

# **Best Practices for Secure Web Development**

Razvan Peteanu  
razvan.peteanu@home.com

**Revision 3.0**

**Revision Date** September 23, 2000

## Revision History

Version	Release Date	Notes
3.0	September 23, 2000	second public release
2.0	July 18, 2000	first public release
1.0	Feb 2000	first non-public release

## Acknowledgments

Following the publication of version 2.0, the author has received useful comments and contributions from the following people:

Jesús López de Aguilera  
Vittal Aithal  
Toby Barrick  
Gunther Birznieks  
Matt Curtin  
Dig Dug  
Tiago Pascoal  
Dimitrios Petropoulos  
Steve Posick  
Dave Rogers  
Kurt Seifried  
David Spinks  
David A. Wheeler  
David Woods  
Greg A. Woods

This document has and can become better thanks to such feedback. No document can be comprehensive, though. Two excellent documents the reader is encouraged to consult are:

David A. Wheeler's *Secure Programming for Linux and Unix HOWTO* available at <http://www.dwheeler.com/secure-programs/>

Lincoln D. Stein's *The World Wide Web Security FAQ* available at <http://www.w3.org/Security/faq/www-security-faq.html>

# Contents

---

<b>1</b>	<b>WHY?</b> .....	<b>1</b>
<b>1.1</b>	<b>Frequently Asked Questions</b> .....	<b>2</b>
<b>2</b>	<b>FUNDAMENTALS</b> .....	<b>4</b>
<b>2.1</b>	<b>Security as part of the business picture</b> .....	<b>4</b>
<b>2.2</b>	<b>Security as part of the requirements gathering</b> .....	<b>4</b>
<b>2.3</b>	<b>Security as part of the architecture</b> .....	<b>4</b>
<b>2.4</b>	<b>Watch what you use</b> .....	<b>5</b>
<b>2.5</b>	<b>Never trust incoming data. Never</b> .....	<b>5</b>
<b>2.6</b>	<b>When possible, help the user</b> .....	<b>6</b>
<b>2.7</b>	<b>Code reviews are your friends</b> .....	<b>6</b>
<b>2.8</b>	<b>Privacy &amp; law</b> .....	<b>6</b>
<b>2.9</b>	<b>Stay up-to-date!</b> .....	<b>7</b>
<b>2.10</b>	<b>Document, document, document!</b> .....	<b>7</b>
<b>3</b>	<b>TECH DETAILS</b> .....	<b>8</b>
<b>3.1</b>	<b>Don't be anonymous when you won't end up so</b> .....	<b>8</b>
<b>3.2</b>	<b>Do not use more power than you actually need</b> .....	<b>8</b>
<b>3.3</b>	<b>Don't use GET to send sensitive data!</b> .....	<b>9</b>
<b>3.4</b>	<b>Never trust incoming data – details</b> .....	<b>10</b>
<b>3.5</b>	<b>Don't rely on the client to keep important data</b> .....	<b>10</b>
<b>3.6</b>	<b>Don't store sensitive stuff in the *SP page itself</b> .....	<b>11</b>
<b>3.7</b>	<b>Beware of extensions</b> .....	<b>11</b>
<b>3.8</b>	<b>Keep an eye on HTML Comments left in production code</b> .....	<b>12</b>
<b>3.9</b>	<b>Error messages</b> .....	<b>12</b>
<b>3.10</b>	<b>Cross-site scripting</b> .....	<b>12</b>
<b>3.11</b>	<b>Check the wizard-generated or sample code</b> .....	<b>16</b>
<b>3.12</b>	<b>Language &amp; technology specifics</b> .....	<b>16</b>
3.12.1	C/C++ .....	16
3.12.2	Java .....	17
3.12.3	CGI .....	18
3.12.4	Perl .....	18
3.12.5	Unix .....	18
3.12.6	XML.....	19
<b>3.13</b>	<b>Middleware security</b> .....	<b>19</b>
3.13.1	COM/COM+/DCOM .....	19
3.13.2	EJB.....	20
<b>3.14</b>	<b>Declarative vs programmatic</b> .....	<b>20</b>
<b>3.15</b>	<b>Distributed systems and firewalls</b> .....	<b>21</b>
3.15.1	DCOM .....	21
3.15.2	Corba/RMI/IIOP .....	22
3.15.3	SOAP.....	22
<b>3.16</b>	<b>PKI is not a silver bullet</b> .....	<b>23</b>
<b>3.17</b>	<b>Snake oil</b> .....	<b>23</b>
<b>3.18</b>	<b>When randomness matters</b> .....	<b>23</b>
<b>3.19</b>	<b>Use the logs. Create useful logs.</b> .....	<b>24</b>
<b>3.20</b>	<b>SSL</b> .....	<b>24</b>
<b>3.21</b>	<b>Other pointers</b> .....	<b>25</b>

**Legal Notice.**

All names, products and services mentioned are the trademarks or registered trademarks of their respective owners.

