "A directory located on this computer" with the local path to the virtual directory specified. If we create a network share for the virtual directory on this computer, we can instead specify that the content will come from "A share located on another computer". However, in our case, "another computer" will actually be "this computer". Instead of the local path to the virtual directory, we specify a network directory of *servername**sharename*, where *servername* is the name of this computer and *sharename* is the name of the network share we just created. We can now specify a username and password to be used as "connect as" for this share. This username and password will be used as the security credentials when accessing the network share from this site. If we create a username called ConnectAsID and use it as the "connect as" name, we can now use NTFS folder permissions to restrict access to the virtual directory to only ConnectAsID. Any user accessing our web site from a browser will now have the folder permissions of ConnectAsID, giving them access to our web site. However, when mapping a network drive, no one but ConnectAsID can read the content of our web site files. This means that the ASP can contain the database ID and password, but no one except for ConnectAsID (and administrators) can view the value in the source.

While solving our problem of securing the database password, this method (which I will refer to as the Connect-As method), has now given everyone web access to our application, since we can no longer restrict them via folder permissions – everyone is using ConnectAsID's folder permissions. For web sites containing only nonsensitive data, this may be acceptable, as the only requirement may be to keep the database password confidential. If so, we've been able to satisfy the security needs without requiring any additional effort from the end user developer.

However, many sites contain confidential data and access must be limited to only authorized users. While we cannot use folder security to limit that access, we

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

can still determine the actual username from the AUTH_NAME server variable. With that information and a little application program logic, use of the application can be restricted to only authorized users. There are several authorization techniques which can be easily done by the end user developer using only simple VBScript within an ASP,

A small, nonvolatile list of authorized users can just be hardcoded within the ASP script itself, requiring a trivial amount of effort by the developer. The list could also be kept in a database and accessed via ADO. For some applications, the database may already contain this information. For instance, the database for a survey application might be pre-populated with a response row for each user who is authorized to take the survey, indexed by username. Since the application should only allow a user to update their own row, the mere existence of a row for the AUTH_USER serves as authorization for the user.

Often, there are existing Windows NT groups containing the authorized users for the application. While we can no longer use them to control web folder permissions due to our use of the Connect-As method, we can simulate the use of traditional folder security by using ADSI calls to determine if the AUTH_USER is a member of one of these NT groups. This only requires a few lines of VBScript code and is well within the capabilities of the end user developer. [3] [8]

```
Set Group = GetObject("WinNT://" & strDomain & "/" _
              & strGroup & ",group")
Set User = GetObject("WinNT://" & strDomain & "/" _
              & strUser &  ",user")
If Group.IsMember(User.ADsPath) then . . . ` User is in group
```

The main advantage of the Connect-As method is that secure applications can be written in easy-to-learn ASP code. It does not require the end user developer to have the additional skills required to write an MTS component. There are a number of disadvantages to this method, however. There is a small performance hit because ASP pages on this Connect-As web site are not cached [15]. More serious performance problems can be caused by the application logic required to perform authorization, especially if numerous ADSI calls are required. This authorization check cannot be done in just the main ASP for the application, since this would allow a user to bypass the main ASP and directly call one of the "worker" ASP subordinate functions. For performance reasons, it is desirable to only perform this authorization once per user session rather than on each interaction. This will require the user of session state cookies or other methods to keep track of whether or not the user has been authorized. All this additional coding can quickly make the application much more complex than an application which could rely on folder security.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

The Case for a Custom Component

After much research, I've concluded that trying to write secure end user computing intranet applications using only ASP code is not the right approach to take, at least in a Microsoft IIS 4.0 environment. The Connect-As method comes close, but still has too many undesirable side effects. Creating a compiled MTS component instead of plain text ASP scripts could secure the database password while still allowing simple folder security for authorization, but it requires too great of skill set for the average end user computing developer.

