WEB APPLICATION SECURITY

February 2008

TABLE OF CONTENTS

SUMMARY

Advances in web technologies coupled with a changing business environment, mean that web applications are becoming more prevalent in corporate, public and Government services today. Although web applications can provide convenience and efficiency, there are also a number of new security threats, which could potentially pose significant risks to an organisation's information technology infrastructure if not handled properly.

For more than a decade, organisations have been dependent upon security measures on the perimeter of the network to protect their IT infrastructure. However, traditional network security measures and technologies may not be sufficient to safeguard web applications from new threats since attacks are now specifically targeting security flaws in the design of web applications. New security measures, both technical and administrative, need to be implemented alongside the development of web applications.

In order to tackle the threats related to these new application services, it is essential to understand the vulnerabilities commonly found in web applications. This article discusses critical web application vulnerabilities and how they can be addressed during different phases of a system development lifecycle. Tips on how to surf the Internet safely are also provided to end-users, as these can be the weakest link in web application information security.

I. MANAGEMENT

EVOLVING TECHNOLOGIES AND THREATS ARE DRIVING CHANGES

Advances in web technologies coupled with a changing business environment, mean that web applications are becoming more prevalent in corporate, public and Government services today. Although web applications can provide convenience and efficiency, there are also a number of new security threats, which could potentially pose significant risks to an organisation's information technology infrastructure if not handled properly.

The rapid growth in web application deployment has created more complex, distributed IT infrastructures that are harder to secure. For more than a decade, organisations have been dependent upon security measures at the perimeter of the network, such as firewalls, in order to protect IT infrastructures. However, now that more and more attacks are targeting security flaws in the design of web applications, such as injection flaws, traditional network security protection may not be sufficient to safeguard applications from such threats.

These threats originate from non-trusted client access points, session-less protocols, the general complexity of web technologies, and network-layer insecurity. With web applications, client software usually cannot always be controlled by the application owner. Therefore, input from a client running the software cannot be completely trusted and processed directly. An attacker can forge an identity to look like a legitimate client, duplicate a user's identity, or create fraudulent messages and cookies. In addition, HTTP

is a session-less protocol, and is therefore susceptible to replay and injection attacks. Hypertext Transport Protocol messages can easily be modified, spoofed and sniffed.

As such, organisations must understand and be fully aware of the threats to properly implement appropriate defensive strategies. Additional security controls, both technical and administrative, may be required to reinforce the protection of vital infrastructure in response to the deployment of web applications.

ADMINISTRATIVE CONTROLS

The following are recommended administrative controls that may help in strengthening the security of web applications and protecting data handled by such applications.

1. Put in place key guidelines to provide direction on the development and maintenance of websites and/or online applications. As an example, the Hong Kong Government has developed a series of guidelines on dissemination of information through government websites.
2. Put in place key guidelines on coding and development practices for web applications. Software development teams should follow a set of secure web application coding practices, designed to combat common web application security vulnerabilities.
3. Collect and manage sensitive information and user data in compliance with policy and regulations.
4. Prepare a security and quality assurance plan, and adopt quality assurance methods such as code review, penetration testing, user acceptance tests, and so on;
5. Perform a complete IT security audit before the final production launch of a web application, and after any major changes or upgrades to the system.

TECHNICAL CONTROLS

Details of important technical measures for securing web applications can be found in the section "Tips on Securing Web Applications" later in this article.

GUIDELINES ON WEB APPLICATION SECURITY

To improve the security of web applications, an open and freely-accessible community called the Open Web Application Security Project (OWASP)[1] has been established to coordinate worldwide efforts aimed at reducing the risks associated with web application software.

A number of major organisations and government departments have also devoted resources themselves to develop strategies, policies and guidelines aimed at managing the risks from the open nature of web applications. To ensure a minimum level of assurance of web application security, some organisations have developed checklists designed to assess overall web application security before final production launch. The US Department of Defence has developed their own Application Security Checklist[2] as one example, designed just for this purpose.