Throughout the text, a number of vendors, products or services are listed. This is not an endorsement of the author for the above, but merely pointers of real world examples of issues discussed. Omissions are possible and the author welcomes feedback.

LIMITATION OF LIABILITY. THE AUTHOR WILL NOT BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR PERSONAL INJURY, LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS OR CONFIDENTIAL INFORMATION, LOSS OF PRIVACY, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION IN THIS DOCUMENT, EVEN IF THE AUTHOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Permission is hereby granted to freely distributed this document as long as it is not altered.

**About the author:** Razvan Peteanu lives in Toronto and can be contacted by e-mail at [razvan.peteanu@home.com](mailto: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
- *accountability & 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 FUNDAMENTALS

---

### 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 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 (commercial or crafted by yourself) or by reviewing the code (for open source software).

## 2.5 Never trust incoming data. Never.

You've got to be paranoid if you want to build the application securely. One symptom is to only rely on what you control and even then doubt yourself.

We cannot control what comes from the client's browser (even if we think it comes "back") therefore we must validate everything. Now, in the real world, this level of distrust has various degrees. For instance, it will probably be higher for an Internet site compared to an intranet. Or it will be higher when the stakes are higher (such as with e-commerce sites).

## 2.6 When possible, help the user

The strength of a chain is as good as its weakest link and often in practice the human user is the weakest. We cannot completely fix this with code, but we can help the user make better decision. Perhaps the most typical example is when the user is asked to choose a password. Don't put meaningless limits on them (such as a small length) and consider using password strength validators.

This isn't the only possible application of this recommendation. Help users understand the various settings or decisions they are prompted for that have security implications. A message such as "Do you want to allow this ActiveX object to run?" would not tell much to someone having no idea what ActiveX could be. By providing an explanation about the risks ("selecting yes may allow malicious actions take place") and by preselecting safe (not necessarily convenient!) default values, we can go a long way in preventing problems.

## 2.7 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 trained eyes (the more the better). For serious applications you should have code reviews anyway, so here is a great opportunity to add the security review in. This isn't the place to discuss how reviews should be done, though so we'll leave this item this short.

## 2.8 Privacy & law

Depending on where in the world you live, the Internet may not be that unregulated as it used to be some years ago. Or, if not the Internet itself, the actions you can use it for, no matter what side of the server are you on :-)

This section cannot possibly cover the entire span of computer and Internet law. We will focus on what would matter from an application development standpoint and not on monitoring, protecting from and reacting to intrusions.

Perhaps the most important issue that comes up is the *collection and handling of private data*. The advent of stricter privacy laws makes it an early requirement to identify how customer data will be stored and used on the application side.

For sites having European customers, make sure you check the Data Protection Act (<http://www.dataprotection.gov.uk/>) .

If you plan to use encryption, Bert-Jaap Koops' *Crypto Law Survey* may prove of help. <http://cwis.kub.nl/~frw/people/koops/lawsurvey.htm>

## 2.9 Stay up-to-date!

Security is a changing world and keeping abreast of the developments is a must. Granted, not all vulnerabilities are within the scope of this document (application security) but new ways to exploit a web application are found often enough that subscribing to the vendor's bulletins and to the relevant mailing lists becomes a necessity.

## 2.10 Document, document, document!

The most wonderful security solution is of little value if everything is in its designer's mind. As Bruce Schneier likes to say, "Security is a process, not a product". A process includes the ability of being repeatable and how to ensure correct repeatability unless the steps are documented?

What to include in the documentation? Perhaps the best answer would be anything that you need in order to maintain the same level of security if the system is changed (updated/rebuilt/etc). For an Internet-based app, this means documenting the server and application settings, resource permissions, what the sensitive resources are, and, quite important, how to do things the appropriate way. Imagine for some reason some operations are performed by someone less familiar with the application (say, during vacation time). It's likely that person will not have time to read and understand all processes to follow. Help your team mates.

