OWASP
The Open Web Application Security Project
6th OWASP AppSec Conference - Italy 2007

The new OWASP Web Application Penetration Testing Guide

Matteo Meucci, Alberto Revelli
Owasp Italy
matteo.meucci@owasp.org, r00t@northernfortress.net

Abstract. Penetration testing has become a common technique used to test network security for many years. It is also commonly known as black box testing or ethical hacking. Penetration testing is essentially the "art" of testing a running application remotely, without knowing the inner workings of the application itself to find security vulnerabilities. The OWASP Testing Project has been in development for many years and has now reached version 2.0. With this release, we want to help people understand the what, why, when, where, and how of testing their web applications, and not just provide a simple checklist or prescription of issues that should be addressed.

Keywords: .OWASP, web security, ethical hacking, penetration testing

1 Introduction

A penetration test is a method of evaluating the security of a computer system or network by simulating an attack. A Web Application Penetration Test focuses only on evaluating the security of a web application. The process involves an active analysis of the application for any weaknesses, technical flaws or vulnerabilities. Any security issues that are found will be presented to the system owner together with an assessment of their impact and often with a proposal for mitigation or a technical solution. The OWASP testing guide aims to become a 'de facto' standard in describing how a penetration test should be performed.

2 The OWASP approach

The OWASP approach is Open and Collaborative:

- Open: every security expert can participate with his experience in the project. Everything is free.
- Collaborative: we usually perform brainstorming before the articles are written so we can share our ideas and develop a collective vision of the project. That means rough consensus, wider audience and participation.

This approach tends to create a defined Testing Methodology that will be:

- Consistent
- Reproducible
- Under quality control

The problems that we want to be addressed are:

- Document all
- Test all

We think it is important to use a method to test all known vulnerabilities and document all the the pen test activities.

3 The OWASP testing methodology

Penetration testing will never be an exact science where a complete list of all possible issues that should be tested can be defined. Indeed, penetration testing is only an appropriate technique for testing the security of web applications under certain circumstances. The goal is to collect all the possible testing techniques, explain them and keep the guide updated. The OWASP Web Application Penetration Testing method is based on the black box approach. The tester knows nothing or very little information about the application to be tested.

We have split the set of tests in 8 sub-categories:

- Information Gathering
- Business Logic Testing
- Authentication Testing
- Session Management Testing
- Data Validation Testing
- Denial of Service Testing
- Web Services Testing
- AJAX Testing

Each of these categories is further divided in a series of chapters, each of which describes a specific type of test, providing examples, screenshots, tools and links to additional information. In the following sections, a general overview of each category is provided.

3.1 Information gathering

The first phase in security assessment is focused on collecting as much information as possible about a target application. Information Gathering is a necessary step of a penetration test.

This task can be carried out in many different ways.

Using public tools (search engines), scanners, sending simple HTTP requests, or specially crafted requests, it is possible to force the application to leak information by sending back error messages or revealing the versions and technologies used by the application.

Often it is possible to gather information by receiving a response from the application that could reveal vulnerabilities in the bad configuration or bad server management.

The information gathering techniques are the following:

- Testing Web Application Fingerprint
- Application Discovery
- Spidering and Googling

- Analysis of Error Code
- Infrastructure Configuration Management Testing
- SSL/TLS Testing
- DB Listener Testing
- Application Configuration Management Testing
- Testing for File Extensions Handling
- Old, Backup and Unreferenced Files

3.2 Business logic

Business logic can have security flaws that allow a user to do something that isn't allowed by the business. For example, if there is a limit on reimbursement of $1000, could an attacker misuse the system to request more money than is allowed? Or perhaps you are supposed to do operations in a particular order, but an attacker could invoke them out of sequence. Or can a user make a purchase for a negative amount of money? Frequently these business logic security checks simply are not present in the application. Automated tools find it hard to understand context, hence it's up to a person to perform these kinds of tests.

3.3 Authentication

Authentication (Greek: αυθεντικός = real or genuine, from 'authentes' = author) is the act of establishing or confirming something (or someone) as authentic, that is, that claims made by or about the thing are true. Authenticating an object may mean confirming its provenance, whereas authenticating a person often consists of verifying her identity. Authentication depends upon one or more authentication factors. In computer security, authentication is the process of attempting to verify the digital identity of the sender of a communication. A common example of such a process is the logon process. Testing the authentication schema means understanding how the authentication process works and using that information to circumvent the authentication mechanism. The authentication system is tested with the following techniques:

- Default or guessable (dictionary) user account
- Brute Force
- Bypassing authentication schema
- Directory traversal/file include
- Vulnerable remember password and pwd reset
- Logout and Browser Cache Management Testing

3.4 Session management

At the core of any web-based application is the way in which it maintains state and thereby controls user-interaction with the site. Session Management broadly covers all controls on a user from authentication to leaving the application. HTTP

is a stateless protocol, meaning web servers respond to client requests without linking them to each other. Even simple application logic requires a user's multiple requests to be associated with each other across a "session". This necessitates third party solutions – through either Off-The-Shelf (OTS) middleware and web server solutions, or bespoke developer implementations. Most popular web application environments, such as ASP and PHP, provide developers with built in session handling routines. Some kind of identification token will typically be issued, which will be referred to as a "Session ID" or Cookie. There are a number of ways a web application may interact with a user. Each is dependent upon the nature of the site, the security and availability requirements of the application. Whilst there are accepted best practices for application development, such as those outlined in the OWASP Guide to Building Secure Web Applications, it is important that application security is considered within the context of the provider's requirements and expectations. In this chapter we describe the following items.