In Hong Kong, the Office of the Government Chief Information Officer (OGCIO) has also disseminated a set of security policies and guideline documents for the reference of

[1] http://www.owasp.org/

[2] http://iase.disa.mil/stigs/checklist/

government departments and organisations. These documents have been published on the OGCIO website at the following URL:

http://www.ogcio.gov.hk/eng/prodev/esecpol.htm

The security considerations and principles of a general application design and development schedule are described in “Section 10.1.1 - Security Considerations in Application Design and Development” of the OGCIO “IT Security Guidelines”. These can also apply to web application development. Since web applications are subject to additional security threats, as described in “Section 10.7 - Web Application Security” of the guideline document, software development teams should follow a set of web application secure coding practices designed to defend against common web application security vulnerabilities.

II. I.T. PRACTITIONERS

This section focuses on areas that need to be observed from a technical perspective, in order to increase the reliability and security of all programs and systems involved.

COMMON VULNERABILITIES IN WEB APPLICATIONS

The Open Web Application Security Project (OWASP) is a worldwide volunteer community aimed at making web application security "visible", so that people and organisations can make informed decisions about application security risks[3]. OWASP lists the most critical web application security flaws in a document entitled "*The Ten Most Critical Web Application Security Vulnerabilities 2007 Update*"[4]:

1. Cross Site Scripting (XSS)

   The potential threat of XSS is allowing the execution of scripts in the victim's browser that could hijack user sessions, deface websites, and possibly introduce worms, etc. This flaw is caused by the improper validation of user-supplied data when an application takes that data and sends it to a web browser without first validating or encrypting the content.

2. Injection Flaws

   The potential threat from this flaw is that an attacker could trick the application into executing unintended commands or into changing system data. Injection

[3] http://www.owasp.org/index.php/Main_Page

[4] https://www.owasp.org/index.php/Top_10_2007#Summary

flaws, particularly SQL injection, are common in web applications. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query.

3. Malicious File Execution

   The potential threat to code vulnerable to remote file inclusion (RFI) is that it could allow attackers the opportunity to include hostile code and data, resulting in devastating attacks, such as a total compromise of the server. Malicious file execution attacks can affect PHP, XML and any framework that accepts filenames or files from users.

4. Insecure Direct Object Reference

   The potential threat here is that attackers could manipulate those references to access other objects without authorisation. A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter.

5. Cross Site Request Forgery (CSRF)

   The potential threat from this flaw is that it might force a logged-on victim's browser to send a pre-authenticated request to a vulnerable web application, which then forces the victim's browser to perform a hostile action to the benefit of the attacker. CSRF can be as powerful as the web application that it attacks.

6. Information Leakage and Improper Error Handling

   The potential threat from this flaw is that attackers can use this weakness to steal sensitive data, or conduct more serious attacks. Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems.

7. Broken Authentication and Session Management

   The potential threat here is that attackers might compromise passwords, keys, or authentication tokens in order to assume the identity of other users. This flaw

is caused when account credentials and session tokens are not properly protected.

8. Insecure Cryptographic Storage

   This potential threat comes when attackers use poorly protected data to conduct identity theft and other crimes, such as credit card fraud. This flaw is due to web applications not making proper user of cryptographic functions to protect data and credentials.

9. Insecure Communications

   This flaw comes from the possible leakage of sensitive information over the network communication infrastructure. This is caused by a failure to encrypt network traffic when it's necessary to protect sensitive communications.

10. Failure to Restrict URL Access

    This flaw gives attackers the opportunity to access and perform unauthorised operations by accessing those URLs directly. This flaw is caused by applications that only protect sensitive functionality when preventing the display of links or URLs to unauthorised users.

Application developers should be aware of these common security flaws and develop programming standards that avoid such problems in the coding phase. A good reference is the OWASP Guide to Building Secure Web Applications[5].