If a compiled MTS component is the best solution but cannot be created by the end user developer, the most viable approach in an IIS 4.0 environment is to have the IT department provide the end user developer with a pre-written component which will open an ADO connection object using an encrypted password contained within this compiled component. The functionality of this component should be kept simple: decrypt the database password, construct the connection string, and open the database using that connection string. If the end user computing environment contains both test and production, the component can be made to handle both. The component can be created from a cookie-cutter pattern, changing only the test and production server names, database names, database IDs, and the DES3 encrypted database passwords. Using this technique, a custom component can be created in under fifteen minutes.

The call to this custom component replaces the ADO connection Open method. Other ADO calls in the ASP page remain unchanged, so there is minimal impact on the end user developer.

ASP code using a connection string:

```
set CN = Server.CreateObject("ADODB.Connection")
CN.Open "PROVIDER=SQLOLEDB;Server=servername;" + _
        "UID=dbid;PWD=password;Database=dbname"
```

ASP code using a custom open connection component:

```
set CN = Server.CreateObject("ADODB.Connection")
Set objCustom = Server.CreateObject("EUCAPP1.ADOConnect")
rcCustom = objCustom.ADOConnection ("test", CN)
```

There are many advantages to using this custom component. The database password is extremely secure. Not only is it DES3 encrypted, it is contained within a compiled component, and that MTS component can be secured with NTFS permissions which prevent it from being viewed or copied. Knowledge of the database password is also more strictly controlled, as even the developer does not need to know the password. This also prevents the application database ID and password from being disclosed to others or from being used outside of the application's control in a query tool or other application. It does not require the end user computing developer to have advanced skills; the

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

application can be written entirely with ASP code. NTFS folder permissions can be used to control access rather than requiring application logic; this results in applications which are both simpler to develop and better performing.

There is a small cost to the IT department to develop the custom component, but this is likely to be less than the cost of administering and reviewing all the alternatives that the end user developer might create, and the resulting security is stronger. Use of a common security solution will also promote automation efficiencies in setting up and administering each end user web site.

While the use of this custom component is very effective at securing the database password, it does have a vulnerability which may be significant depending upon how the end user computing environment is set up: the component can be invoked by another end user application on the web server. If new applications are subject to a review, the review point may be sufficient to prevent unauthorized use of the component by another application. Another option is to add to the complexity of the custom component by introducing the use of MTS roles. The role can be populated with authorized users of the application. While this does not prevent a second application from utilizing the custom component, it does restrict the use of that second application to only those users who are authorized to use the first application. This may be sufficient if the database ID used by the component only offers read access to the database, as the authorized audience remains the same. And finally, don't overlook non-technical methods such as auditing and accountability to discourage the unauthorized use of the custom component by an end user developer.

The Future

As the end user computing environment migrates to more powerful technologies, such as .NET, additional and perhaps better solutions will become available. For instance, the version of ASP.NET for Windows .NET Server 2003 introduces a tool called aspnet_setreg, which allows the use of an encrypted username and password stored in a secure area of the registry. Text placed in the web.config file provides information to instruct IIS to use this identity for the ASP.NET worker process which accesses the database. [4] [6] [7]

In addition to environment changes, we can also expect continual advances in the development tools which are available to the end user computing developer. The security options will need to be continually reevaluated as these factors change. End user computing is not static; the security cannot be static either.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

References

1. Morey, James. "Untangling Web Security: Getting the Most from IIS Security." 5 January 1999. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/WebsecIISsec.asp.

2. Microsoft Corporation. "Microsoft Database Security." URL: http://www.governmentsecurity.org/articles/MicrosoftDatabaseSecurity.php.

3. Bolte, Remie. "Add to Your ADSI Code Library." URL: http://www.15seconds.com/issue/020130.htm.

4. Falter, Chris. "Resolving the ASP.NET Database Security Dilemma." URL: http://www.eggheadcafe.com/articles/20021211.asp.

5. Microsoft Corporation. "HOWTO: Encrypt a String with Password Security." Microsoft Knowledge Base. 4 March 2003. URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;110308.

6. Microsoft Corporation. "HOW TO: Use the ASP.NET Utility to Encrypt Credentials and Session State Connection Strings." Microsoft Knowledge Base. 1 July 2003. URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;329290.