## 3 TECH DETAILS

---

### 3.1 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.

If deciding what authentication mechanism to use is not easy, you can find a brief overview at <http://www.securityportal.com/research/www-auth/>

### 3.2 Do not use more power than you actually need

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

This section used to be about administrative accounts being [ab]used and their use for common tasks continues to be the most frequent "abuse of power". SQL servers are only part of the picture. Running code as an administrator or suid root is equally inappropriate (unless really needed, of course).

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?

### 3.3 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](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 (and even if you haven't configured any proxy, transparent proxies can still be in 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.

### 3.4 Never trust incoming data – details

It's worth saying it again and in more details. What can constitute incoming data for a web application?

- The HTTP request itself. The URL, the method, the cookie if any, the HTTP headers. Think what could happen if the URL is different (say, if any field passed by the client is changed or if the actual URL requests another page). Could the client see the session of another user? What if the parameters are not consistent with each other? Does the server application handle this case or does it fail, possibly with revealing error messages?
- Data fields (eg form fields). There is so much to do with user supplied data that you'll find this point several times in the sections below. They can overflow buffers (if you are not sure why this is dangerous, see the section on C/C++). If appended to a SQL statement, they can execute code on the SQL server. For a detailed explanation, see Rain Forrest Puppy's article in Phrack 54:  
<http://www.phrack.com/search.phtml?view&article=p54-8>

### 3.5 Don't rely on the client to keep important data

This is a more specific case of the previous section, but worth pointing out. 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 funny (but innocent) example of this flaw is on the web sites of two well-known security companies. They offer a number of whitepapers for download but conditioned by filling in a form with personal details. A quick look in the HTML source shows that the form uses a hidden field to store the page where the visitor is redirected after filling in the form. Thus, a simple copy & paste into the URL bar will bypass the information collection stage.

The previous version of the whitepaper talked about a possible workaround by having the data hashed on the server prior to send it to the client. As some readers pointed out, the description was incomplete. Indeed, simply hashing the data is not enough since, once the algorithm is identified, the client can modify the data and rehash the data. Salting the data with a random value

can solve this, but this means storing session-specific data on the server. Well, if session data is used anyway, it's actually more convenient to store all fields in session variables. Storing just the salt and the hash value uses less memory than a regular form, but the CPU overhead incurred by the two hashing operations means the performance suffers more in this scenario.

The morale of the story is that if you suspect the client might change the data, simply don't rely on it at all, store whatever you need on the server side.

The paper referred to in the previous whitepaper is still worth mentioning because it illustrates another mistake: relying on the HTTP Referer field. See <http://www.webtechniques.com/archives/1998/09/webm/> for details.

### 3.6 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 (including the recent one) has seen a number of holes in web servers that allowed the *source* of an \*SP 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`). More recently, the "*Translate: f*" bug allowed the same outcome.

Similarly, two recent bugs have affected BEA Weblogic (<http://www.foundstone.com/FS-072800-9-BEA.txt>) and IBM WebSphere (<http://www.foundstone.com/FS-072400-6-IBM.txt>).

A different issue but with the same outcome was reported about Allaire's JRun JSP engine (<http://www.allaire.com/handlers/index.cfm?ID=16290&Method=Full>).

Another reason for not hardcoding credentials in the page itself relates to development practices. Such information should be stored in a centralized place preferably in a resource to which access can be audited.

### 3.7 Beware of extensions

An often seen practice is to distinguish included files by using an **.inc** extension on the server side. However, this opens security risks when that extension is not registered to be processed by the server and thus such a file, if known to an attacker who asks for it specifically instead of the including

page, will be served back in all its glory, possibly revealing sensitive information.

### 3.8 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.

### 3.9 Error messages

Server error messages can be revealing and thus disclose information otherwise well protected under normal conditions. What can an error reveal however? A number of things, such as:

- *physical paths*. If an included file is not found, the server may reply with an error stating "include file: c:\inetpub\wwwroot\common.asp not found". The physical path is not a danger by itself, but can reveal information that can be used further in other attacks or can simply give away information about the infrastructure, such as in the case when UNC paths are used.
- *platform architecture*. For instance, an ODBC error message may reveal the database server used. Or the message can contain the exact version of the OS or the CGI/scripting engine, thus helping a malicious party to tune an attack. For a skillful attacker, even indirect information is helpful: if a particular piece of software is older, it may indicate the server is not properly maintained and thus other vulnerabilities are likely to be found out as well. Having detailed information about a platform can also serve in social engineering attacks, especially in large organizations.

