Best Practices for Secure Web Development

Razvan Peteanu
razvan.peteanu@home.com

Revision 2.0 (1.0 was of limited circulation)

Revision Date July 18, 2000

Contents

1 Why? 1
1.1 Frequently Asked Questions 2

2 Best Practices 4
2.1 Security as part of the business picture 4
2.2 Security as part of the requirements gathering 4
2.3 Security as part of the architecture 4
2.4 Don't be anonymous when you won't end up so 5
2.5 Do not use administrative accounts unless needed 5
2.6 Don't use GET to send sensitive data! 6
2.7 Don't rely on the client to keep important data 7
2.8 Don't store sensitive stuff in the *SP page itself 7
2.9 Keep an eye on HTML Comments left in production code 8
2.10 Cross-site scripting 9
2.11 Check the wizard-generated or sample code 12
2.12 Middleware security 12
2.13 Declarative vs programmatic 13
2.14 PKI is not a silver bullet 14
2.15 Snake oil 14
2.16 Code reviews are your friends 14
2.17 Watch what you use 15
2.18 For troubleshooting, use the logs 15
2.19 Other pointers 16

About the author: Razvan Peteanu works in Toronto and can be contacted by e-mail at razvan.peteanu@home.com

1 Why?

The following document is intended as a guideline for developing secure web-based applications. It is not about how to configure firewalls, intrusion detection, DMZ or how to resist DDoS attacks. This is a task best addressed at system and network level. However, there is little material available today intended for developers. We have entered the dotcom age in which a web site is no longer an isolated site, but an extension of the internal business systems, yet there isn't much about how to create this extension securely.

Traditionally, developers have worked on systems for environments where malicious intents were not a real threat: internal systems, software for home use, intranets. There may have been occasional exceptions, sometimes with embarrassing outcomes, but they could be dealt with at HR level and the example prevented others from attempting it again. An isolated (read: not linked with internal systems) web site is not far from the same scenario: the security was treated mostly at the system level by installing the necessary OS and web server fixes and applying correct settings and permissions. If a breach occurred, the system was taken offline, rebuilt better and the site put up again. Everything at a system administration level.

However, as the Internet becomes more and more commercial (after all, this is where the .com comes from), a web site becomes more and more an application. Thus, the team has more and more developers, skilled in web and traditional development. However, few resources for them focus enough on security to make them aware about what's out there on the Internet. We often read that "this web site is secure because it uses 128-bit encryption". Most often, programming books will have a single chapter on security, compressing SSL, signatures, permissions, cookies and other topics in 20 pages. Little if anything is said about how to think maliciously about your own code, trying to find out if it has a vulnerability. Little if anything is said about how to do security-focused code reviews.

We hope this document will fill some of the gap.

It is and will continue to be a work in progress and your feedback is highly appreciated.

Target Audience

The primary audience are developers and architects as well as infosec professionals. Project managers may be interested as well to understand various issues that impact specifications and project schedules.

1.1 Frequently Asked Questions

What exactly do you mean by ‘information security’?

If you care about definitions, you’ll find many. The definition we use most is “Information security is comprised by a set of *technologies* and *processes* designed in order to *protect* the information-based assets and *enable* business functionality”. Is it the best? Most likely not, but it serves its purpose.

Several areas covered by security are:

- *authentication*: positively identifying parties involved in an information exchange
- *authorization*: controlling access to resources
- *privacy*: protecting information from third parties
- *non-repudiation*: making sure [with legal-strength] a user cannot deny performing a certain activity when it has been logged as such.
- *integrity*: protecting information from tampering, intentional or not
- *detection & monitoring* of unauthorized activities
- *legal aspects* regarding of protection and response

Some of the areas above are not within the scope of this document. Detection and monitoring is best addressed at system and network level while legal issues are best addressed by, well, lawyers.

I thought the firewall would take care of this. Or file permissions. Or SSL.

Each of the above is useful and necessary to ensure the overall security of the site, but they address different risks.

Firewalls protect a system from a different class of risks by preventing access to non-public services and preventing malicious network traffic to reach the server. SSL provides server (and sometimes client) authentication and communication privacy, but otherwise it’s blind to the content of the traffic. File permissions may prevent abuses of rights when two different user levels are involved but it will not do so between two users with the same level.

