![OWASP - The Open Web Application Security Project - http://www.owasp.org]

The Ten Most Critical Web Application Security Vulnerabilities

January 13, 2003

Commentary

"This 'Ten-Most-Wanting' List acutely scratches at the tip of an enormous iceberg. The underlying reality is shameful: most system and Web application software is written oblivious to security principles, software engineering, operational implications, and indeed common sense."

- **Dr. Peter G. Neumann, Principal Scientist, SRI International Computer Science Lab, Moderator of the ACM Risks Forum, Author of "Computer-Related Risks"**

"This list is an important development for consumers and vendors alike. It will educate vendors to avoid the same mistakes that have been repeated countless times in other web applications. But it also gives consumers a way of asking vendors to follow a minimum set of expectations for web application security and, just as importantly, to identify which vendors are not living up to those expectations"

- **Steven M. Christey, Principal Information Security Engineer and CVE Editor, Mitre**

"The new ROI is security. Companies must be able to provide trusted web applications and web services to their trading partners; who are requiring both secure technology and traditional indicia of financial security, such as insurance. It doesn't stop there though; bad security places a companies' data and applications at risk and hence its viability as a commercial entity. Disappointing your customers is one thing, trying to come back from the loss or corruption of your own systems is quite another."

- **Robert A. Parisi, Jr., Senior VP and Chief Underwriting Officer, AIG eBusiness Risk Solutions**

"The OWASP Top Ten shines a spotlight directly on one of the most serious and often overlooked risks facing government and commercial organizations. A stunning number of organizations spend big bucks securing the network and somehow forget about the applications that have all the keys to the kingdom and full access to all of our sensitive information. These flaws are far too easy to avoid and far too dangerous to ignore."

- **Jeffrey R. Williams, CEO, Aspect Security, Inc.**

"From web pages to back office number crunching, almost all organizations acquire applications code, many people write it, and everybody uses it. But flaws continue to be found in applications, even after nearly fifty years of programming experience. Worse, the same kinds of flaws appear over and over again. This failure to learn from not only our mistakes but also those of our parents' generation creates far too many vulnerabilities for potential attack. It is no wonder that attacks against applications are on the rise.

In compiling this list of the ten most critical applications code flaws, the OWASP has performed a real service to developers and users alike by focusing attention on common weaknesses and what can be done about them. Now it is up to software development organizations, programmers, and users to apply the thoughtful guidance presented here."

- **Dr. Charles P. Pfleeger, CISSP, Master Security Architect, Cable & Wireless, Author of "Security in Computing"**

"The Open Web Application Security Project provides free, open source, commercial quality documentation and software for all those who need it (even if they didn't yet know they need it). OWASP members from all over the world and every background imaginable are sharing information and technology to improve web application security for everyone. The OWASP Top Ten Project was formed to capture our collective wisdom and present it in a way that would bring the attention web application security deserves."

- **Mark Curphey, OWASP Founder and Chair**

"The OWASP Top 10 List is a powerful wake up call for corporate and government IT departments that may have underestimated how vulnerable their assets, buy/sell transactions, and private customer data are to application level attacks. This list is an important first step for evaluating application security risks and exposes the fact that application layer protection is required to block attacks that flow undetected through network level defense systems."

- ***Abhishek Chauhan, Chief Technology Officer, Stratum8 Networks***

"Web developers need to know that the degree to which business applications and customer data are protected from the hostile Internet is directly determined by how securely they've written their code. The OWASP Top Ten list is a great way to understand how to code defensively and avoid the security pitfalls that plague Web applications."

- ***Chris Wysopal, Director of Research & Development, @stake, Inc.***

"The healthcare industry has a critical need to provide secure web applications that protect users' privacy. The OWASP Top Ten will help healthcare organizations evaluate the security of web application products and solutions. Any healthcare organizations using applications that contain these flaws may have difficulty complying with the HIPAA regulations."

- ***Lisa Gallagher, Senior VP, Information and Technology Accreditations, URAC***

"No matter how good you or your security provider are at firewalls, intrusion detection and other security management, such measures cannot protect you from legitimate applications running on your system with flawed code. As a former law enforcement special agent I know the value of "top ten" lists, and this one helps you to catch the bad ones before the violation – a great idea and I encourage wide distribution of this list."

- ***Charles Neal, VP Managed Security Services, Cable & Wireless***

"The OWASP Web Application Security Top 10 underlines what Alphawest have been saying to the Australian market for many years; Security is not just about infrastructure and filtering technologies but about "coding out" application bugs. The OWASP Top Ten highlights the major issues facing organizations today and also, very concisely, details how to avoid these common pitfalls."

- ***Tim Smith, National Security Manager, Alphawest Pty Ltd, Australia***

"Application security considerations are often treated as the domain of specialists, to be applied after coding is done and all other business requirements are satisfied. At Oracle we have found that the task of securing applications is most likely to succeed if security is built in from the start.   Developers must have a basic understanding of security vulnerabilities and how to avoid them, and release managers must explicitly check that security requirements have been satisfied before product can ship. The OWASP Top Ten List is an excellent starting point for companies to build security awareness among their developers, and to prepare security criteria for application deployment."

- ***John Heimann, Director of Security at Oracle Corp.***

"The OWASP has done a great job to give IT professionals a checklist of the most critical vulnerabilities. It is now time for organizations in all industries to start taking application security seriously. Web applications are changing the way business is conducted on a global scale. But if the current security problems inherent in these applications are not addressed, any benefits we gain from them will be soundly negated by attacks."

- ***Yuval Ben-Itzhak, Chief Technology Officer, Founder, KaVaDo Inc.***

"Many people have seen top ten lists detailing the "biggest network security risks". Those lists only point out flaws in third party software that you may or may not be running on your network. Applying the information in this list will keep the software you develop off those other lists!"

- ***John Viega, Chief Scientist, Secure Software Inc., co-author of 'Building Secure Software***

Table of Contents

Introduction

The Open Web Application Security Project (OWASP) is dedicated to helping organizations understand and improve the security of their web applications and web services. This list was created to focus government and industry on the most serious of these vulnerabilities. Web application security vulnerabilities are highly exploitable and the consequence of an attack can be devastating. These vulnerabilities represent an equivalent magnitude of risk as network security problems, and should be given the same degree of attention.

Using this list, organizations can send a message to web site developers that "we want you to make sure that you won't make these mistakes." The security issues raised here are not new. In fact, some have been well understood for decades. Yet for some reason, major software development projects are still making these mistakes and jeopardizing not only their customers' security, but also the security of the entire Internet.

We have chosen to present this list in a format similar to the highly successful SANS/FBI Top Twenty List in order to facilitate its use and understanding. The SANS list is focused on flaws in particular widely used network and infrastructure products. Because each website is unique, the OWASP Top Ten is organized around particular types or categories of vulnerabilities that frequently occur in web applications.