The solution is to carefully review the error-related configuration of the server as well as how errors are handled throughout the application. For instance, under IIS you can choose between "Send detailed ASP error message to client" or a generic error (the setting is under a website's Home Directory/Configuration/App Debugging). The first option is the default value, which is not the more secure one.

It would also be better to work with the QA team which systematically goes through the web site anyway. If they find a detailed error message, it can be logged as an issue and followed up accordingly.

### 3.10 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 be 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](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](http://www.microsoft.com/technet/support/kb.asp?ID=252985)

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

[Java Web Server  
http://www.sun.com/software/jwebserver/faq/jwsca-2000-02.html](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](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](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)  
<http://support.microsoft.com/support/kb/articles/Q253/1/21.ASP>

### 3.11 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, even if they are designed to “improve security”. If the author intended them as such doesn’t necessarily mean they are. Learn from samples, but don’t trust them until at least you have understood and analyzed them.

### 3.12 Language & technology specifics

#### 3.12.1 C/C++

The biggest problem with C is also the most frequent application-level attack: C’s inability to detect and prevent improper memory allocation, with the direct result of allowing **buffer overflows**. A great deal of material has been written on this topic but it is still as valid as 20 years ago. The main reason is that prevention of buffer overflows, not being done by the language itself, is left to the programmer to implement. Which means that in the real world this is rarely done. Of course, until the day someone finds out and the vendor scrambles to fix it while the users hope the fix will come before an attacker uses it.

Two excellent papers on buffer overflows are available at [http://www.cultdeadcow.com/cDc\\_files/cDc-351/](http://www.cultdeadcow.com/cDc_files/cDc-351/) and <http://www.securityfocus.com/data/library/P49-14.txt>

A related problem and the subject of recent debates is the **format string attacks** in which a combination of sloppy programming and lack of input data validation leads to the same fatal result. Read more in Tim Newsham’s paper at <http://www.guardent.com/docs/FormatString.PDF> and the thread on Bugtraq.

Preventing these issues can only be done by reviewing the code for insecure practices. The trained eye can be aided by automated tools which scans for known unsafe constructs. Examples of such tools are L0pht’s SLINT <http://www.l0pht.com/slint.html>, ITS4 from <http://www.rstcorp.com/its4/> or LCLint from <http://lclint.cs.virginia.edu/>

Recently there have been efforts to patch the system libraries so that buffer overflows would not be effectual. Having a more robust run-time environment is certainly a good thing, however, *relying* on it instead of preventing the root cause would not be that wise.

### 3.12.2 Java

One of the reasons Java is so popular is the intrinsic security mechanism. Assuming the virtual machine is implemented correctly, malicious language constructs are not possible in Java. For instance, buffer overflows, use of uninitialized variables, invalid opcodes and other vulnerabilities that plague other languages/platforms are stopped by the JVM (assuming a JVM that works *correctly*. Like with any piece of software, there have been bugs in the JVM implementations allowing exploits that got around the mechanisms).

Important note, however: the above paragraph doesn't assert it is impossible to write a malicious Java application. You can certainly can but it usually involves a non-Java factor for the attack to be successful. For instance, if someone installs a Java application without giving consideration whether it should be trusted, then the application would be able to do pretty much anything it pleases (unless sandboxed, but that's not the default configuration).

Having said that, a Java application is still prone to fewer security problems than its C/C++ counterpart. Also, having built-in features that enable the use of security policies and digital signatures is a definite plus for a language.

The core mechanisms supplied in Java are based on the codesource and identity of the code signer, but not on the identity under which the code is run. Filling in this need is the emerging *Java Authentication and Authorization Service* which adds user-role security to the existing code-centric approach. Check <http://java.sun.com/products/jaas/> for details.

Books published on Java security are many but check Scott Oaks' *Java Security* (see <http://www.oreilly.com/catalog/javasec/>, you can download the code and the errata. Also, Li Gong's *Inside Java 2 Platform Security* (ISBN 0201310007).