- Analysis of the Session Management Schema
- Cookie and Session Token Manipulation
- Exposed Session Variables
- Cross Site Request Forgery (CSRF)
- HTTP Exploiting

3.5 Input validation

The most common web application security weakness is the failure to properly validate input from the client or environment. This weakness leads to almost all of the major vulnerabilities in applications, such as interpreter injection, locale/Unicode attacks, file system attacks and buffer overflows. Data from any external entity/client should never be trusted for an external entity/client has every possibility to tamper with the data: "All Input is Evil" says Michael Howard in his famous book "Writing Secure Code". That's rule number one. The problem is that in a complex application the points of access for an attacker increase and it is easy that you forget to implement this rule. In this chapter we describe how to test all the possible forms of input validation to understand if the application is strong enough against any type of data input. We split Data Validation into these macro categories:

- Cross Site Scripting
- HTTP Methods and XST
- SQL Injection
- LDAP Injection
- ORM Injection
- XML Injection
- SSI Injection
- XPath Injection
- IMAP/SMTP Injection
- Code Injection
- OS Commanding

- Buffer overflow Testing

3.6 Denial of service

The most common type of denial of service (DoS) attack is the kind used on a network to make a server unreachable by other valid users. The fundamental concept of a network DoS attack is a malicious user flooding enough traffic to a target machine, that it renders the target incapable of keeping up with the volume of requests it is receiving. When the malicious user uses a large number of machines to flood traffic to a single target machine, this is generally known as a distributed denial of service (DDoS) attack. These types of attacks are generally beyond the scope of what an application developer can prevent within their own code. This type of “battle of the network pipes” is best mitigated via network architecture solutions.

There are, however, types of vulnerabilities within applications that can allow a malicious user to make certain functionality or sometimes the entire website unavailable. These problems are caused by bugs in the application, often resulting from malicious or unexpected user input. This section will focus on application layer attacks against availability that can be launched by just one malicious user on a single machine.

Here are the DoS testings we talk about:

- Locking Customer Accounts
- Buffer Overflows
- User Specified Object Allocation
- User Input as a Loop Counter
- Writing User Provided Data to Disk
- Failure to Release Resources
- Storing too Much Data in Session

3.7 Web services

SOA (Service Orientated Architecture)/Web services applications are up-and-coming systems which are enabling businesses to interoperate and are growing at an unprecedented rate. Webservice "clients" are generally not user web front-ends but other backend servers. Webservices are exposed to the net like any other service but can be used on HTTP, FTP, SMTP, MQ among other transport protocols.

The vulnerabilities in web services are similar to other vulnerabilities such as SQL injection, information disclosure ad leakage etc but web services also have unique XML/parser related vulnerabilities which are discussed here also.

The specific testing methods are the following:

- XML Structural Testing
- XML Content-level Testing
- HTTP GET parameters/REST Testing

- Naughty SOAP attachments
- Replay Testing

3.8 AJAX testing

AJAX, an acronym for Asynchronous JavaScript and XML, is a web development technique used to create more responsive web applications. It uses a combination of technologies in order to provide an experience that is more like using a desktop application. This is accomplished by using the XMLHttpRequest object and JavaScript to make asynchronous requests to the web server, parsing the responses and then updating the page DOM HTML and CSS.

Utilizing AJAX techniques can have tremendous usability benefits for web applications. From a security standpoint, however, AJAX applications have a greater attack surface than normal web applications, and they are often developed with a focus on what can be done rather than what should be done. Also, AJAX applications are more complicated because processing is done on both the client side and the server side. The use of frameworks to hide this complexity can help to reduce development headaches, but can also result in situations where developers do not fully understand where the code they are writing will execute. This can lead to situations where it is difficult to properly assess the risk associated with particular applications or features.

AJAX applications are vulnerable to the full range of traditional web application vulnerabilities. Insecure coding practices can lead to SQL injection vulnerabilities, misplaced trust in user-supplied input can lead to parameter tampering vulnerabilities, and a failure to require proper authentication and authorization can lead to problems with confidentiality and integrity. In addition, AJAX applications can be vulnerable to new classes of attack such as Cross Site Request Forgery (XSRF).

Testing AJAX applications can be challenging because developers are given a tremendous amount of freedom in how they communicate between the client and the server. In traditional web applications, standard HTML forms submitted via GET or POST requests have an easy-to-understand format, and it is therefore easy to modify or create new well-formed requests. AJAX applications often use different encoding or serialization schemes to submit POST data making it difficult for testing tools to reliably create automated test requests. The use of web proxy tools is extremely valuable for observing behind-the-scenes asynchronous traffic and for ultimately modifying this traffic to properly test the AJAX-enabled application.