When an organization puts up a web application, they invite the world to send them HTTP requests. Attacks buried in these requests sail past firewalls, filters, platform hardening, and intrusion detection systems without notice because they are inside legal HTTP requests. Even "secure" websites that use SSL just accept the requests that arrive through the encrypted tunnel without scrutiny. **This means that your web application code is part of your security perimeter**. As the number, size and complexity of your web applications increases, so does your perimeter exposure.

This list represents the combined wisdom of dozens of leading application security experts. This document is designed to introduce the most serious web application vulnerabilities. There are many books and guidelines that describe these vulnerabilities in more detail and provide detailed guidance on how to eliminate them. One such guideline is the OWASP Guide available at http://www.owasp.org.

The OWASP Top Ten is a list of vulnerabilities that require immediate remediation. Also, in the longer term, this top ten list is intended to be used by development teams and their managers during project planning and execution. Security professionals should include these items in their reviews and provide guidance on how to remedy these problems. Ultimately, web application developers must achieve a culture shift that integrates security into every aspect of their projects. You wouldn't think of deploying a new Internet connected network without thinking of network security at the design stage or performing network security testing. Project managers should include time and budget for application security activities including developer training, application security policy development, security mechanism design and development, penetration testing, and security code review.

The OWASP Top Ten is a living document. It includes instructions and pointers to additional information useful for correcting these types of security flaws. We will update the list and the instructions as more critical threats and more current or convenient methods are identified, and we welcome your input along the way. This is a community consensus document – your experience in fighting attackers and in eliminating these vulnerabilities can help others who come after you. Please send suggestions via e-mail to topten@owasp.org with the subject "OWASP Top Ten Comments."

OWASP gratefully acknowledges the special contribution from **Aspect Security** for their role in the research and preparation of this document.

Background

The challenge of identifying the "top" web application vulnerabilities is a virtually impossible task. There is not even widespread agreement on exactly what is included in the term "web application security." Some have argued that we should focus only on security issues that affect developers writing custom web application code. Others have argued for a more expansive definition that covers the entire application layer, including libraries, server configuration, and application layer protocols. In the hopes of addressing the most serious risks facing organizations, we have decided to give a relatively broad interpretation to web application security, while still keeping clear of network and infrastructure security issues.

Another challenge to this effort is that each specific vulnerability is unique to a particular organization's website. There would be little point in calling out specific vulnerabilities in the web applications of individual organizations, especially since they are hopefully fixed soon after a large audience knows of their existence. Therefore, we have chosen to focus on the top classes, types, or categories of web application vulnerabilities.

We decided to classify a wide range of web application problems into meaningful categories. We studied a variety of vulnerability classification schemes and came up with a set of categories. Factors that characterize a good vulnerability category include whether the flaws are closely related, can be addressed with similar countermeasures, and frequently occur in typical web application architectures.

To choose the top ten from a large list of candidates has its own set of difficulties. There are simply no reliable sources of statistics about web application security problems. In the future, we would like to gather statistics about the frequency of certain flaws in web application code and use those metrics to help prioritize the top ten. However, for a number of reasons, this sort of measurement is not likely to occur in the near future.

We recognize that there is no "right" answer for which vulnerability categories should be in the top ten. Each organization will have to think about the risk to their organization based on the likelihood of having one of these flaws and the specific consequences to their enterprise. In the meantime, we put this list forward as a set of problems that represent a significant amount of risk to a broad array of organizations. The top ten themselves are not in any particular order, as it would be almost impossible to determine which of them represents the most aggregate risk.

The OWASP Top Ten project is an ongoing effort to make information about key web application security flaws available to a wide audience. We expect to update this document in the six-month timeframe based on discussion on the OWASP mailing lists and feedback to topten@owasp.org.

The Top Ten List

The following is a short summary of the most significant web application security vulnerabilities. Each of these is described in more detail in the following sections.

Top Vulnerabilities in Web Applications		
A1	**Unvalidated Parameters**	Information from web requests is not validated before being used by a web application. Attackers can use these flaws to attack backend components through a web application.
A2	**Broken Access Control**	Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access other users' accounts, view sensitive files, or use unauthorized functions.
A3	**Broken Account and Session Management**	Account credentials and session tokens are not properly protected. Attackers that can compromise passwords, keys, session cookies, or other tokens can defeat authentication restrictions and assume other users' identities.
A4	**Cross-Site Scripting (XSS) Flaws**	The web application can be used as a mechanism to transport an attack to an end user's browser. A successful attack can disclose the end user's session token, attack the local machine, or spoof content to fool the user.
A5	**Buffer Overflows**	Web application components in some languages that do not properly validate input can be crashed and, in some cases, used to take control of a process. These components can include CGI, libraries, drivers, and web application server components.
A6	**Command Injection Flaws**	Web applications pass parameters when they access external systems or the local operating system. If an attacker can embed malicious commands in these parameters, the external system may execute those commands on behalf of the web application.
A7	**Error Handling Problems**	Error conditions that occur during normal operation are not handled properly. If an attacker can cause errors to occur that the web application does not handle, they can gain detailed system information, deny service, cause security mechanisms to fail, or crash the server.
A8	**Insecure Use of Cryptography**	Web applications frequently use cryptographic functions to protect information and credentials. These functions and the code to integrate them have proven difficult to code properly, frequently resulting in weak protection.
A9	**Remote Administration Flaws**	Many web applications allow administrators to access the site using a web interface. If these administrative functions are not very carefully protected, an attacker can gain full access to all aspects of a site.
A10	**Web and Application Server Misconfiguration**	Having a strong server configuration standard is critical to a secure web application. These servers have many configuration options that affect security and are not secure out of the box.

A1 Unvalidated Parameters

A1.1 Description

Web applications use information from HTTP requests to determine how to respond. Attackers can tamper with any part of an HTTP request, including the url, querystring, headers, cookies, form fields, and hidden fields, to try to bypass the site's security mechanisms. Common names for common parameter tampering attacks include: forced browsing, command insertion, SQL injection, cookie poisoning, and hidden field manipulation. Each of these attacks is described in more detail later in this paper. Unless the web application has a strong, centralized mechanism for validating all information from HTTP requests, parameter validation flaws are very likely to exist.

Most web environments support a number of different ways of encoding information. These encoding formats are not like encryption, since they are trivial to decode. Still, developers often forget to decode all parameters to their simplest form before using them. Parameters must be converted to the simplest form before they are validated, otherwise, malicious input can be masked and it can slip past such checks. This process is called "canonicalization." Since almost all HTTP input can be represented in multiple formats, this technique can be used to obfuscate any attack targeting the vulnerabilities described in this document.