Sun has published a document on "*Security Code Guidelines*" available online at <http://java.sun.com/security/seccodeguide.html>

A useful resource is David A. Wheeler's briefing on Java Security. You can find it at <http://dwheeler.com/javasec/> David Wheeler also authored the *Secure Programming for Linux and Unix HOWTO* document referenced a few times in this document.

If you are debugging security-related problems and want to go beyond just the exceptions, try running the application with the `-Djava.security.debug` flag. To see what options are available, run

```
java -Djava.security.debug=help
```

You will be able to see the results of CheckPermission calls, loading and granting of policies, a dump of all relevant domains and other info.

Other resources:

JGuru FAQ <http://www.jguru.com/jguru/faq/faqpage.jsp?name=Security>

### 3.12.3 CGI

How to write secure CGI scripts is best described in dedicated FAQs so this section will simply point to the appropriate places:

Lincoln Stein's *World Wide Web Security FAQ*  
<http://www.w3.org/Security/faq/www-security-faq.html>

Selena Sol's <http://stars.com/Authoring/Scripting/Security/>

Paul Phillips' page on CGI Security <http://www.go2net.com/people/paulp/cgi-security/>

Speaking of CGI, using a CGI scanner would be a useful additional check. Have a look at RFP's *whisker* scanner, available at <http://www.wiretrip.net/rfp>. The entire site is a must-see resource for web security.

### 3.12.4 Perl

In the web world, Perl is often used for CGI scripts (the administration uses are not covered by this document) so the previous section is also a good read. For Perl specifics, please refer to the following documents:

Gunther Birznieks' "*CGI/Perl Taint Mode FAQ*"  
<http://www.gunther.web66.com/FAQS/taintmode.html>

Perl CGI FAQ <http://www.perl.com/CPAN-local/doc/FAQs/cgi/perl-cgi-faq.html>

The perlsec documentation pages at <http://www.perl.com/CPAN-local/doc/manual/html/pod/perlsec.html> (or look in your distribution).

### 3.12.5 Unix

This is not a section about Unix security (how could it be just a section when there are books? :-)) but merely a pointer into how to write more secure software running under Unix. Certainly, many of the language-specific sections above are very applicable to Unix. For issues specific to the platform itself, however, please check the following resources:

*Secure Programming for Linux and Unix HOWTO* <http://dwheeler.com/secure-programs/>

*Secure UNIX Programming FAQ* <http://www.whitefang.com/sup/>

*Writing Safe Setuid Programs* <http://olympus.cs.ucdavis.edu/~bishop/secprog.html>

*How to find security holes* <http://www.dnaco.net/~kragen/security-holes.html>

*How to Write Secure Code* <http://www.shmoo.com/securecode/> (provides links to other documents)

### 3.12.6 XML

XML security becomes essential for both B2B and for signing and protecting XML-based forms (which hopefully one day will replace both paper- and HTML-forms). When you talk business, integrity and identity checks are a must for electronic transactions. Setting a standard for digitally signing XML documents and communication is the goal of the joint IETF/W3C *xmldsig* workgroup (see <http://www.ietf.org/html.charters/xmldsig-charter.html> and <http://www.w3.org/Signature/>) Until the standard comes out, there are a number of vendor-specific solutions you can choose from. Some products the author is aware of are those from PureEdge <http://www.pureedge.com/>, Entrust <http://www.entrust.com/xml> or Baltimore Technologies <http://www.baltimore.com/products/xsecure>

## 3.13 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. Please also see the section on distributed architectures and firewalls.

### 3.13.1 COM/COM+/DCOM

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](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.

### 3.13.2 EJB

The EJB specs encourage a separation of duties between the Bean Provider (who is not really concerned with security), the Application Assembler (who assign security roles to the interfaces) and the Deployer who maps security principals to the roles identified by the Assembler.

A presentation of the EJB security model is found in chapter 9 of Sun's "*Designing Enterprise Applications with J2EE*" whitepaper. The document used to be available at [ftp://ftp.java.sun.com/pub/jbp/aspoiduw/jbp-1\\_0\\_1-doc.pdf](ftp://ftp.java.sun.com/pub/jbp/aspoiduw/jbp-1_0_1-doc.pdf) but at the time of writing (mid September) the directory was empty. If you know of an alternative location, please let me know. It is a good read as it focuses more on concepts and less on actual implementations.

