

ABSTRACT

This paper covers that how the increase in the usage of web applications is directly related to an increase in the number of security incidents for them.

The paper also discusses the Web application security, explains common security terminology and presents a set of proven security principles upon which many of the recommendations throughout this paper are based. It presents an overview of the security process and explains why a holistic approach to security that covers multiple layers including the network, host and application, is required to achieve the goal of hack-resilient Web applications. A hack-resilient application is one that reduces the likelihood of a successful attack and mitigates the extent of damage if an attack occurs. A hack-resilient application resides on a secure host (server) in a secure network and is developed using secure design and development guidelines.

INTRODUCTION

Web applications are becoming more prevalent and increasingly more sophisticated, and as such they are critical to almost all major online businesses. As with most security issues involving client/server communications, Web application vulnerabilities generally stem from improper handling of client requests and/or a lack of input validation checking on the part of the developer.

The very nature of Web applications - their ability to collate, process and disseminate information over the Internet - exposes them in two ways.

First and most obviously, they have total exposure by nature of being publicly accessible. This makes security through obscurity impossible and heightens the requirement for hardened code. Second and most critically from a testing perspective, they process data elements from within HTTP requests - a protocol that can employ a myriad of encoding and encapsulation techniques.

Most Web application environments expose these data elements to the developer in a manner that fails to identify how they were captured and hence what kind of validation and sanity checking should apply to them. Because the Web "environment" is so diverse and contains so many forms of programmatic content, input validation and sanity checking is the key to Web applications security. This involves both identifying and enforcing the valid domain of every user-definable data element, as well as a sufficient understanding of the source of all data elements to determine what is potentially user definable.

Today, web application security is finally getting more prominent attention. This attention comes with the benefit of it being addressed as a higher priority now, but with the drawback of still being in an emerging area of technology. The current use of most web application security testing tools is still focused on the penetration tester/information security professional, with use being extended for QA and audit professionals. We are still a fair distance from holding a developer (i.e., software vendors) accountable for writing insecure code, but clearly the trend is moving in that direction. Security has always been a holistic solution, requiring all players and systems to work in concert to form a good defense.

Understanding Web Application & It's Security

When you talk about Web application security, there is a tendency to immediately think about attackers defacing Web sites, stealing credit card numbers, and bombarding Web sites with denial of service attacks. You might also think about viruses, Trojan horses, and worms. These are the types of problems that receive the most press because they represent some of the most significant threats faced by today's Web applications.

These are only some of the problems. Other significant problems are frequently overlooked. Internal threats posed by rogue administrators, disgruntled employees, and the casual user who mistakenly stumbles across sensitive data pose significant risk. The biggest problem of all may be ignorance.

The solution to Web application security is more than technology. It is an ongoing process involving people and practices. ***"If you do not know your threats, how can you secure your system?"***

Organizations need to find methods for achieving their mission goals in using the Internet and at the same time keeping Sending information over the internet, whilst preserving its integrity is becoming a must. The general understanding which is prevalent that having a firewall secures a company’s network can be held true to some extent. However, how can a firewall protect against an internally-initiated LAN attack? Or what is the likelihood that secure login credentials were not intentionally compromised to make it seem like an accident?