A surprising number of web applications use only client-side mechanisms to validate input. Client side validation mechanisms are easily bypassed, leaving the web application without any protection against malicious parameters. Attackers can generate their own HTTP requests using tools as simple as telnet. They do not have to pay attention to anything that the developer intended to happen on the client side. Note that client side validation is a fine idea for performance and usability, but it has no security benefit whatsoever. Server side checks are required to defend against parameter manipulation attacks. Once these are in place, client side checking can also be included to enhance the user experience for legitimate users and/or reduce the amount of invalid traffic to the server.

These attacks are becoming increasingly likely as the number of tools that support parameter "fuzzing", corruption, and brute forcing grows. The impact of using unvalidated parameters should not be underestimated. A huge number of attacks would become difficult or impossible if developers would simply validate information before using it.

A1.2 Environments Affected

All web servers, application servers, and web application environments are susceptible to parameter tampering.

A1.3 Examples and References

- OWASP discussion on Parameter Manipulation: http://www.owasp.org/asac/parameter_manipulation. (Additional links to Cookie, Form, HTTP Header, and URL manipulation discussions can be found on this page)
- OWASP discussion on Canonicalization issues: http://www.owasp.org/asac/canonicalization. (Additional links to Unicode and URL encoding can be found on this page)
- OWASP Guide to Building Secure Web Applications and Web Services, Chapter 9: Data Validation: http://www.owasp.org/guide/

A1.4 How to Determine If You Are Vulnerable

Any part of an HTTP request that is used by a web application without being carefully validated is known as a "tainted" parameter. The simplest way to find tainted parameter use is to have a detailed code review, searching for all the calls where information is extracted from an HTTP request. For example, in a J2EE application, these are the methods in the HttpServletRequest class. Then you can follow the code to see where that variable gets used. If the variable is not checked before it is used, there is very likely a problem. In Perl, you should consider using the "taint" (-t) option.

It is also possible to find tainted parameter use by using tools like Achilles and WebScarab. By submitting unexpected values in HTTP requests and viewing the web application's responses, you can identify places where tainted parameters are used.

A1.5 How to Protect Yourself

The best way to prevent parameter tampering is to ensure that all parameters are validated before they are used. A centralized component or library is likely to be the most effective, as the code performing the checking should all be in one place. Each parameter should be checked against a strict format that specifies exactly what input will be allowed. "Negative" approaches that involve filtering out certain bad input or approaches that rely on signatures are not likely to be effective and may be difficult to maintain.

Parameters should be validated against a "positive" specification that defines:

- Data type (string, integer, real, etc…)
- Allowed character set
- Minimum and maximum length
- Whether null is allowed
- Whether the parameter is required or not
- Whether duplicates are allowed
- Numeric range
- Specific legal values (enumeration)
- Specific patterns (regular expressions)

A new class of security devices known as web application firewalls can provide some parameter validation services. However, in order for them to be effective, the device must be configured with a strict definition of what is valid for each parameter for your site. This includes properly protecting all types of input from the HTTP request, including URLs, forms, cookies, querystrings, hidden fields, and parameters.

The OWASP Filters project is producing reusable components in several languages to help prevent many forms of parameter tampering. OWASP has released CodeSeeker, an application level firewall.

A2 Broken Access Control

A2.1 Description

Access control, sometimes called authorization, is how a web application grants access to content and functions to some users and not others. These checks are performed after authentication, and govern what 'authorized' users are allowed to do. Access control sounds like a simple problem but is insidiously difficult to implement correctly. A web application's access control model is closely tied to the content and functions that the site provides. In addition, the users may fall into a number of groups or roles with different abilities or privileges.

Developers frequently underestimate the difficulty of implementing a reliable access control mechanism. Many of these schemes were not deliberately designed, but have simply evolved along with the web site. In these cases, access control rules are inserted in various locations all over the code. As the site nears deployment, the ad hoc collection of rules becomes so unwieldy that it is almost impossible to understand.

Many of these flawed access control schemes are not difficult to discover and exploit. Frequently, all that is required is to craft a request for functions or content that should not be granted. Once a flaw is discovered, the consequences of a flawed access control scheme can be devastating. In addition to viewing unauthorized content, an attacker might be able to change or delete content, perform unauthorized functions, or even take over site administration.

A2.2 Environments Affected

All known web servers, application servers, and web application environments are susceptible to at least some of these issues. Even if a site is completely static, if it is not configured properly, hackers could gain access to sensitive files and deface the site, or perform other mischief.

A2.3 Examples and References

- OWASP discussion on Path Traversal: http://www.owasp.org/asac/input_validation/pt.shtml
- OWASP Guide to Building Secure Web Applications and Web Services, Chapter 8: Access Control: http://www.owasp.org/guide/

A2.4 How to Determine If You Are Vulnerable

Virtually all sites have access control requirements. Therefore, an access control policy should be clearly documented. Also, the design documentation should capture an approach for enforcing this policy. If this documentation does not exist, then a site is likely to be vulnerable.

The code that implements the access control policy should be checked. Such code should be well structured, modular, and most likely centralized. A detailed code review should be performed to validate the correctness of the access control implementation. In addition, penetration testing can be quite useful in determining if there are problems in the access control scheme.

A2.5 How to Protect Yourself

The most important step is to think through an application's access control and capture it in a web application security policy. Without documenting the security policy, there is no definition of what it means to be secure for that site. The policy should document what types of users can access the system, and what functions and content each of these types of users should be allowed to access. The access control mechanism should be extensively tested to be sure that there is no way to bypass it. This testing requires a variety of accounts and extensive attempts to access unauthorized content or functions.

Some specific access control issues include:

- Insecure Id's – Most large web sites use some form of id, key, or index as a way to reference users, content, or functions. If an attacker can guess these id's they can exercise the access control scheme freely to see what they can access. Web applications should not rely on the secrecy of any id's for protection.

- Forced Browsing Past Access Control Checks – many sites require users to pass certain checks before being granted access to certain URLs that are typically 'deeper' down in the site. These checks must not be bypassable by a user that simply skips over the page with the security check.

- Path Traversal – This attack involves providing relative path information (e.g., "../../target_dir/target_file") as part of a request for information. Such attacks try to access files that are normally not directly accessible by anyone, or would otherwise be denied if requested directly. Such attacks can be submitted in URLs as well as any other input that ultimately accesses a file (i.e., system calls and shell commands).

- File Permissions – Many web and application servers rely on access control lists provided by the file system of the underlying platform. Even if almost all data is stored on backend servers, there are always files stored locally on the web and application server that should not be publicly accessible, particularly configuration files, default files, and scripts that are installed on most web and application servers. Only files that are specifically intended to be presented to web users should be marked as readable using the OS's permissions mechanism, most directories should not be readable, and very few files, if any, should be marked executable.