To draw a parallel to the traditional development, coding for security would be very roughly equivalent to putting error handling. It’s got to be *in*, nothing *around* the application can replace it.

I’m an experienced web developer and don’t think I need this.

This is not about how to do web development. It's specifically about how to do *secure* web development. Why is this emphasis relevant? Because creating an application able to withstand malicious use (and we'll see later what this could mean) is not something that (a) is immediately visible; a non-secure code can do its primary functionality very well (b) has been a concern during development phases (c) taught in programming books or seen in traditional development projects when the user community was limited and not particularly hacker populated.

Can't someone do this after I finish my dev work?

No. Within the context of this document, security needs to be built into the application from the beginning, it's not something that's applied at the end. Of course, we'll still have permissions and other administrative operations, but again, they are not a replacement.

Note: We will try to make this document as vendor-neutral as possible. However, the author's experience has been mostly with Microsoft technologies so there will be an inherent slant in this space.

2 Best Practices

2.1 Security as part of the business picture

Surprise-surprise. Until the past year or so, security and business did not often come together in the same paragraph.

Well, it shouldn't be a surprise because ultimately, security is not about technology but about *managing risk*. Security is present in Internet projects precisely because it's needed to mitigate some risks. Any business has some *assets* to protect and in the Internet world, it's the information assets we are concerned of. Examples of assets: integrity of the site content, site availability, data privacy, trust. As you can see, not all assets are physical.

Once the assets are identified, the next step is too identify the *risks*. If we look at the example above, we can quickly derive some risks associated with the enumerated assets: site defacement (the integrity is lost), site is brought down (remember the DDoS attacks?), customer data gets published on the web (credit card info is a typical example) or the transaction is made with the wrong party.

Now, having the risks clearly spelled out, thinking of what security measures must be put in becomes an easier task, which brings us to the next step:

2.2 Security as part of the requirements gathering

This stage is not specific to security but a normal step in building any project. The security would come into place for the following topics:

- identifying the *assets* (see 2.1)
- *use cases*. How the application will be used is essential to understand the security implications.
- identifying *the users, their roles and rights*. Again, this goes straight to designing the authentication and authorization schemes.
- *legal and business issues*: support for non-repudiation? An audit trail? Digital signatures (and if so, what is their legal status in the countries/states/provinces where the customers are)? Strong encryption (fortunately, the last months have seen a relaxation of export regulations, but it's still worth checking)?

2.3 Security as part of the architecture

As with any other item in the requirements list, the first place to address them is at architectural level. Most of the professionals who have been in the software industry for a couple of years have seen what happened with projects with poor or missing architecture: scrambling teams trying to patch the system so it provides the desired functionality or performance, unscalable applications, lost money and time.

In a parallel with the items under the requirements section, the security architecture will focus on:

- protective measures around the assets (permissions, logins, encryption etc)
- possibilities to abuse the use cases (this includes thinking of malicious use cases)
- selecting the platform and technologies that support the users, roles and access rights. This includes choosing an operating system, the web server, an application server if applicable, the directory service when a large number of users is concerned, a user authentication mechanism (anonymous, cookie, basic, challenge response, digest, certificate-based etc), the authentication mechanism between the different application tiers and so on. Certainly, the decisions are not made solely from the security standpoint but this is the role of the architect: to take in all the application requirements and find the best possible solution within the constraints.

2.4 Don't be anonymous when you won't end up so

If certain pieces of functionality require authentication, plan to use it as early as possible instead of continuing to use anonymous access (be it to a web server, a directory or as a guest-like account for the operating system. Using authenticated access to resources may require a different syntax and/or may expose authentication/authorization/impersonation issues that will otherwise stay hidden until later.

Also, using anonymous access to resources also means that the code responsible for authentication/authorization is not actually used. If it's not used, it cannot be [unit-] tested. If it cannot be tested, bugs cannot be discovered until later.

Certainly, the amount of security put in the development stage must be reasonable. For instance, enforcing complex and unique passwords might be a nuisance for the developers while they are writing the code. Such restrictions can be added later.