TIPS ON SECURING WEB APPLICATIONS

As mentioned in the previous section of this article, new security risks come with the benefits of deploying web applications. To tackle these risks effectively, various security

[5] http://www.owasp.org/index.php/OWASP_Guide_Project

controls should be considered throughout the entire development lifecycle of the project. To help understand at what point in the lifecycle a recommended security control might be relevant, this section goes through the lifecycle phase by phase and points out key security concerns that require special attention.

The Requirement Stage

At this stage, the application development team should gather together all the system and security specifications required by the various parties involved in the project. The system requirements should provide the development team with an overview on the core purpose of the application, including what the application should do and what it should not do. This information will help the development team in defining key security controls for the application.

In addition, certain security controls or mechanisms are required to be built into the application in order to comply with regulations or requirements. For example, the Payment Card Industry Data Security Standard (PCI DSS) Requirement 6, entitled "Develop and Maintain Secure Systems and Applications", focuses on the establishment of controls that minimise the existence of security vulnerabilities in systems and software. It specifies requirements for secure software development and attack protection.

Correctly establishing system and user security requirements will be vital in driving the design, development and testing stages, as this will increase the overall security of the web application, and ensure greater user satisfaction with the end result.

The Design Stage

The design stage involves not only design of the application in accordance with the specifications outlined in the first stage, but also defining secure coding standards, performing threat modelling, and developing a security architecture for the application.

Define Secure Coding Standards

Secure coding standards are guidelines on how developers should write the code for the application, and should include guidelines for developing secure code and for proper commenting to identify high-risk areas, data input and other interface and error handling. A number of secure coding practices have been suggested by various organisations, including OWASP and CERT[6]. The OGCIO has also included a set of common secure coding practices in their IT Security Guidelines:

1. Validate all input parameters to prevent attacks such as SQL injection and cross-site scripting attacks:

   Programmers should develop a centralised module to perform input parameter validation, and check each input parameter against a strict format that specifies exactly which types of input will be allowed. Special characters such as "~!#$%^&*[]<>'\r\n" coming from the input form should be filtered, or replaced with an escape sequence. Client-side scripts should not be relied on to perform the necessary validation checks.

   The application should only accept data containing a strictly limited and expected set of characters. If a number is expected, only digits should be accepted. If a word, only letters should be allowed. Input data should also be validated for the proper format. If an email address is expected, only letters, numbers, the "at" (@) symbol, dashes, and dots in the proper arrangement should be accepted. Enforcing minimum and maximum length restrictions on all incoming data should also be included. This technique should be used for account numbers, session credentials,

[6] http://www.cert.org/secure-coding/

usernames, and so on. All these techniques restrict the number of potential entry points for incoming attacks.

2. Sanitised application response

   A centralised module should be developed to perform any sanitisation. All output, return codes and error codes from calls (e.g. calls to the backend database) should be checked to ensure that the processing expected actually occurred. For example, unnecessary internal system information such as internal IP addresses, internal host names, internal directory structures, verbose error messages generated by internal server errors during a response should not be exposed on the client side. Most application/web servers allow the generation of a custom error page should an internal server error occur.

3. HTTP trust issues

   Programmers should not trust or rely on HTTP REFERER headers, form fields or cookies to make security decisions, as this type of data can be spoofed. Unless strong cryptographic techniques are used to verify the integrity of HTTP headers, do not trust these parameters coming in from a client browser. In addition, do not assume hidden parameters cannot be changed by the user, as hidden parameters can be easily manipulated by attackers.

4. Keep sensitive session values on the server to prevent client-side modification

   Do not put sensitive information in any client browser cookies. If sensitive values have to be stored in a client browser, strong cryptographic techniques should be used to protect the confidentiality and integrity of the data.

5. Encrypt pages containing sensitive information and prevent caching

   Pages containing sensitive information should be encrypted with proper algorithms and keys such as SSL and TLS during transmission. Use signed Java applets or ActiveX to acquire and display sensitive information, and set the appropriate HTTP header attributes to prevent caching, by browser or proxy, of an individual page should that page contain sensitive information.