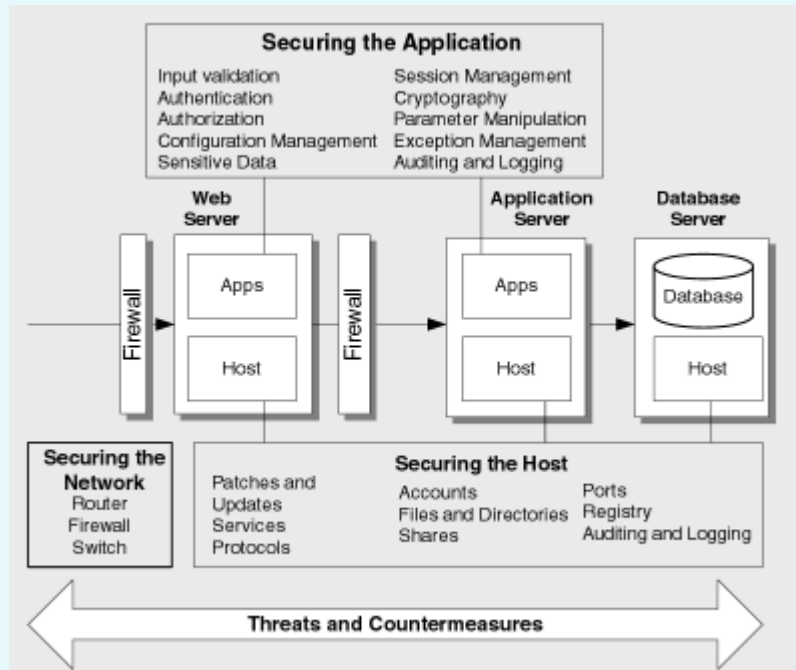


Figure 1

A Web application is an application, generally comprised of a collection of scripts that reside on a Web server and interact with databases or other sources of dynamic content. They are fast becoming ubiquitous as they allow service providers and their clients to share and manipulate information in an (often) platform-independent manner via the infrastructure of the Internet. Some examples of Web applications include search engines, Webmail, shopping carts and portal systems.

Web application security must be addressed across the tiers and at multiple layers. A weakness in any tier or layer makes your application vulnerable to attack.

If we have a Firewall, Are we “Really” Secured?

This is a common misconception; it depends on the threat. For example, a firewall may not detect malicious input sent to your Web application. Also, consider the scenario where a rogue administrator has direct access to your application.

Do firewalls have their place? Of course they do. Firewalls are great at blocking ports. Some firewall applications examine communications and can provide very advanced protection. Firewalls are an integral part of your security, but they are not a complete solution by themselves.

The same holds true for Secure Sockets Layer (SSL). SSL is great at encrypting traffic over the network. However, it does not validate your application's input or protect you from a poorly configured server.

Preventing from Web Application Security Threats

It is not possible to design and build a secure Web application until you know your threats. An increasingly important discipline and one that is recommended to form part of your application's design phase is threat modeling. The purpose of threat modeling is to analyze your application's architecture and design and identify potentially vulnerable areas that may allow a user, perhaps mistakenly, or an attacker with malicious intent, to compromise your system's security.

After you know your threats, design with security in mind by applying timeworn and proven security principles. As developers, you must follow secure coding techniques to develop secure, robust, and hack-resilient solutions. The design and development of application layer software must be supported by a secure network, host, and application configuration on the servers where the application software is to be deployed.

Web applications typically interact with the user via FORM elements and GET or POST variables (even a 'Click Here' button is usually a FORM submission). With GET variables, the inputs to the application can be seen within the URL itself, however with POST requests it is often necessary to study the source of form-input pages (or capture and decode valid requests) in order to determine the users inputs.

As a tester you must use all input methods available to you in order to elicit exception conditions from the application. Thus, you cannot be limited to what a browser or automatic tools provide. It is quite simple to script HTTP requests using utilities like curl, or shell scripts using net cat. The process of exhaustive black box testing a Web application is one that involves exploring each data element, determining the expected input, manipulating or otherwise corrupting this input, and analyzing the output of the application for any unexpected behavior.

"A vulnerability in a network will allow a malicious user to exploit a host or an application. A vulnerability in a host will allow a malicious user to exploit a network or an application. A vulnerability in an application will allow a malicious user to exploit a network or a host."

Carlos Lyons, Corporate Security, Microsoft

By using the hack-resilient applications we can reduce the threat of Web Application's attack. The three-layered approach that uses: securing the network, securing the host, and securing the application, you can minimize the attack on web applications. Hack-resilient application also include the process called threat modeling, which provides a structure and rationale for the security process and allows you to evaluate security threats and identify appropriate countermeasures.

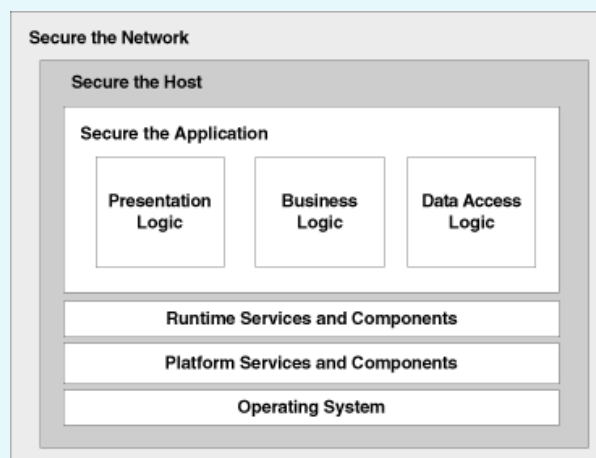


Figure 2

Categories of Application Vulnerability

What better way to measure the security of a system than to evaluate its potential weak points? To measure the security resilience of your application, you can evaluate the application vulnerability categories. When you do this, you can create application security profiles, and then use these profiles to determine the security strength of an application.