2.5 Do not use administrative accounts unless needed

Ummm… Just don't ask me how many times I had to say "Don't use the 'sa' account to access a SQL Server" ☺

Why is the administrative login not good, even in a secure environment without any sensitive data? Because it prevents application isolation, accurate testing and proper accountability and especially the first two direct impact the development work.

Using admin accounts is very appealing at the first sight: the developer doesn't have to bother with access restrictions and can focus on the functionality. You've already guessed it, the problem has just been spelled out: with admin accounts, there is no access restriction. The code can do anything, anytime. At least, until the release date comes closer or the code is moving in a pre-production environment where accounts and permissions are managed properly and then things start to break. Tables that used to be accessible or writable are no longer because specific access rights have not been assigned, ACLs are applied and various run time errors occur. In a distributed application even identifying the root cause can be a bit challenging. All these will add debugging time at a time when no one wants it.

There is another operational danger posed by using admin accounts: because access is not confined to a specific application you may inadvertently overwrite something else. When I was working on a project with SQL Server, I was personally very close to deleting someone else's tables because I was using the 'sa' account when operating from the management console. On that particular server there were several databases for different phases of the same project. They looked very similar, at least as the tables names went, so a slip of the mouse to the next database in row followed by a 'Select All' and Delete almost made me the first lynched individual in the company's history. I still live because of the confirmation message.

The lesson: use application-specific accounts with rights identified as early as possible. Yes, it is likely the access rights will have to be refined in time, but unless you start making use of them, how to find out?

2.6 Don't use GET to send sensitive data!

This is an old one but still very valid.

When sensitive data is to be passed to the server, do not send it as a parameter in the query string like in:
http://www.your-own-site-here.com/process_card.asp?cardnumber=1234567890123456

This is not appropriate because, like any other HTTP request, this will get logged in the logs of the web server as well as in whatever proxies might be on the way. The above request will get logged in clear text similar to:

```
2000-03-22 00:26:40 - W3SVC1 GET /process_card.asp cardnumber=1234567890123456 200 0 623 360 570
80 HTTP/1.1 Mozilla/4.0+(compatible;+MSIE+5.01;+Windows+NT) - -
```

Also, the entire URL may be stored by the browser in its history, potentially exposing the sensitive information to someone else using the same machine later.

SSL wouldn't help in this case, because it only protects the request *in transit*. Once arrived at the destination, the request will be happily decrypted and logged in clear text. An apparent rebuttal may come from the standpoint that the server must be trusted. Yes, it must but this trust implies that private customer data be dealt with sensibly. There is no reason for the server to store such sensitive information in clear text. In some credit card authorization schemes, the application server is but an intermediate and once the payment is authorized, the web application does not store the credit card number or at least not all digits.

The **POST** method uses the HTTP *body* to pass information and this is good in our case because the HTTP body is not logged. Note however, that by itself POST doesn't offer enough protection. The data's confidentiality and integrity are still at risk because the information is still sent in clear text (or quasi clear text as in Base64 encoding) so the use of encryption is a must for sensitive information.

2.7 Don't rely on the client to keep important data

If you work on the server side of the application, never assume that what you sent to the browser got back unchanged. A case in point is **relying on hidden form fields to maintain sensitive data between requests**. An example is with shopping carts that send the item price or a discount rate as a hidden field in the form so that when the customer submits the form, the price/discount will be submitted as well although this particular field has not been displayed to the user.

A malicious user can save the web page locally, change the hidden field to whatever he wants and then submit it or simply use a scripted tool to post fake orders.

Detailed information and an analysis of real-world commercial products that have this problem is found in **Form Tampering Vulnerabilities in Several Web-Based Shopping Cart Applications**, issued by the ISS on Feb 1, 2000 and available online at http://xforce.iss.net/alerts/advise42.php

A mechanism to detect tampering is using a hash of the sensitive fields. A *hash* (also referred to as a *message digest*) is a one-way function that processes a variable-length input and produces a fixed-length output. It is computationally unfeasible to reverse the process, that is to start from a given hash and find a message that produces it. Standard hashing algorithms are MD5 and SHA-1 and an example of how to use them with Perl is available at http://www.webtechniques.com/archives/1998/09/webm/ However, once you grasped the concept, there is nothing to stop you from implementing the same mechanisms with alternative technologies, such as COM components. A search will easily find a number of free or commercial components implementing MD5 or SHA1.