6. Session management

   A session ID should be long, complicated, contain random numbers that are unpredictable, and it should be changed frequently during a session to reduce the duration that a session ID remains valid. In addition, a session ID should not be stored in a URL, persistent cookies, hidden HTML fields or HTTP headers. Programmers can consider storing session IDs in a client browser's session cookies. By employing SSL or TLS, session IDs can be protected from sniffing by attackers. A logout function for the application, and an idle session timeout should also be implemented. When logging off a user or timing-out an idle session, not only should the client-side cookie should be cleared (if possible), but also the server side session state for that browser plus connections to backend servers should be cleaned up.

7. Access restriction

   Ensure that the end-user account only has specific privileges to access those functions that they are authorised to access, restricting access to the backend database, or to run SQL commands, and OS commands. When an application makes system calls to access certain programs, do not make calls to actual file names and directory paths. If attackers have access to the source code, they may uncover system-level information. Use mapping provided by the web server as a filtering layer

8. Build a centralised module for application auditing and reporting.

9. Use the most appropriate type of authentication method for the job, to identify and authenticate incoming user requests.

Perform Threat Modelling

Building a secure application requires an understanding of the threats against that application. A threat modelling process helps to identify threats, attacks, vulnerabilities,

and countermeasures in the context of your application scenario. Threat modelling can be accomplished through these steps:

Step 1: Identify the key security objectives.

Step 2: Create an overview of the application by itemising the important characteristics of that application.

Step 3: Deconstruct the application to identify the features and modules that have a security impact, and that need to be evaluated.

Step 4: Identify all threats

Step 5: Identify all vulnerabilities.

Design Web Application Security Architecture

A typical web application architecture contains 3 tiers, separating the externally-facing web server from the internal application server and database server. With a tier-based architecture such as this, even if an attacker compromises an externally-facing web server from the outside, they still have to find ways to gain access and attack the internal network. This is the principle of defence-in-depth protection. Defence-in-Depth is a practical approach to information security. The fundamental concept always centres on the idea of multiple layers of security to protect vital assets. Layers of security include input validation, database layer abstraction, server configuration, proxies, web application firewalls, data encryption, OS hardening, and so on.

The Development Stage

This is one of the most important stages in terms of mitigating security issues within the code. Observing secure coding standards certainly helps improving security and reducing the number of common mistakes that result in security breaches. In addition, performing

security risk assessments during the development stage also helps to identify the security controls required.

The Testing and Quality Assurance Stage

Before any application is launched for production, the need for comprehensive testing is paramount. In addition to user acceptance tests, there are others, such as system tests, stress tests, regression tests and unit tests that are useful in validating the performance and accuracy of system functionalities. This section describes some of the tests that can be carried out in order to increase the reliability and security of the program/systems being developed.

Web Application Unit Testing

Web application unit testing is an important part of the development stage, designed to identify the vulnerabilities in a web application. Unit testing involves the testing of individual programs or modules to ensure that all internal operations of a program or module perform according to specifications. Unit testing should include tests for common security issues, such as buffer overflows, and is especially important if the module is being integrated into a "build" with other components. If no unit tests are carried out, it becomes very hard to implement an automated security testing process in the middle of the development stage.

There are a variety of tools that can help find and eliminate web application vulnerabilities, but it is important to note that these tools can only cover a small fraction of the testing needed for an effective application security program. Relying only on tools rather than focusing on improving the software development life cycle leads to a false sense of security, as automated scanning tools are limited in the types of vulnerabilities they can uncover and identify.

Code Review

A peer review process, undertaken by walking through the source code, can help identify security flaws, ensure adherence to security development standards, and ensure consistency with the overall program design. Usually, development managers, system administrators and database administrators would be present to examine the workings of the source code of the application, and improvements can be suggested and recommended by all parties. In addition, by walking through the source code, hidden or secure content, such as keys and passwords, can be identified and the adequacy of protective measures thoroughly evaluated.