These categories are used as a framework throughout this guide. Because the categories represent the areas where security mistakes are most frequently made, they are used to illustrate guidance for application developers and architects. The categories are also used as a framework when evaluating the security of a Web application. With these categories, you can focus consistently on the key design and implementation choices that most affect your application's security.

Input validation is the root of the issues and can be difficult to locate in a large code base with lots of user interactions, which is the main reason that developers employ penetration testing methodologies to expose these problems. Web applications are, however, not immune to the more traditional forms of attack. Poor authentication mechanisms, logic flaws, unintentional disclosure of content and environment information, and traditional binary application flaws (such as buffer overflows) are rife. When approaching a Web application as a tester, all this must be taken into account and a methodical process of input/output or "black box" testing must be applied.

Application vulnerability categories are described in the table below.

Category	Description
Input Validation	How do you know that the input that your application receives is valid and safe? Input validation refers to how your application filters, scrubs, or rejects input before additional processing.
Authentication	"Who are you?" Authentication is the process where an entity proves the identity of another entity, typically through credentials, such as a user name and password.
Authorization	"What can you do?" Authorization is how your application provides access controls for resources and operations.
Configuration Management	Who does your application run as? Which databases does it connect to? How is your application administered? How are these settings secured? Configuration management refers to how your application handles these operational issues.
Sensitive Data	Sensitive data refers to how your application handles any data that must be protected either in memory, over the wire, or in persistent stores.
Session Management	A session refers to a series of related interactions between a user and your Web application. Session management refers to how your application handles and protects these interactions.
Cryptography	How are you keeping secrets, secret (confidentiality)? How are you tamperproofing your data or libraries (integrity)? How are you providing seeds for random values that must be cryptographically strong? Cryptography refers to how your application enforces confidentiality and integrity.
Parameter Manipulation	Form fields, query string arguments, and cookie values are frequently used as parameters for your application. Parameter manipulation refers to both how your application safeguards tampering of these values and how your application processes input parameters.
Exception Management	When a method call in your application fails, what does your application do? How much do you reveal? Do you return friendly error information to end users? Do you pass valuable exception information back to the caller? Does your application fail gracefully?
Auditing and Logging	Who did what and when? Auditing and logging refer to how your application records security-related events.

Fingerprinting the Web Application Environment through Penetration Test

One of the first steps of the penetration test should be to identify the Web application environment, including the scripting language and Web server software in use, and the operating system of the target server. All of these crucial details are simple to obtain from a typical Web application server through the following steps:

Investigate the Output from HEAD and OPTIONS http requests

The header and any page returned from a HEAD or OPTIONS request will usually contain a SERVER: string or similar detailing the Web server software version and possibly the scripting environment or operating system in use.

Investigate the Format and Wording of 404/other Error Pages

Some application environments (such as ColdFusion) have customized and therefore easily recognizable error pages and will often give away the software versions of the scripting language in use. The tester should deliberately request invalid pages and utilize alternate request methods (POST/PUT/Other) in order to glean this information from the server.

Test for Recognized File Types/Extensions/Directories

Many Web services (such as Microsoft IIS) will react differently to a request for a known and supported file extension than an unknown extension. The tester should attempt to request common file extensions such as .ASP, .HTM, .PHP, .EXE and watch for any unusual output or error codes.

Examine Source of Available Pages

The source code from the immediately accessible pages of the application front-end may give clues as to the underlying application environment.

Manipulate Inputs in Order to Elicit a Scripting Error

TCP/ICMP and Service Fingerprinting

Using traditional fingerprinting tools such as Nmap and Queso, or the more recent application fingerprinting tools Amap and WebServerFP, the tester can gain a more accurate idea of the underlying operating systems and Web application environment than through many other methods. NMAP and Queso examine the nature of the host's TCP/IP implementation to determine the operating system and, in some cases, the kernel version and patch level. Application fingerprinting tools rely on data such as Server HTTP headers to identify the host's application software.