- Client Side Caching – Many users access web applications from shared computers located in libraries, schools, airports, and other public access points. Browsers frequently cache web pages that can be accessed by attackers to gain access to otherwise inaccessible parts of sites. Developers should use multiple mechanisms, including HTTP headers and meta tags, to be sure that pages containing sensitive information are not cached.

There are some application layer security components that can assist in the proper enforcement of some aspects of your access control scheme. Again, as for parameter validation, to be effective, the component must be configured with a strict definition of what access requests are valid for your site. When using such a component, you must be careful to understand exactly what access control assistance the component can provide for you given your site's security policy, and what part of your access control policy that the component cannot deal with, and therefore must be properly dealt with in your own custom code.

A3 Broken Account and Session Management

A3.1 Description

Account and session management includes all aspects of handling user accounts and active sessions. Authentication is one part of this process, but even solid authentication mechanisms can be undermined by flawed credential management functions, including password change, forgot my password, remember my password, account update, and other related functions. In addition, managing active sessions requires a strong session identifier that can't be guessed, hijacked, or captured.

User authentication on the web typically involves the use of a userid and password. Stronger methods of authentication are commercially available such as software and hardware based cryptographic tokens, biometrics, etc. but such mechanisms are usually cost prohibitive for most web applications. A wide array of account and session management flaws can result in the compromise of user or system administration accounts. Development teams frequently underestimate the complexity of designing an authentication and session management scheme that adequately protects credentials in all aspects of the site.

Web applications must establish sessions to keep track of the stream of requests from each user. HTTP does not provide this capability, so web applications must create it themselves. Frequently, the web application environment provides a session capability, but many developers prefer to create their own session tokens. In any case, if the session tokens are not protected, an attacker can hijack an active session and assume the identity of a user. Creating strong session tokens and protecting them throughout their lifecycle has proven elusive for many developers.

Another major risk in this category is related to backend authentication. Backend authentication is how a web application authenticates itself to backend systems such as databases, directories, and web services. The web application must have access to the proper credentials, which means that if the web server is compromised, the backend systems are at risk. Often, plaintext passwords are embedded in the source code or configuration files. Server flaws can be exploited to view these files.

Many of these flaws are trivial to discover and exploit. The consequences are typically the compromise of a single account. However, in some cases, these flaws can result in the compromise of a backend system, an administrator account, or a technique for gaining access to virtually any user's account. These attacks could be devastating for a web application.

A3.2 Environments Affected

All known web servers, application servers, and web application environments are susceptible to broken account and session management issues.

A3.3 Examples and References

- OWASP discussion on Authentication and Session Management: http://www.owasp.org/asac/auth-session/
- OWASP Guide to Building Secure Web Applications and Web Services, Chapter 6: Authentication and Chapter 7: Session Management: http://www.owasp.org/guide/
- White paper on the Session Fixation Vulnerability in Web-based Applications: http://www.acros.si/papers/session_fixation.pdf
- White paper on Password Recovery for Web-based Applications -- http://fishbowl.pastiche.org/archives/docs/PasswordRecovery.pdf

A3.4 How to Determine If You Are Vulnerable

Both code review and penetration testing can be used to diagnose account and session management problems. Carefully review each aspect of your account mechanisms to ensure that user's credentials are protected at all times, while they are at rest (e.g., on disk), and while they are in transit (e.g., during login). Review every available mechanism for changing a user's credentials to ensure that only an authorized user can change them. Review your session management mechanism to

ensure that session identifiers are always protected and are used in such a way as to minimize the likelihood of accidental or hostile exposure.

A3.5 How to Protect Yourself

Careful and proper use of custom or off the shelf authentication and session management mechanisms should significantly reduce the likelihood of a problem in this area. Defining and documenting your site's policy with respect to securely managing users is a good first step. Ensuring that your implementation consistently enforces this policy is key to having a secure and robust account and session management mechanism. Some critical areas include:

- Password Change Controls: A single password change mechanism should be used wherever users are allowed to change a password, regardless of the situation. Users should always be required to provide both their old and new password when changing their password (like all account information). If forgotten passwords are emailed to users, the system should require the user to reauthenticate whenever the user is changing their e-mail address, otherwise an attacker who temporarily has access to their session (i.e., by walking up to their computer while they are logged in) can simply change their e-mail address and request a 'forgotten' password be mailed to them.

- Password Strength - passwords should have restrictions that require a minimum size and complexity for the password. Users should be required to change them periodically. Users should be restricted to a defined number of login attempts per unit of time and repeated failed login attempts should be logged. Password's provided during failed login attempts should not be recorded, as this may expose a user's password to whoever can gain access to this log. The system should not indicate whether it was the username or password that was wrong if a login attempt fails.

- Password Storage - All passwords must be stored in either hashed or encrypted form to protect them from exposure, regardless of where they are stored. Hashed form is preferred since it is not reversible. Encryption should be used when the plaintext password is needed, such as when using the password to login to another system. Passwords should never be hardcoded in any source code. Decryption keys must be strongly protected to ensure that they cannot be grabbed and used to decrypt the password file.

- Protecting Credentials in Transit - The only effective technique is to encrypt the entire login transaction using something like SSL. Simple transformations of the password such as hashing it on the client prior to transmission provide little protection as the hashed version can simply be intercepted and retransmitted even though the actual plaintext password is not known.

- Session ID Protection – Ideally, a user's entire session should be protected via SSL. If this is done, then the session ID cannot be grabbed off the network, which is the biggest risk of exposure for a session ID. If SSL is not viable, for performance or other reasons, then session IDs themselves must be protected in other ways. First, they should never be included in the URL as they can be cached by the browser, sent in the referrer header, or accidentally forwarded to a 'friend'. Session IDs should be long, complicated, random numbers that cannot be easily guessed. Session IDs can also be changed frequently during a session to reduce how long a session ID is valid. Session IDs must be changed when switching to SSL, authenticating, or other major transitions. Session IDs chosen by a user should never be accepted.

- Account Lists - Systems should be designed to avoid allowing users to gain access to a list of the account names on the site. If lists of users must be presented, it is recommended that some form of pseudonym (screen name) that maps to the actual account be listed instead. That way, the pseudonym can't be used during a login attempt or some other hack that goes after a user's account.

- Browser Caching – Authentication and session data should never be submitted as part of a GET, POST should always be used instead. Authentication pages should be marked with all varieties of the no cache tag to prevent someone from using the back button in a user's browser to backup to the login page and resubmit the previously typed in credentials.