A number of automated code scanning tools are available in the market that may help offload some of the code-walkthrough burden. However, as with web application scanning tools, these tools may only be able to identify common errors, but not more complex security issues. These tools should therefore not be a substitute for human analysis.

Before any code review begins, the project team should define which areas of the code are considered high risk and likely to lead to a vulnerability, exposure or weakness. In general, any aspects of the code that provide access control, configuration management, auditing, logging, authentication or interface with third-party components or the operating system should be reviewed. This exercise is generally referred to as threat modelling or risk analysis and consists of analysing various aspects of the design to determine which areas of the project should be part of a security code review.

The Pre-Production Stage

An IT security audit should be performed before the production launch and after any major changes to the system. Each vulnerability fix requires updates to custom code, and it follows that each repair requires a code push that could introduce a new vulnerability. Therefore, it is imperative to continuously assess the impact of each fix to maintain secure applications.

The Maintenance and Support Stage

Security is an on-going process. Security issues may still be found after the application is released to the public. Protection and detection mechanisms should be put in place to ensure the application is running securely and smoothly. Key ongoing security measures can include the following:

Application Log Review

To detect any anomaly in a web application, a log review is indispensable. Many web servers support detailed logs that capture all web requests made to the web application. By regularly reviewing the web access log and studying web requests made to the application, it is possible to gain significant insight into the safety of the web application. If abnormal URLs are uncovered, it is very likely that some kind of web attack have taken place.

In addition, application owners can also request the implementation of application audit trails for the web application. Exception reports to undefined requests, or abnormal transactions should be generated and reviewed regularly.

Version Control and a Separate Environment for Development

The integrity of an application should be maintained with appropriate security controls such as a version control mechanism and the separation of environments for development, system testing, acceptance testing, and live operation. The production and development environments should be kept synchronised. Application development staff should not be permitted to access production information unless absolutely necessary.

Web Application Firewalls

Standard firewalls can help restrict or permit network access to network ports authorised by the organisation. Although application proxy firewalls exist, they cannot understand the specific content of all web applications being run by the organisation. According to the Web Application Security Consortium, a web application firewall (WAF) is "*an intermediary device, sitting between a web-client and a web server, analysing OSI Layer-7 messages for violations in the programmed security policy*"[7].

Usually, web application firewalls are installed in front of a web server. Like standard firewalls, these can be software or hardware based, but always with the purpose of protecting the web server from attack. There are two protection approaches:

1. Signature based: The WAF identifies attacks by checking web request against an "attack signature" file.
2. Abnormal behaviour based: The WAF identifies attacks by detecting abnormal traffic patterns.

[7] http://www.webappsec.org/projects/glossary/

A CHECKLIST FOR WEB APPLICATION ACCEPTANCE

Upon acceptance of the web application, an independent security assessment should be conducted to ensure full compliance with company policy or project security requirements by assessing the web application as well as the source code. This assessment is an essential component for all web application projects that may be outsourced to external development houses. Test cases on security controls required in the project initiation phase should also be included in the User Acceptance Test.

Security should be considered as early as possible in the project initiation phase. Security expectations and requirements — in particular, requirements on the authentication mechanisms, input validation and audit trails — must be communicated to development vendors.

The following are some examples of areas that might be examined in an assessment of web application security:

Identification and Authentication

1. How are users and processes authenticated?
2. Is the authentication process implemented in accordance with specifications and in compliance with the security policy of the organisation?
3. If the authentication is based on passwords, how are the user passwords being handled and stored? Is the password handling mechanism in compliance with the security policy of the organisation? Are there any hard-coded passwords or keys embedded in the program source?
4. Is the application required to authenticate each and every session?