Hidden form Elements and Source Disclosure

In many cases developers require inputs from the client that should be protected from manipulation, such as a user-variable that is dynamically generated and served to the client, and required in subsequent requests. In order to prevent users from seeing and possibly manipulating these inputs, developers use form elements with a HIDDEN tag. Unfortunately, this data is in fact only hidden from view on the rendered version of the page - not within the source.

This practice is still common on many sites, though to a lesser degree. Typically only non-sensitive information is contained in HIDDEN fields, or the data in these fields is encrypted. Regardless of the sensitivity of these fields, they are still another input to be manipulated by the tester.

All source pages should be examined (where feasible) to determine if any sensitive or useful information has been inadvertently disclosed by the developer - this may take the form of active content source within HTML, pointers to included or linked scripts and content, or poor file/directory permissions on critical source files. Any referenced executables and scripts should be probed, and if accessible, examined.

Determining Authentication Mechanisms

One of the biggest shortcomings of the Web applications environment is its failure to provide a strong authentication mechanism. Of even more concern is the frequent failure of developers to apply what mechanisms are available effectively. It should be explained at this point that the term Web applications environment refers to the set of protocols, languages and formats - HTTP, HTTPS, HTML, CSS, JavaScript, etc. - that are used as a platform for the construction of Web applications.

HTTP provides two forms of authentication: **Basic and Digest**.

These are both implemented as a series of HTTP requests and responses, in which the client requests a resource, the server demands authentication and the client repeats the request with authentication credentials. The difference is that Basic authentication is clear text and Digest authentication encrypts the credentials using a nonce (time sensitive hash value) provided by the server as a cryptographic key.

Besides the obvious problem of clear text credentials when using Basic, there is nothing inherently wrong with HTTP authentication and this clear-text problem are mitigated by using HTTPS. The real problem is twofold. First, since this authentication is applied by the Web server, it is not easily within the control of the Web application without interfacing with the Web server's authentication database. Therefore custom authentication mechanisms are frequently used.

These open a veritable Pandora's Box of issues in their own right. Second, developers often fail to correctly assess every avenue for accessing a resource and then apply authentication mechanisms accordingly.

Given this, testers should attempt to ascertain both the authentication mechanism that is being used and how this mechanism is being applied to every resource within the Web application. Many Web programming environments offer session capabilities, whereby a user provides a cookie or a Session-ID HTTP header containing a psuedounique string identifying their authentication status. This can be vulnerable to attacks such as brute forcing, replay, or re-assembly if the string is simply a hash or concatenated string derived from known elements.

Every attempt should be made to access every resource via every entry point. This will expose problems where a root level resource such as a main menu or portal page requires authentication but the resources it in turn provides access to do not. An example of this is a Web application providing access to various documents as follows. The application requires authentication and then presents a menu of documents the user is authorized to access, each document presented as a link to a resource such as: <http://www.server.com/showdoc.asp?docid=10>

Although reaching the menu requires authentication, the showdoc.asp script requires no authentication itself and blindly provides the requested document, allowing an attacker to simply insert the docid GET variable of his desire and retrieve the document. As elementary as it sounds this is a common flaw in the wild.

ENDING NOTES

A Secure Web Application relies upon a secure network infrastructure. The network infrastructure consists of routers, firewalls, and switches. The role of the secure network is not only to protect itself from TCP/IP-based attacks, but also to implement countermeasures such as secure administrative interfaces and strong passwords. The secure network is also responsible for ensuring the integrity of the traffic that it is forwarding. If you know at the network layer about ports, protocols, or communication that may be harmful, counter those potential threats at that layer.

An ever-increasing number of attacks target your application. They pass straight through your environment's front door using HTTP. The conventional fortress model and the reliance on firewall and host defenses are not sufficient when used in isolation. Securing your application involves applying security at three layers: the network layer, host layer, and the application layer. A secure network and host platform infrastructure is a must. Additionally, your applications must be designed and built using secure design and development guidelines following timeworn security principles.

REFERENCES

1. Software Security Testing: Seeking security in an insecure world by Gary McGraw, Ph.D. CTO, Cigital
2. <http://www.net-security.org/secworld.php?id=9899>
3. <http://searchsoftwarequality.techtarget.com/resources/Software-Security-Test-Best-Practices>