- Trust Relationships – Your site architecture should avoid implicit trust between components whenever possible. Each component should authenticate itself to any other component it is interacting with unless there is a strong reason not to (such as performance or lack of a usable mechanism). If trust relationships are required, strong procedural and architecture mechanisms should be in place to ensure that such trust cannot be abused as the site architecture evolves over time.

- Backend Authentication – Backend authentication credentials should be encrypted and the master key should be stored in a way that makes it very difficult for a hacker to obtain. The use of SSL is recommended for backend connections to minimize the risk from sniffing.

A4 Cross-Site Scripting (XSS) Flaws

A4.1 Description

Cross-site scripting (sometimes referred to as XSS) vulnerabilities occur when an attacker uses a web application to send malicious code, generally JavaScript, to a different end user. When a web application uses input from a user in the output it generates without filtering it, an attacker can insert an attack in that input and the web application sends the attack to other users. The end user trusts the web application, and the attacks exploit that trust to do things that would not normally be allowed. Attackers frequently use a variety of methods to encode the malicious portion of the tag, such as using Unicode, so the request is less suspicious looking to the user.

XSS attacks can generally be categorized into two categories: stored and reflected. Stored attacks are those where the injected code is permanently stored on the target servers, in a database, in a message forum, visitor log, etc. Reflected attacks are those where the injected code takes another route to the victim, such as in an e-mail message, or on some other server. When a user is tricked into clicking on a link or submitting a form, the injected code travels to the vulnerable web server, which reflects the attack back to the user's browser. The browser then executes the code because it came from a 'trusted' server.

The consequence of an XSS attack is the same regardless of whether it is stored or reflected. The difference is in how the payload arrives at the server. Do not be fooled into thinking that a "read only" or "brochureware" site is not vulnerable to serious reflected XSS attacks. XSS can cause a variety of problems for the end user that range in severity from an annoyance to complete account compromise. The most severe XSS attacks involve disclosure of the user's session cookie, allowing an attacker to hijack the user's session and take over the account. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programs, redirecting the user to some other page, and modifying presentation of content. An XSS vulnerability allowing an attacker to modify a press release or news item could change a company's stock price or lessen consumer confidence. An XSS vulnerability on a pharmaceutical site could allow an attacker to modify dosage information resulting in an overdose.

XSS issues can also be present in web and application servers as well. Most web and application servers generate simple web pages to display in the case of various errors, such as a 404 'page not found' or a 500 'internal server error.' If these pages reflect back any information from the user's request, such as the URL they were trying to access, they may be vulnerable to a reflected XSS attack.

The likelihood that a site contains XSS vulnerabilities is extremely high. There are a wide variety of ways to trick web applications into relaying malicious scripts. Developers that attempt to filter out the malicious parts of these requests are very likely to overlook possible attacks or encodings. Finding these flaws is not tremendously difficult for attackers, as all they need is a browser and some time. There are numerous free tools available that help hackers find these flaws as well as carefully craft and inject XSS attacks into a target site.

A4.2 Environments Affected

All web servers, application servers, and web application environments are susceptible to cross site scripting.

A4.3 Examples and References

- OWASP Discussion on Cross Site Scripting: http://www.owasp.org/asac/input_validation/css.shtml
- The Cross Site Scripting FAQ: http://www.cgisecurity.com/articles/xss-faq.shtml
- CERT Advisory on Malicious HTML Tags: http://www.cert.org/advisories/CA-2000-02.html
- CERT "Understanding Malicious Content Mitigation" http://www.cert.org/tech_tips/malicious_code_mitigation.html
- Cross-Site Scripting Security Exposure Executive Summary:
  http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/topics/ExSumCS.asp
- Understanding the cause and effect of CSS Vulnerabilities: http://www.technicalinfo.net/papers/CSS.html

- XSS attacks usually come in the form of embedded JavaScript. However, any embedded active content is a potential source of danger, including: ActiveX (OLE), VBscript, Shockwave, Flash and more.

A4.4 How to Determine If You Are Vulnerable

XSS flaws can be difficult to identify and remove from a web application. The best way to find flaws is to perform a security review of the code and search for all places where input from an HTTP request could possibly make its way into the HTML output. Note that a variety of different HTML tags can be used to transmit a malicious JavaScript. Nessus, Nikto, and some other available tools can help scan a website for these flaws, but can only scratch the surface. If one part of a website is vulnerable, there is a high likelihood that there are other problems as well.

A4.5 How to Protect Yourself

The best way to protect a web application from XSS attacks is to have a detailed code review that searches the code for validation of all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed. The validation should not attempt to identify active content and remove, filter, or sanitize it. There are too many types of active content and too many ways of encoding it to get around filters for such content. We strongly recommend a 'positive' security policy that specifies what is allowed. 'Negative' or attack signature based policies are difficult to maintain and are likely to be incomplete.

Filtering script output can also defeat XSS vulnerabilities by preventing them from being transmitted to users. Applications can gain significant protection by converting the following characters in the generated output (see the XSS FAQ link above for more information):

From:	To:
<	<
>	>
(	(
)	)
#	#
&	&

The OWASP Filters project is producing reusable components in several languages to help prevent many forms of parameter tampering, including the injection of XSS attacks. OWASP has released CodeSeeker, an application level firewall. In addition, the OWASP WebGoat training program has a lesson on encoding.

A5 Buffer Overflows

A5.1 Description

Attackers use buffer overflows to corrupt the execution stack of a web application. By sending carefully crafted input to a web application, an attacker can cause the web application to execute arbitrary code – effectively taking over the machine. Buffer overflows are not easy to discover and even when one is discovered, it is generally extremely difficult to exploit. Nevertheless, attackers have managed to identify buffer overflows in a staggering array of products and components. Another very similar class of flaws is known as format string attacks.

Buffer overflow flaws can be present in both the web server or application server products that serve the static and dynamic aspects of the site, or the web application itself. Buffer overflows found in widely used server products are likely to become widely known and can pose a significant risk to users of these products. When web applications use libraries, such as a graphics library to generate images, they open themselves to potential buffer overflow attacks.

Buffer overflows can also be found in custom web application code, and may even be more likely given the lack of scrutiny that web applications typically go through. Buffer overflow flaws in custom web applications are less likely to be detected because there will normally be far fewer hackers trying to find and exploit such flaws in a specific application. If discovered in a custom application, the ability to exploit the flaw (other than to crash the application) is significantly reduced by the fact that the source code and detailed error messages for the application are normally not available to the hacker.

A5.2 Environments Affected

Almost all known web servers, application servers, and web application environments are susceptible to buffer overflows, the notable exception being Java and J2EE environments, which are immune to these attacks (except for overflows in the JVM itself).