Data Protection

1. Is the data protection mechanism implemented in accordance with the security policy of the organisation?
2. Is all data protected adequately at rest? Is all data protected adequately in transit?
3. If encryption is used, how is the encryption handled? Does encryption handling comply with the overall security policy of the organisation?

Logging

1. Is the audit trail logging mechanism implemented in accordance with specifications?
2. Are the application audit records vulnerable to unauthorised deletion, modification or disclosure?

Error Handling

1. How are error messages handled? Is there any chance of an information leak that could be utilised in a subsequent attack? Would an application failure result in the system entering an insecure state?

Operation

1. Are segregation of duties and least privilege principles enforced?

2. Have all built-in user IDs, testing user IDs, and IDs with default passwords been removed from the operating system, web servers and application itself before final production launch?
3. Are the system administration procedures, change management procedures, disaster recovery procedures, and backup procedures fully and clearly defined?

It must be emphasised that this checklist is not exhaustive. Depending on the security requirements and specific nature of the target web application, additional test cases or checking criteria should be included according to specific needs.

In addition, when any information system is outsourced to third party service provider, proper security management processes must be in place to protect data as well as to mitigate the security risks associated with outsourced IT projects/services. It is recommended that the reader should refer to the "IT Outsourcing Security" Section for more information on the security considerations specific to outsourcing.

APPENDIX I: HOW TO PROTECT YOURSELF WHILE SURFING THE INTERNET

While enjoying the convenience of modern web applications and application services, end-users should take active steps to protect themselves. In many transactional web applications, it is not uncommon for customers to sign or agree to a 'Terms and Conditions' statement, in which customers agree that the web application provider not be liable for any loss or damage that may occur due to any security compromise of the customer's account.

Common safeguards for end-users:

1. Don't login to critical web applications from a public computer.
2. Don't cache your username and password in the workstation.
3. Remember to logoff at the end of a session.
4. Use different sets of logins and passwords for different web applications and services.
5. Regularly change your passwords used in critical web applications if a one-time password is not supported.
6. Report abnormal behaviour to the service provider immediately.
7. Ensure that the operating system and system components like Internet Explorer (browser) are fully patched and up to date.
8. Install a personal firewall as well as anti-virus software with latest virus signatures. Any anti-virus software should be good enough to detect malware such as keyloggers.

9. Don’t download software or plug-ins from unknown sources.

APPENDIX II: HOW TO ELIMINATE THE "TOP TEN" SECURITY VULNERABILITIES IN YOUR CODE

The OWASP Top Ten Project has analysed the top ten most critical security vulnerabilities in web applications. The project team not only provides details of key vulnerabilities but also provides recommendations on how to eliminate these vulnerabilities from code. The following is an excerpt of the recommendations on the OWASP website:

1. Use a standard input validation mechanism to validate all input data.
2. Strong output encoding.
3. Specify the output encoding (such as ISO 8859-1 or UTF 8).
4. Do not use "blacklist" validations to detect XSS in input or to encode output.
5. Watch out for canonicalisation errors.
6. Use strongly typed parameterised query APIs.
7. Enforce least privilege.
8. Avoid detailed error messages.
9. Show care when using stored procedures.
10. Do not use dynamic query interfaces.
11. Do not use simple escaping functions.
12. Use an indirect object reference map.
13. Use explicit taint checking mechanisms.
14. Add firewall rules to prevent web servers making new connections to external websites and internal systems.
15. Check user-supplied files or filenames.

16. Consider implementing a “chroot jail”.
17. Avoid exposing your private object references to users whenever possible.
18. Validate any private object references extensively with an "accept known good" approach.
19. Verify authorisation to all referenced objects.
20. Insert custom random tokens into every form and URL.
21. For sensitive data or value transactions, re-authenticate or use transaction signing.
22. Do not use GET requests (URLs) for sensitive data or to perform value transactions.

For the details, refer to the OWASP Top Ten Project Theme Page of the OWASP website. (http://www.owasp.org/index.php/OWASP_Top_Ten_Project)