A useful resource I found and one that goes more into the real world issues is the recent book *Building Java Enterprise Systems with J2EE* (ISBN: 0672317958). See chapters 24-28.

Details on how the EJB security specs are actually implemented by various vendors can be found in their product documentation, often online. Here are some links to such documentation:

BEA

<http://www.weblogic.com/docs51/classdocs/securityguide.html>

Gemstone

<http://ftp.unidata.ucar.edu/staff/robb/gemstone/ejb/tasks/development/Security3.html> (not Gemstone's site, though, please let me know if you find the direct link)

IBM

<http://www-4.ibm.com/software/webservers/appserv/security.pdf>

Inprise

[http://www.inprise.com/techpubs/books/appserver/appserver40/web/administration/resource\\_concepts.html](http://www.inprise.com/techpubs/books/appserver/appserver40/web/administration/resource_concepts.html)

Oracle

<http://isis.web-eis.com/ows-doc/pdf/security.pdf>

## 3.14 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 than 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 or use COM+. 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.

## 3.15 Distributed systems and firewalls

Any serious web application is inevitably spread across multiple machines. Since these machines communicate, the protocol used and the infrastructure layout are significant security-wise.

Unlike a common development environment where all machines are placed in the same LAN and thus able to communicate freely, a production environment often has the web server isolated in a DMZ with the rest of the servers (database, directory, application) hosted in the internal network. Firewalls/routers would protect both the DMZ from the open Internet *and* the internal LAN from the DMZ and this means the second firewall must be configured in order to allow the traffic required by the piece running on the web server to the rest of the servers in the LAN. Or, to put it from another perspective, the application must be designed so the servers can talk through the existing firewalls.

In the non-mainframe world (with which the author is not familiar), the major protocols used for intermachine communication are DCOM and RMI.

### 3.15.1 DCOM

Microsoft's DCOM is built on top of RPC. We won't discuss here what DCOM is and how to use it. For the security architect, DCOM poses some problems because it is not a firewall-friendly protocol. You can learn details from a paper on exactly this topic, please see "*Using Distributed COM with Firewalls*" at <http://www.microsoft.com/com/wpaper/dcomfw.asp>

At the end of the day, you will still have to open ports that administrators are not comfortable with. Allowing incoming traffic to the RPC port mapper (port 135) is not without risks: using the freely available RPC end point mapper you can learn a number of things about the server being inquired.

*COM Internet Services* (aka *DCOM over HTTP*) does not make all security administrators comfortable. Tunnelling a protocol through another implies a degree of coyness (read makes it harder to monitor intrusions) and may not work with the existing proxy infrastructure.

If you use DCOM, please see the section on COM/DCOM for additional information.

### 3.15.2 Corba/RMI/IIOP

RMI has several methods of connecting across the firewall: directly, tunnelled through HTTP and relayed through a CGI. The first would generally be less appropriate in a secure environment. You can read about the pros and cons of the other in the *RMI ,Servlets and Object Serialization FAQ* available at <http://gene.wins.uva.nl/~nhusein/java.html>

Other useful resources are the official *Joint Firewall Revised Submission* <http://cgi.omg.org/cgi-bin/doc?orbos/98-05-04> with an errata at <http://cgi.omg.org/cgi-bin/doc?orbos/98-07-04> and Rudolf Schreiner's paper on *CORBA Firewalls* available at <http://www.objectsecurity.com/whitepapers/corba/fw/main.html> This website has other documents of interest for CORBA security.

### 3.15.3 SOAP

SOAP is the industry's response to the clash between traditional distributed protocols and Internet security requirements. The result of a group effort by multiple vendors, the protocol has been designed to be firewall-friendly: it works over HTTP by adding a few specific headers (such as `SOAPMethodName`) and by transporting the bulk of the data in an HTTP body (with a content type of `text/xml` or `text/xml-SOAP`, I've found both in different docs).

Alternatively, a SOAP specific HTTP verb: `M-POST`. Both ways allow firewalls that inspect the content to filter the SOAP traffic. The ability of using SSL is also a Good Thing as some environments require an encrypted traffic between a DMZ and the internal LAN.