A5.3 Examples and References

- Aleph One, "Smashing the Stack for Fun and Profit", http://www.phrack.com/show.php?p=49&a=14
- Mark Donaldson, "Inside the Buffer Overflow Attack: Mechanism, Method, & Prevention", http://rr.sans.org/code/inside_buffer.php

A5.4 How to Determine If You Are Vulnerable

For server products and libraries, keep up with the latest bug reports for the products you are using. For custom application software, all code that accepts input from users via the HTTP request must be reviewed to ensure that it can properly handle arbitrarily large input.

A5.5 How to Protect Yourself

Keep up with the latest bug reports for your web and application server products and other products in your Internet infrastructure. Apply the latest patches to these products. Periodically scan your web site with one or more of the commonly available scanners that look for buffer overflow flaws in your server products and your custom web applications.

For your custom application code, you need to review all code that accepts input from users via the HTTP request and ensure that it provides appropriate size checking on all such inputs. This should be done even for environments that are not susceptible to such attacks as overly large inputs that are uncaught may still cause denial of service or other operational problems.

A6 Command Injection Flaws

A6.1 Description

Command injection flaws allow attackers to relay malicious code through a web application to another system. These attacks include calls to the operating system via system calls, the use of external programs via shell commands, as well as calls to backend databases via SQL (i.e., SQL injection). Whole scripts written in perl, python, and other languages can be injected into poorly designed web applications and executed. Any time a web application uses an interpreter of any type there is a danger of an injection attack.

Many web applications use operating system features and external programs to perform their functions. Sendmail is probably the most frequently invoked external program, but many other programs are used as well. When a web application passes information from an HTTP request through as part of an external request, it must be carefully scrubbed. Otherwise, the attacker can inject special (meta) characters, malicious commands, or command modifiers into the information and the web application will blindly pass these on to the external system for execution.

SQL injection is a particularly widespread and dangerous form of command injection. To exploit a SQL injection flaw, the attacker must find a parameter that the web application passes through to a database. By carefully manipulating the content of the parameter, the attacker can trick the web application into forwarding a malicious query to the database. These attacks are not difficult to attempt and more tools are emerging that scan for these flaws. The consequences are particularly damaging, as an attacker can obtain, corrupt, or destroy database contents.

Command injection attacks can be very easy to discover and exploit, but they can also be extremely obscure. The consequences can also run the entire range of severity, from trivial to complete system compromise or destruction. In any case, the use of external calls is quite widespread, so the likelihood of a web application having a command injection flaw should be considered high.

A6.2 Environments Affected

Every web application environment allows the execution of external commands such as system calls, shell commands, and SQL requests. The susceptibility of an external call to command injection depends on how the call is made and the specific component that is being called, but almost all external calls can be attacked if the web application is not properly coded.

A6.3 Examples and References

- Examples: A malicious parameter could modify the actions taken by a system call that normally retrieves the current user's file to access another user's file (e.g., by including path traversal "../" characters as part of a filename request). Additional commands could be tacked on to the end of a parameter that is passed to a shell script to execute an additional shell command (e.g., "; rm –r *") along with the intended command. SQL queries could be modified by adding additional 'constraints' to a where clause (e.g., "OR 1=1") to gain access to or modify unauthorized data.
- OWASP discussion on OS Command Injection: http://www.owasp.org/asac/input_validation/os.shtml
- OWASP discussion on SQL Injection: http://www.owasp.org/asac/input_validation/sql.shtml
- OWASP discussion on Meta Characters: http://www.owasp.org/asac/input_validation/meta.shtml
- White Paper on SQL Injection: http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf

A6.4 How to Determine If You Are Vulnerable

The best way to determine if you are vulnerable to command injection attacks is to search the source code for all calls to external resources (e.g., system, exec, fork, Runtime.exec, SQL queries, or whatever the syntax is for making external requests in your language). Note that many languages have multiple ways to run external commands. Developers should

review their code and search for all places where input from an HTTP request could possibly make its way into any of these calls. You should carefully examine each of these calls to be sure that the protection steps outlined below are followed.

A6.5 How to Protect Yourself

The simplest way to protect against command injection is to avoid them where possible. For many shell commands and some system calls, there are language specific libraries that perform the same functions. Using such libraries does not involve the operating system shell interpreter, and therefore avoids a large number of problems with shell commands.

For those calls that you must still employ, such as calls to backend databases, you must carefully validate the data provided to ensure that it does not contain any malicious content. You can also structure many requests in a manner that ensures that all supplied parameters are treated as data, rather than potentially executable content. Most system calls and the use of stored procedures or prepared statements will provide significant protection, ensuring that supplied input is treated as data, which will reduce, but not completely eliminate the risk involved in these external calls. You still must always validate such input to make sure it meets the expectations of the application in question.

Another strong protection against command injection is to ensure that the web application runs with only the privileges it absolutely needs to perform its function. So you should not run the webserver as root or access a database as DBADMIN, or else an attacker can abuse these administrative privileges granted to the web application. Some of the J2EE environments allow the use of the Java sandbox, which can prevent the execution of system commands.

If an external command must be used, any user information that is being inserted into the command should be rigorously checked. Mechanisms should be put in place to handle any possible errors, timeouts, or blockages during the call.

All output, return codes and error codes from the call should be checked to ensure that the expected processing actually occurred. At a minimum, this will allow you to determine that something has gone wrong. Otherwise, the attack may occur and never be detected.

The OWASP Filters project is producing reusable components in several languages to help prevent many forms of command injection. OWASP has released CodeSeeker, an application level firewall.

A7 Error Handling Problems

A7.1 Description

Improper handling of errors can introduce a variety of security problems for a web site. The most common problem is when detailed internal error messages such as stack traces, database dumps, and error codes are displayed to the user (hacker). These messages reveal implementation details that should never be revealed. Such details can provide hackers important clues on potential flaws in the site and such messages are also disturbing to normal users.

Web applications frequently generate error conditions during normal operation. Out of memory, null pointer exceptions, system call failure, database unavailable, network timeout, and hundreds of other common conditions can cause errors to be generated. These errors must be handled according to a well thought out scheme that will provide a meaningful error message to the user, diagnostic information to the site maintainers, and <u>no</u> useful information to an attacker.

Even when error messages don't provide a lot of detail, inconsistencies in such messages can still reveal important clues on how a site works, and what information is present under the covers. For example, when a user tries to access a file that does not exist, the error message typically indicates, "file not found". When accessing a file that the user is not authorized for, it indicates, "access denied". The user is not supposed to know the file even exists, but such inconsistencies will readily reveal the presence or absence of inaccessible files or the sites' directory structure.

One common security problem caused by improper error handling is the fail-open security check. All security mechanisms should deny access until specifically granted, not grant access until denied, which is a common reason why fail open errors occur. Other errors can cause the system to crash or consume significant resources, effectively denying or reducing service to legitimate users.