The article referenced above also illustrates another example of why relying on the client is not a wise decision, this time the focus being on the HTTP Referer field.

2.8 Don't store sensitive stuff in the *SP page itself

(*SP stands for ASP or JSP)

Most of the time, this "sensitive stuff" would be username/passwords for accessing various resources (membership directories, database connection strings). Such credentials can be entered there manually or automatically put by various wizards or Design Time Controls.

A legitimate question is why would this be a concern since the *SP is processed on the server and only its results sent to the client. For a number of reasons: from the security standpoint, the past has seen a number of holes in IIS that allowed the *source* of an ASP page to be displayed instead of being executed. For example, two [old and] very well-known IIS bugs caused the ASP being displayed by appending a dot or the string ::$DATA after the URL ending in asp (http://<site>/anypage.asp**.** or http://<site>/anypage.asp**::$DATA**) . Similarly, two recent bugs have affected JRun (a JSP engine) (http://www.ntsecurity.net/go/load.asp?iD=/security/jrun1.htm) and Weblogic (http://www.ntsecurity.net/go/load.asp?iD=/security/weblogic1.htm)

Another reason for not hardcoding credentials in the page itself relates to development practices. Such information should be stored in a centralized place. In the IIS world, such places are:

- **GLOBAL.ASA**
- the **IIS metabase**
- a public property of a COM component
- the registry.

GLOBAL.ASA is generally the best place because declarations put there have global or application scope and are easy to refer to from ASP. For example, you can have the database or LDAP connection string defined as an application-level variable. Then, referring to it would be as easy as something like `Application("DBConnectionString") or Application ("ADsPath")` where the two are variable names, you don't have to use the same. There is still a chance however that a security bug may reveal the source of global.asa.

The metabase is a directory-like collection where IIS keeps its settings and can be accessed through ADSI (the IIS Resource Kit has a useful GUI-based MetaEdit tool, but you could also use Option Pack's **MDUtil** command line approach). Physically it is stored in %WINNT%\%SYSTEM32%\inetsrv\metabase.bin, but once loaded the metabase is kept in memory by the web service and therefore access to it is fast. However, there is little reason to use it instead of the global.asa for regular tasks.

The last two methods are less used. Using the registry imposes a penalty on performance. Also, both add some administration overhead (component registration, .reg files etc), especially at publishing and deployment times.

2.9 Keep an eye on HTML Comments left in production code

This is a no-brainer. Of course, be sensible: not all comments are bad, only those embedded in the HTML or client script and which may contain private information (such as a connection string that was once part of the server side script, then commented out. In time, through inadvertent editing, it can reach the client script and thus be transmitted to the browser). The comments are not dangerous per se, but can reveal information.

2.10 Cross-site scripting

This is a more complex issue and, after going through the introductory pages below, the reader is encouraged to read the materials available at the following links (more at the end of the section).

CERT® Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests at http://www.cert.org/advisories/CA-2000-02.html

Not a very straightforward name, but a significant problem which can occur with sites that allow the user to input some data and later display it. Typical examples are registration information, bulletin board messages or product descriptions. In the context of this discussion, the "user" is the more or less anonymous user who visits the site and *not* the site administrator that changes the content.

Why is this a problem?

Because it breaches trust. When a user visits a site, it has a level of trust in the content that comes from the server. Usually, this means the user expects the web site will not perform malicious actions against the client and it will not attempt to mislead the user to reveal personal information.

With sites that accept user-provided data, later used to build dynamic pages, an entire can of worms is opened. No longer is the web content authored by the web creators only, it also comes from what other (potentially anonymous) users have put in. The risk comes from the existence of a number of ways in which more than the user-input fields can be manipulated to include more than simple text, such as scripts or links to other sites. Taking the script example, the code would be executed on the client machine because it would undistinguishable from the genuine code written by the site developers. Everything comes in the HTML stream to the browser.