7. Landgrave, Tim. "Design secure intranet applications to meet standards." 26 June 2003. URL: http://www.cnetasia.com/builder/architect/system/0,39009336,39137607,00.htm.

8. Mueller, Richard. "VBScript program demonstrating the use of an efficient IsMember function to test for group membership for a single user or computer." 13 July 2003. URL: http://www.rlmueller.net/IsMember4.htm.

9. Computer Incident Advisory Capability. "Microsoft Internet Information Server 4.0 Security Checklist." 11 February 1998. URL: http://www.ciac.org/ciacNT/iis/CheckListFurtherDetails.htm.

10. Microsoft Corporation. "Choosing a SQL Server Authentication Method." URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsentpro/html/veconchoosingsqlserverauthenticationmethod.asp

11. Windows Web Solutions. "IIS 101: The Basics of IIS Authentication." 11 October 2000. URL: http://www.windowswebsolutions.com/Articles/Index.cfm?ArticleID=15843.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

12. "IIS User Authentication Tutorial." URL: http://www.authenticationtutorial.com.

13. Maxfield & Montrose Interactive Inc. "The First Personal Computers (PCs)." URL: http://www.maxmon.com/1973ad.htm.

14. VeriSign, Inc. "Guide To Securing Intranet And Extranet Servers." 16 October 2002. URL: http://secinf.net/misc/Guide_To_Securing_Intranet_And_Extranet_Servers.html.

15. Braginski, Leon. "Virtual Directories: Targeting Local Directories and Network Shares." September 2002. URL: http://www.windowswebsolutions.com/Articles/Index.cfm?ArticleID=25930.

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Last Updated: May 19th, 2013

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Malaysia @ MCMC 2013	**Cyberjaya, MY**	**Jun 03, 2013 - Jun 08, 2013**	**Live Event**
SANS Pen Test Berlin 2013	**Berlin, DE**	**Jun 03, 2013 - Jun 08, 2013**	**Live Event**
Industial Control Systems Security Training - Houston	**Houston, TXUS**	**Jun 10, 2013 - Jun 15, 2013**	**Live Event**
Security Impact of IPv6 Summit 2013	**Washington, DCUS**	**Jun 14, 2013 - Jun 16, 2013**	**Live Event**
SANSFIRE 2013	**Washington, DCUS**	**Jun 14, 2013 - Jun 22, 2013**	**Live Event**
SANS Canberra 2013	**Canberra, AU**	**Jul 01, 2013 - Jul 13, 2013**	**Live Event**
SANS London Summer 2013	**London, GB**	**Jul 09, 2013 - Jul 16, 2013**	**Live Event**
Digital Forensics & Incident Response Summit 2013	**Austin, TXUS**	**Jul 09, 2013 - Jul 16, 2013**	**Live Event**
SANS Rocky Mountain 2013	**Denver, COUS**	**Jul 14, 2013 - Jul 20, 2013**	**Live Event**
SANS Mumbai 2013	**Mumbai, IN**	**Jul 22, 2013 - Jul 27, 2013**	**Live Event**
SEC528 SANS Training Program for the CompTIA® New Advanced Security Practitioner (CASP) Certification	**Washington, DCUS**	**Jul 22, 2013 - Jul 26, 2013**	**Live Event**
SANS San Francisco 2013	**San Francisco, CAUS**	**Jul 29, 2013 - Aug 03, 2013**	**Live Event**
SANS SEC 560: Network Penetration Testing @ Bangalore 2013	**Bangalore, IN**	**Aug 05, 2013 - Aug 10, 2013**	**Live Event**
SANS Boston 2013	**Boston, MAUS**	**Aug 05, 2013 - Aug 10, 2013**	**Live Event**
Critical Security Controls Summit	**Washington, DCUS**	**Aug 12, 2013 - Aug 18, 2013**	**Live Event**
Industrial Control Systems Security Training - DC	**Washington, DCUS**	**Aug 12, 2013 - Aug 16, 2013**	**Live Event**
Mobile Device Security Summit 2013	**OnlineCAUS**	**May 30, 2013 - Jun 06, 2013**	**Live Event**
SANS OnDemand	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**