Good error handling mechanisms should be able to handle any feasible set of inputs, while enforcing proper security. Simple error messages should be produced and logged so that their cause, whether an error in the site or a hacking attempt, can be reviewed. Error handling should not focus solely on input provided by the user, but should also include any errors that can be generated by internal components such as system calls, database queries, or any other internal functions.

A7.2 Environments Affected

All web servers, application servers, and web application environments are susceptible to error handling problems.

A7.3 Examples and References

■ OWASP discussion on generation of error codes: http://www.owasp.org/asac/informational/errors.shtml

A7.4 How to Determine If You Are Vulnerable

Typically, simple testing can determine how your site responds to various kinds of input errors. More thorough testing is usually required to cause internal errors to occur and see how the site behaves.

Another valuable approach is to have a detailed code review that searches the code for the error handling code. The approach should be consistent across the entire site and each piece should be a part of a well-designed scheme. A code review will reveal how the system is intended to handle various types of errors. If you find that there is no organization to the error-handling scheme or that there appear to be several different schemes, there is quite likely a problem.

A7.5 How to Protect Yourself

A specific policy for how to handle errors should be documented, including the types of errors to be handled, and for each, what information is going to be reported back to the user, and what information is going to be logged. All developers need to understand the policy and ensure that their code follows it.

In the implementation, ensure that the site is built to gracefully handle all possible errors. When errors occur, the site should respond with a specifically designed result that is helpful to the user without revealing unnecessary internal details. Certain classes of errors should be logged to help detect implementation flaws in the site and/or hacking attempts.

Very few sites have any intrusion detection capabilities in their web application, but it is certainly conceivable that a web application could track repeated failed attempts and generate alerts. Note that the vast majority of web application attacks are never detected because so few sites have the capability to detect them. Therefore, the prevalence of web application security attacks is likely to be seriously underestimated.

The OWASP Filters project is producing reusable components in several languages to help prevent error codes leaking into user's web pages by filtering pages when they are constructed dynamically by the application.

A8 Insecure Use of Cryptography

A8.1 Description

Most web applications have a need to store sensitive information, either in a database or on a file system somewhere. The information might be passwords, credit card numbers, account records, or proprietary information. Frequently, encryption techniques are used to protect this sensitive information. While encryption has become relatively easy to implement and use, developers still frequently make mistakes while integrating it into a web application. Developers may overestimate the protection gained by using encryption and not be as careful in securing other aspects of the site. A few areas where mistakes are commonly made include:

- Insecure storage of keys, certificates, and passwords
- Improper storage of secrets in memory
- Poor sources of randomness
- Poor choice of algorithm
- Failure to encrypt critical data
- Attempting to invent a new encryption algorithm
- Failure to include support for encryption key changes and other required maintenance procedures

The impact of these weaknesses can be devastating to the security of a website. Encryption is generally used to protect a site's most sensitive assets, which may be totally compromised by a weakness.

A8.2 Environments Affected

Most web application environments include some form of cryptographic support. In the rare case that such support is not already available, there are a wide variety of third-party products that can be added. Only web sites that use encryption to protect information in storage or transit are susceptible to these attacks. Note that this section does not cover the use of SSL, which is covered in the "Server Configuration" section. This section deals only with programmatic encryption of application layer data.

A8.3 Examples and References

- OWASP discussion on common cryptographic flaws: http://www.owasp.org/asac/cryptographic/
- Bruce Schneier, "Applied Cryptography", 2nd edition, John Wiley & Sons, 1995

A8.4 How to Determine If You Are Vulnerable

Discovering cryptographic flaws without access to the source code can be extremely time consuming. However, it is possible to examine tokens, session IDs, cookies and other credentials to see if they are obviously not random. All the traditional cryptanalysis approaches can be used to attempt to uncover how a web site is using cryptographic functions.

By far the easiest approach is to review the code to see how the cryptographic functions are implemented. A careful review of the structure, quality, and implementation of the cryptographic modules should be performed. The reviewer should have a strong background in the use of cryptography and common flaws. The review should also cover how keys, passwords, and other secrets are stored, loaded, processed, and cleared from memory.

A8.5 How to Protect Yourself

The easiest way to protect against cryptographic flaws is to minimize the use of encryption and only keep information that is absolutely necessary. For example, rather than encrypting credit card numbers and storing them, simply require users to re-enter the numbers. Also, instead of storing encrypted passwords, use a one-way function, such as SHA-1, to hash the passwords.

If cryptography must be used, choose a library that has been exposed to public scrutiny and make sure that there are no open vulnerabilities. Encapsulate the cryptographic functions that are used and review the code carefully. Be sure that secrets, such as keys, certificates, and passwords, are stored securely. To make it difficult for an attacker, the master secret should be split into at least two locations and assembled at runtime. Such locations might include a configuration file, an external server, or within the code itself.

A9 Remote Administration Flaws

A9.1 Description

Administrative Interfaces provide powerful features for managing a web application. Such features are frequently used to allow site administrators to efficiently manage users, data, and content on their site. In many instances, sites support a variety of administrative roles to allow finer granularity of site administration. Due to their power, these interfaces are frequently prime targets for attack by both outsiders and insiders.

Common problems that occur in this area include lack of strong authentication and/or encryption for web accessible interfaces, inability to keep lesser administrators restricted to their intended functions, flaws in the separation mechanism between users and administrators, and providing unnecessarily powerful administrative features.

A9.2 Environments Affected

All web servers, application servers, and web application environments that provide an administrative interface.

A9.3 Examples and References

- http://www.infosecuritymag.com/2002/jun/insecurity.shtml

A9.4 How to Determine If You Are Vulnerable

Find out how your website is administrated. You want to discover how changes are made to webpages, where they are tested, and how they are transported to the production server. If administrators can make changes remotely, you want to know how those communications are protected.

Carefully review each interface to make sure that only authorized administrators are allowed access. Also, if there are different types or groupings of data that can be accessed through the interface, make sure that only authorized data can be accessed as well. If such interfaces employ external commands, review their use of such commands to make sure they are not subject to any of the command injection flaws described in this paper.

Since authentication is typically the first line of defense for access to administrative functions, review the parts of your authentication mechanism for administrators even more thoroughly.

A9.5 How to Protect Yourself

The primary recommendation is to never allow administrator access through the front door if at all possible. Given the power of these interfaces, most organizations should not accept the risk of making these interfaces available. If remote administrator access is absolutely required, this can be accomplished without opening the front door of the site. The use of VPN technology could be used to provide an outside administrator access to the internal company (or site) network from which an administrator can access the site through a protected backend connection.

If web based access to such interfaces must be provided, the use of strong authentication such as certificates, token based authenticators, or other technology is highly recommended. Strong authentication should also be considered for backend only access, but the need is not as great. Given the small number of administrators typically required for a site, the costs of using strong authentication should not be prohibitive relative to the risk of compromise of administrative functions.