Quick example: Let's take a site that allows users to input the user's name through a form and that the value entered is later displayed. For brevity, we'll use the same form for both inputting the string and displaying it. The source for the form is

```
<html>
<%
      if request.form ("yourname") <>"" then
            Response.Write("Hello " + request.form ("yourname"))
      else
%>
      <form method="POST">
            <input type="text" name= yourname>
            <input type="submit" value="submit">
      </form>
<%
      end if
%>
</html>
```

Enter Bad Guy who, instead of typing his name, types the following in the input field:

```
<script language='javascript' >alert ('gotcha!');</script>
```

When later the variable containing the name is displayed as part of a web page, *the visitor will get the script as if it were part of the legitimate site and the script will get executed on the browser.* Feel free to check for yourself and then view the HTML source of the response web page.

In our case, the script only consisted of a message box being displayed, but the author could be more "creative". Such a scenario becomes very dangerous when a web site accepts content from one user and displays it to others as well (the code above is rather usable for "self hacking"). Typical examples are web-based message boards or community sites. The injected script could perform unwanted actions on the client or send information to external sites.

Again, the fundamental issue here is that the trust the user put into the web site is broken: the web page that gets sent to the visitor contains not only trusted content from the authors but also untrusted content which, equally important, *cannot be identified by the browser* as being so.

There are other ways to inject script, such as *within an HTML tag*:

```
      <a href=" [event]='bad script here' "> click me </a>
```

The script can even be hosted on another web server (anonymous hosting companies or previously compromised servers being an excellent choice). In this case, the malicious string would contain links to the real script. An example below, illustrating an alternative way of submitting malicious content via cookies:

If the dynamic content comes from *a cookie (example taken from the Microsoft advisory)*:

```
      <% Response.Write("<BODY BGCOLOR=\"" +
      Request.Cookies("UserColor") + "\">"); %>
```

The cookie can be trivially manipulated on the client side to:

```
      Cookie: %22+onload%3D%27window%2Elocation%3D
      %22http%3A%2F%2Fwww%2Eevilsite%2Ecom%22%3B%27
```

which would lead to

```
<body BGCOLOR="" onload=
'window.location="http://www.evilsite.com";'">
```

redirecting the user to another site.

There are other ways to inject the script, please refer to the two hyperlinks at the beginning of the section.

What to do?

There are a number of ways of dealing with this issue. The core idea is to encode the user-input information in such a way that it will be displayed the same as the user input it but stored and transmitted in a form that will prevent the vulnerability from being exploited.

The solution is offered by what is called HTML Encoding, a technique used when transmitting special characters in an HTML code. In HTML, the characters < and >, for instance, have a special meaning: they signal the boundaries of a tag. But what if we want a web page to contain those characters? The workaround is to use special character sequences that will be stored as such but *displayed* as the character intended (similar to \t, \n from the C world). The character < is HTML-encoded as **&lt;** and the > sign is encoded as **&gt;**.

This is classic HTML knowledge for a web developer but how is this used? The information input by the user is HTML-encoded by the server and stored as such. For instance, the `Server` object in IIS exposes a method called exactly **`HTMLEncode`** which takes a regular string as input and produces an output string having special HTML characters replaced with the associated escape sequences. At display time, the HTML encoded string will be sent to the browser which will interpret the character sequences and display the characters accordingly. What this means is that if the Bad User typed in **<script>**, the server will encode it to **&lt;script&gt;** and when the Well Behaved User will get a page with this field, the WBU will *see* <script> (and may get alerted if he read this document ☺) but the HTML source of the page will contain those character sequences and not the <script> string itself. What does this do? Well, it prevents the browser from interpreting the string as a tag.

URLs can be exploited as well, reason for which they would be encoded with the appropriate method, **`Server.URLEncode`**.

In practice, there is more to discuss on this. There isn't a magic bullet and the various options available are discussed more extensively at the links below. Perhaps one more thing to note is that protecting against this vulnerability requires code reviews.

