



WEB APPLICATIONS SECURITY

Abstract

An increase in the usage of web applications is directly related to an increase in the number of security incidents for them. 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.

The Problem

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.

Understanding Web Application Security Testing

Organizations need to find methods for achieving their mission goals in using the Internet and at the same time keeping



their Internet sites secure from attack. 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?

The Root of the Issue: Input Validation

Input validation issues 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 "blackbox" testing must be applied.

What exactly is a Web application?

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.

How does it look from the users' perspective?

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 netcat. The process of exhaustive blackbox testing a Web application is one that involves exploring each data element, determining the expected input, manipulating or otherwise corrupting this input, and analysing the output of the application for any unexpected behaviour.



Fingerprinting the Web Application Environment

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 recognised 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 be 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 pseudo-unique 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 authorised 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.

Solution

Security testing aims to address inherent flaws and weaknesses in applications to ensure that the final product is robust, and can safeguard sensitive data without compromising data confidentiality and integrity. Depending on testing time frames, more specific, or thoroughly comprehensive testing can be performed to determine the robustness of an application in the environment within which it will eventually be deployed.

Conclusion

Security testing has gained unparalleled significance over the years, and will continue to rise up the popularity chart given the frequent advancements that we have come to notice and appreciate in



the world of technology. Moreover, the importance of data integrity and confidentiality have begun to play a significant role, as have the client demands increased in terms of having access to secure software which delivers without compromising sensitive information. Our aim is to perform thorough security testing to highlight shortcomings and weaknesses in applications to enable developers to build more reliable, and dependable applications in future.