Regardless of how an administrator accesses the site (through the front or the back), the use of encryption (e.g., VPN or SSL) should be used for the entire administrative session.

Once strong administrator authentication and session protection has been confirmed, the specific rules for which administrators can access which interfaces need to be understood. We recommend that you document exactly which administrator interfaces can be accessed by which administrative roles and any data access rules in your site security policy. Each administrative interface should then be examined to determine that it is protected in a manner that ensures complete enforcement of your documented security policy. You should also consider whether all such administrative interfaces are absolutely necessary. Frequently, there may be other ways of performing similar functions (e.g., via direct access to an underlying database), so providing a web based administrative function may not be absolutely necessary. The elimination of these interfaces can reduce the overall risk to your organization.

As an additional layer of defense, it is recommended that administrative interfaces be separate from interfaces provided to normal users. Such separation can stop cold any attempts by normal users to raise their privileges or pick holes in the access control scheme that separates normal users from administrative functions. These separated applications can be run on the same server using different ports, or even better, on a completely different server. Separating the interfaces in this manner can make it more difficult for an attacker to even find the administrative interfaces, but this is not foolproof. Don't rely on the use of non-standard port numbers or separate servers to provide any real amount of security, as they can be easily scanned for and found. You must still provide the protections recommended above.

Once the normal user and administrative interfaces are separated, another additional defense can be employed: network separation or IP filtering. If access to the administrative interface is only to be allowed from inside the site, then the administrative application can be bound to a separate network card and the network configuration can be set to only allow access from internal IP addresses. If the user and admin applications are run off two different ports on the same card, source IP address checks can restrict which hosts can connect to the admin application port.

Adopting the recommendations above should significantly reduce the risk that any administrative interfaces are exposed to external attack. Unfortunately, many applications today have their user and administrative functions blended together into a single application and no extra precautions are taken to strongly protect the administrative aspects of the application.

A10   Web and Application Server Misconfiguration

A10.1 Description

Web server and application server configurations play a key role in the security of a web application. These servers are responsible for serving content and invoking applications that generate content. In addition, many application servers provide a number of services that web applications can use, including data storage, directory services, mail, messaging, and more.

Frequently, the web development group is separate from the group operating the site. In fact, there is often a wide gap between those who write the application and those responsible for the operations environment. Web application security concerns often span this gap and require members from both sides of the project.

There are a wide variety of server configuration problems that can plague a site. These include:

- Unpatched security flaws in the server software
- Server software flaws or misconfigurations that permit directory listing and directory traversal attacks
- Unnecessary default, backup, or sample files, including scripts, applications, configuration files, and web pages
- Improper file and directory permissions
- Unnecessary services enabled, including content management and remote administration
- Default accounts with their default passwords
- Administrative or debugging functions that are enabled or accessible
- Overly informative error messages (more details in the error handling section)
- Misconfigured SSL certificates and encryption settings
- Use of self-signed certificates to achieve authentication and man-in-the-middle protection
- Use of default certificates

Many of these problems can be detected with readily available security scanning tools. Once detected, these problems can be easily exploited and result in total compromise of a website. Successful attacks can also result in the compromise of backend systems including databases and corporate networks. Having secure software and a secure configuration are both required in order to have a secure site.

A10.2 Environments Affected

All web servers, application servers, and web application environments are susceptible to misconfiguration.

A10.3 Examples and References

- OWASP discussion on vendor security patches: http://www.owasp.org/asac/configuration/patches.shtml
- OWASP discussion on default accounts: http://www.owasp.org/asac/configuration/accounts.shtml
- Web Server Security Best Practices: http://www.pcmag.com/article2/0,4149,11525,00.asp
- Securing Public Web Servers (from CERT): http://www.cert.org/security-improvement/modules/m11.html
- Microsoft Internet Information Server 4.0 Security Checklist: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/chklist/iischk.asp
- A tool to help implement the IIS 4.0 Security Checklist: http://ntbugtraq.ntadvice.com/default.asp?pid=55&did=36

A10.4 How to Determine If You Are Vulnerable

If you have not made a concerted effort to lock down your web and application servers you are most likely vulnerable. Few, if any, server products are secure out of the box. There are a number of scanning products available that will externally scan a web or application server for known vulnerabilities, including Nessus and Nikto. You should run these tools on a regular basis, at least monthly, to find problems as soon as possible. The tools should be run both internally and externally. External scans should be run from a host located external to the server's network. Internal scans should be run from the same network as the target servers.

A10.5 How to Protect Yourself

The first step is to create a hardening guideline for your particular web server and application server configuration. We recommend starting with any existing guidance you can find from your vendor or those available from the various existing security organizations such as OWASP, CERT, and SANS and then tailoring them for your particular needs. The hardening guideline should include the following topics including:

- Configuring all security mechanisms
- Turning off all unused services
- Setting up roles, permissions, and accounts
- Logging and alerts

Once your guideline has been established, use it to configure and maintain your servers. If you have a large number of servers to configure, consider semi-automating or completely automating the configuration process. Use an existing configuration tool or develop your own. A number of such tools already exist. You can also use disk replication tools such as Ghost to take an image of an existing hardened server, and then replicate that image to new servers. Such a process may or may not work for you given your particular environment.

Keeping the server configuration secure requires vigilance. You should be sure that the responsibility for keeping the server configuration up to date is assigned to an individual or team. The maintenance process should include:

- Monitoring the latest security vulnerabilities published
- Applying the latest security patches
- Updating the security configuration guideline
- Regular vulnerability scanning from both internal and external perspectives
- Regular status reports to upper management documenting overall security posture

OWASP will be releasing a series of security configuration guides for Java application servers in early 2003.

Conclusions

OWASP has assembled this list to raise awareness about web application security. The experts at OWASP have concluded that these vulnerabilities represent a serious risk to agencies and companies that have exposed their business logic to the Internet. Web application security problems are as serious as network security problems, although they have traditionally received considerably less attention. Attackers have begun to focus on web application security problems, and are actively developing tools and techniques for detecting and exploiting them.

This top ten list is a starting point. We believe that these flaws represent the most serious risks to web application security, but there are many other security critical areas that were considered for the list and also represent significant risk to organizations deploying web applications. These include flaws in the areas of:

- Unnecessary and Malicious Code
- Broken Thread Safety and Concurrent Programming
- Denial of Service
- Unauthorized Information Gathering
- Accountability Problems and Weak Logging
- Data Corruption
- Broken Caching, Pooling, and Reuse

We welcome your feedback on this Top Ten list. Please participate in the OWASP mailing lists and help to improve web application security. Visit http://www.owasp.org to get started.