More on this topic:

Understanding Malicious Content Mitigation for Web Developers
http://www.cert.org/tech_tips/malicious_code_mitigation.html

HOWTO: Prevent Cross-Site Scripting Security Issues
http://www.microsoft.com/technet/support/kb.asp?ID=252985

Apache Cross Site Scripting Info
http://www.apache.org/info/css-security

Java Web Server
http://www.sun.com/software/jwebserver/faq/jwsca-2000-02.html

Q253119 HOWTO: Review ASP Code for CSSI Vulnerability
http://support.microsoft.com/support/kb/articles/Q253/1/19.ASP

Q253120 HOWTO: Review Visual InterDev Generated Code for CSSI Vulnerability
http://support.microsoft.com/support/kb/articles/Q253/1/20.ASP

Q253121 HOWTO: Review MTS/ASP Code for CSSI Vulnerability
http://support.microsoft.com/support/kb/articles/Q253/1/21.ASP

2.11 Check the wizard-generated or sample code

Wizards – when available - are nice and handy to learn new things but when it comes to security, check what they do behind the scenes, namely, what the generated code is. It may be possible you'll find hardcoded credentials to access resources such as a database or a directory. Not only is it bad from the security standpoint, but from the development one as well: if the credentials change (for instance, when moving the coding in a production environment), the functionality will break.

Same story with code copied & pasted from samples.

2.12 Middleware security

Most serious web applications would be complex enough so that componentizing them is a must. Whether it's with COM or EJB, this adds a layer of complexity to the [security] architecture.

For the security architect, it raises a few specific issues such as how *authentication*, *authorization* and *impersonation*/*delegation of credentials* work in a distributed environment.

This version of the whitepaper will focus on COM security but we hope to add more material about EJB in the future. In the meantime, keep an eye on the emerging *Java Authentication and Authorization Service* http://java.sun.com/products/jaas/ which adds user-role security to the already existing mechanisms based on code base and signature. Also, Scott Oaks' *Java Security* book published by O'Reilly is an excellent reference into how Java's security mechanisms actually work. Check the book's web site at http://www.oreilly.com/catalog/javasec/, you can download the code or errata.

COM security also is a topic big enough for a book and in fact it is. It's written by *the* man to ask about COM security, Keith Brown from Developmentor. Be sure to check his page http://www.developmentor.com/kbrown/ and http://www.developmentor.com/securitybriefs/ for details on his brand new book, *Programming Windows Security* and also for cool info and utilities to explore the COM world.

To find out how IIS and MTS/COM+ work together to impersonate a client, read the following resources:

http://msdn.microsoft.com/msdnmag/issues/0600/websecure/websecure.asp
http://msdn.microsoft.com/msdnmag/issues/0700/websecure2/websecure2.asp
http://www.asptoday.com/articles/20000224.htm
http://www.asptoday.com/articles/20000302.htm and the backgrounder at http://msdn.microsoft.com/library/techart/msdn_practicom.htm

This last resource has useful tips on the difference between DCOMCNFG and OleView when it comes to setting component security.

2.13 Declarative vs programmatic

Declarative security takes place when the access control is set from outside the application, usually through an administrative interface. *Programmatic* security is the case in which the logic in the code checks the credentials and associated rights. In most cases, web applications will be a mixture of these two methods of enforcing security controls.

When it's available, declarative security is quite useful: file permissions, MTS or database server roles are all examples of this type. They are easy to administer, require no code changes or an understanding of the code for regular operational tasks. Of course, knowing how to apply and integrate them into the whole picture requires a thorough understanding, but once the pieces are in place, daily tasks (such as user and group management) can be delegated to a different team.

Declarative security is good to protect resources between different groups of users (i.e., with different rights). However, when you want a greater granularity, you'll have to use programmatic security. For instance, to distinguish between two users from the same group, permissions and roles are not enough. When you do web banking, the web server can allow anonymous access to some pages and enforce authentication to others, but once the users authenticate, it's the code's task to prevent one user from accessing another's account.

Programmatic security can also help when you need better granularity of controls then what declarative can offer. For instance, with MTS components, you can enforce security on a per-interface level. If you want to have different permissions for some methods within the same interface, however, you'll have to resort to calling ObjectContext's *IsCallerInRole* method. The same story when you want to know more about the security context in which the execution takes place and to distinguish between the original and the direct caller. COM+ is better at delegation and impersonation so, in this context, make sure you know whether the application will run under IIS 4.0 or IIS 5.0

There is no hard and fast rule for when to choose each of the two approaches. The key is to understand where each fits and how you can use better for your purposes.

2.14 PKI is not a silver bullet

For the past few years, each has been touted as the Year of the PKI. Now, PKI is a very cool technology and can do a lot of things, but only if understood and implemented properly.