To find out more, read *Simple Object Access Protocol (SOAP) and Firewalls* available [http://msdn.microsoft.com/xml/general/soap\\_white\\_paper.asp](http://msdn.microsoft.com/xml/general/soap_white_paper.asp)

SOAP 1.0 specifications  
<http://www.msdn.microsoft.com/xml/general/soapspec.asp>

Developmentor's SOAP FAQ  
<http://www.developmentor.com/soap/soapfaq.htm>

### 3.16 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.

### 3.17 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> (the last version is not very recent but it's still a very good read) or of numerous commentaries in *Cryptogram* (<http://www.counterpane.com/crypto-gram.html>)

### 3.18 When randomness matters

Web applications (and not only) use random values for a number of purposes, most often for session or user IDs. Network-savvy readers will also know a similar mechanism is employed by the TCP protocol for the Sequence Numbers.

Why is it important that a session ID is truly random? Well, because from the server's standpoint, the session ID is what distinguishes one client from another. This information must be shared with the client, of course, as part of the query string in the URL or sent in a cookie. A malicious client may try to type a different ID there in order to pretend to be someone else, incidentally having a session at the same time. If the session IDs of the other connected users can be guessed, then the malicious client will probably succeed. This is

where the algorithm used to generate the session IDs becomes essential: if they are generated in a random fashion (and there are degrees of randomness here), then the malicious client has a much harder chance.

Writing a random number generator is not trivial. The first hurdle is that anything generated algorithmically is not random in nature, it can only *appear* random. This is why such algorithmic solutions are called *pseudo*-random number generators (PRNG for short). Most modern languages have a built-in PRNG but this is usually limited to generating the same sequence of values. The user can only change the place in the circular sequence where the values will be retrieved from. For our purposes, this is not secure enough.

In order to overcome the inherent predictability of PRNG algorithms, the use of some external input is needed. For instance, Java's *SecureRandom* class creates a very large number of threads and uses the various timing and other parameters but this incurs a high load on the CPU, not particularly what we want on a web server.

An interesting solution (which also provides true randomness) is offered by <http://www.random.org> where atmospheric noise is used to generate, well, random numbers. The site also provides more information about randomness.

### 3.19 Use the logs. Create useful 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?". Of course, to make use of them you must have logging on (and set to as much as detail as reasonable). If you don't find any relevant entries at all, check whether that particular information is enabled for logging (often full logging is *not* the default configuration for performance reasons) and also whether the application logs that at all. Once in a while you may services that do not have enough logging capabilities.

Which brings us to the second half of this section: in order to help yourself and other debug your application, create meaningful logs. This is a part of the larger issue of providing real help for the user. A mere "Access denied" error doesn't tell much. An "Access denied for user ... while attempting to ... the resource ...." is much better.

### 3.20 SSL

SSL is often misunderstood ("we use SSL therefore we are secure") or over-trusted. SSL is in fact a common term for several protocols (SSLv2, SSLv3 and the TLSv1 standard) of which SSLv2 should not be used anymore. An interesting recent survey regarding the level of security provided by has been published by Eric Murray at the following address: [http://www.meer.net/~ericm/papers/ssl\\_servers.html](http://www.meer.net/~ericm/papers/ssl_servers.html).

Another possible oversight is to leave SSL disabled until the day of going live. The major problem here (apart from some testing issues) is that https

imposes a much higher load on the CPU than normal http. A site that supports 500 concurrent users may find out that if SSL is enabled, the number goes down dramatically. One way of addressing this without multiplying the machines is to use an SSL accelerator, a device that offloads the crypto load from the server's CPU. There are several products out there offering this. If you want to know which to choose and on what criteria, check the following document, it's an excellent intro into the real world of SSL benchmarking: <http://www.intel.com/network/documents/pdf/sslbenchmarking.pdf>

## 3.21 Other pointers

Some resources you may find useful:

Georgi Guninski's home page <http://www.guninski.com/>. George Guninski is a researcher who uncovered numerous vulnerabilities, especially when it comes to client and cross-site scripting.

David LeBlanc's *Writing Secure Code* columns  
<http://www.ntsecurity.net/Articles/Index.cfm?AuthorID=1021>

security portals:

<http://securityfocus.com>  
<http://securityportal.com/>  
<http://www.sans.org/>  
<http://www.esecurityonline.com/>  
<http://www.ntsecurity.net>  
<http://www.infosyssec.net/infosyssec/index.html>