A common mistake in the web world is to decide to use certificate authentication when there is no PKI in place and no plans to implement certificate management. Because certificates are easy to generate, it may give the wrong impression there's nothing more to worry about. You generate the certificate, install it in the browser and, behold, you have certificate authentication. However checking a certificate's validity or managing certificates is not necessarily a trivial task.

An excellent introduction (and not only) into PKI is *Understanding the Public-Key Infrastructure* (by Carlisle Adams, Steve Lloyd , ISBN: 157870166X). Ellison and Stinger's *Ten Risks of PKI* whitepaper is also a good read, see http://www.counterpane.com/pki-risks.html

Also, make sure you understand the default policies in the different products involved and whether you can customize them enough for your needs.

2.15 Snake oil

The Real World is not necessarily fair and trustworthy and this applies to security software as well. Once in a while, you will find products with larger-than-life claims. "Revolutionary breakthroughs", the cure to all your security concerns, the secret software that gives you 100% security forever without even requiring you to do or understand anything etc. You've got the idea. Why they are bad and how to spot such cases is the subject of a dedicated FAQ: http://www.interhack.net/people/cmcurtin/snake-oil-faq.html (no longer maintained but still a very good read) or of numerous commentaries in the Cryptogram newsletter at http://www.counterpane.com/crypto-gram.html

2.16 Code reviews are your friends

There is no better tool in your arsenal in your search for security holes than a code and architecture review done by a trained eye. For serious applications you should have code reviews anyway, so you can add the security review.

This isn't the place to discuss how reviews should be done, though so we'll leave this item this short.

2.17 Watch what you use

The security of the entire application is dependent on all constituent parts. It is not enough for only the OS and the web server to be secure, all exposed services must be so. What this boils down to is that if you integrate another product into the web application (such as a streaming media or a chat server or any piece someone could connect to, directly or indirectly) you need to understand the risks the new piece adds.

We mentioned the streaming or chat servers because they are becoming more common these days. If these servers can be compromised (e.g., via a classic buffer overflow attack), then the entire application will become so as well. Now, hosting a streaming server on the same machine as the main web server is not a good idea when you take performance into account but even if the machines are different but located on the same network segment, a sniffer installed on the compromised server can gather data from the other, non-compromised machines.

The same principle applies for the main server as well. I prefer to use a server that had security problems in the past which have been fixed (naturally) then an unknown product that has no *reported* vulnerabilities. No news doesn't necessarily mean good news, it can simply indicate that no one bothered to really test the server or if someone did, it hasn't been made public.

If time is on your side, you can try to evaluate the product's resilience to malicious attacks by using tools (such as eEye's Retina or created by yourself) or by reviewing the code (for open source software).

2.18 For troubleshooting, use the logs

If you suspect you're having security problems with your code, check the logs, they may save you a lot of time of "what on earth is it happening?". When the development work starts, make sure you turn logging on and set the level as high as reasonable, it will prove useful later. Below are some Microsoft-specific logs you can use and some typical examples:

The **NT Security eventlog**:

If you're denied access to some resources, you may find that the account used (a real user or the IUSR/IWAM accounts) are either denied access to a resource or are denied the required logon type (interactive vs network). For distributed apps, the credentials' impersonation or delegation can be tricky; check the resources indicated at that section for details.

The **IIS log**

By default IIS only logs several fields but the others are useful as well during development. The easiest way to set extended logging is to do it at the Master site level, thus all sites you'll create in the future will have full logging enabled by default.

The **LDAP log**

This is for Site Server Membership Authentication only. Unfortunately, Site Server does not log failed logons with membership authentication which would be useful when you bind to the directory from code. Workarounds are handling the failed logins from code (which should be done anyway) and using the PerfMon counters to monitor the authentication problems (look for the Site Server Authentication and LDAP Service counters).

2.19 Other pointers

Some resources you may find useful:

http://java.sun.com/security/seccodeguide.html

Georgi Guninski's home page http://www.nat.bg/~joro

David LeBlanc's "Writing Secure Code" columns
http://www.ntsecurity.net/go/loader.asp?id=/columns/seccode/default.asp

security portals:

http://www.securityfocus.com
http://securityportal.com/
http://www.esecurityonline.com/
http://www.ntsecurity.net

mailing lists (SecurityFocus indexes them quite well).