]HackingTeam[	

HT Application Methodology

HT S.r.l.	http://www.hackingteam.it
Via della Moscova, 13 *20121 MILANO (MI) - Italy*	info@hackingteam.it
Tel. +39.02.29060603	*Fax +39.02.63118946*

INDEX

Executive Summary

This document describes a vulnerability assessment activity carried out by HT on the web application of XXXXX. The goal of such an activity has been to assess the actual security of the aforementioned application in front of external and internal attack attempts for the purpose of identifying possible vulnerabilities which could allow an attacker to perform a fraud or other criminal abuses.

1 Introduction

Security attacks against web applications providing various kinds of financial on line services represent nowadays a serious threat to the business of financial institutions and their clients. Hacking Team carried out a vulnerability assessment of the on line application of XXXX for the purpose of identifying possible vulnerabilities before crackers and other criminals find and exploit them to perform financial frauds. During this assessment we considered many realistic scenarios which envolve attacks varying from those originating from an external user with little information about the application to attacks originating from a legitimate user with considerable information about the application.

We acted in two operational modes, namely in a black box mode and in a white box mode. In black box mode we were granted no access to the application and were given no information about the application code, its software components and its administration. In white box mode we were granted two categories of an account for the same user which we used as an internal profile. We tried a rich set of attacks and evasion against the application, which demonstrated to be equipped with a good coding and administration defense. For most of the attacks we carried out a selective code auditing, i.e. analyzed only the code strictly related to the functionality we were testing for vulnerabilities.

This document provides a description of the main attacks tried against the application, the affects on the application deriving from the execution of each attack, a description of some weaknesses identified in the application, and the corresponding countermeasures.

2 Security Evaluation

2.1 Hacking Tools Employed

During the security assessment on the on line application of XXXXX in addition to a manual analysis we performed a quantitative automatic analysis through various hacking tools and hacking support tools. The main tools we utilized for the vulnerability assessment activity are the following:

- ***Application Vulnerability Scanners***: these tools automatically crawl every possible resource of the target application and try to identify possible vulnerabilities by using a data base of signatures of known vulnerabilities. Furthermore, some of these tools scan the web server hosting the target application as well.

- ***Web proxies***: these tools place themselves in the middle of a browser and a target web server and allow for interception and modification on the fly of all HTTP requests and responses.

- ***HTTP Editors***: these tools enable an attacker to manually build HTTP requests, send them to a target web server and view the possible corresponding response.

- ***Encoding/Decoding tools***: these tools are used for encoding some defined bytes from a plain form into URL representation, base64, overlong UTF-8, etc., and vice versa, i.e. decoding some defined bytes from a URL representation, base64, overlong UTF-8, etc., into a plain form.

- ***Cookie Analysis Tools***: these tools allow an attacker to gather all cookies used by a target application and perform several kinds of analysis on them for the purpose of evaluating the possibility of corrupting their value and damage the functionality of the target application.

- ***Site mapping tools***: these tools perform an automatic download of all possible files from a defined site. They are usually used for constructing the structure of a target application and gathering material to be used in offline analysis.

- ***Database Scanning tool***: these tools scan a target network looking for database servers reachable directly from the network. These tools also assist an attacker in attacking possible database servers found on a network.

- ***Injection Tools*** : these tools allow for automatically performing various kinds of injection on resources specified by an attacker.

2.2 Hacking Tests Performed

In this section we describe the attacks we implemented against the on line application of XXXXX. For each attack we specify whether it was performed in a black or white box mode, and whether it was successful or not.

2.2.1 Brute Force Attacks to the Authentication System

The on line application of XXXXX uses a form based authentication to authenticate users. The authentication system authenticates them through a combination of user name and password they should provide. Operating in a black box mode we tried a dictionary attack against the application. Assuming we do not know how a user name is structured we used common user names and a dictionary of italian words for such attack. This attack was unsuccessful. Passing into a white box mode we noticed that the user name is composed of the first four letters of the user's first name, the first four letters of the user's last name, and two digits like 18 or 21. We also tried to perform the attack in a pure brute force style, i.e. trying with randomly generated passwords. We replayed the attack in both forms, and it was again unsuccessful.

2.2.2 Other Attacks to the Authentication System

Operating in a black box mode we tried a SQL injection attack against the authentication form, assuming that the target application executes a query on a database to determine whether a user provided a valid combination of user name and password. We also considered the possibility that the application authenticates a user by connecting to an LDAP server and performing a query on it. Consequently we tried an LDAP injection attack against the authentication form. While trying to carry out these attacks it was clear that HTTP requests passed through an application firewall before reaching the web server hosting the target application.

As a matter of fact while feeding to the authentication form certain particular attack characters in various encoding we were redirected to the authentication page. These attack characters are fundamental for extending a possible SQL query or LDAP query for the purpose of forcing the condition to be true and return a positive result, and consequently grant access to the application. The application firewall was efficient in blocking the aforementioned attacks as it rejected the attack characters.

Operating in white box and auditing the code which performs the authentication of a user we found out that in reality the target application performs an LDAP query for the purpose of authenticating a user. We crafted the LDAP injection attack and retried it. Nevertheless, besides being blocked by the application firewall, the LDPA injection attack did not succeed against the authentication system because the later uses the user name and password for connecting to the LDAP server. This is a smart decision and eliminates any kind of injection attack since if the user name and password are not valid the authentication will fail. Under these circumstances the application will not be able to connect to the LDAP server.

A common security error consists in connecting to the LDAP server usually as an administrator and then performing queries which authenticate a user. In that case an LDAP injection could be feasible as the application already provides the connection to the LDAP server, and what is left to the attacker is to maliciously extend an LDAP query. Nevertheless, the application does not suffer from such a vulnerability.

2.2.3 Session/Credential Prediction

These attacks were performed in white box mode. We gathered several cookie values and analyzed them. We focused on critical elements such as the length of the cookie values, the set of characters used to generate cookie values, the presence of sensitive information in cookie values, the entropy of cookie values, i.e. the number of characters which change when the application generates a new cookie value for a given cookie, and ended up with no vulnerabilities. The application generates robust cookie values.

With regard to the prediction of credentials we noticed that the user name is partially predictable since the attacker should only guess the last two digits in a user name. Furthermore, the initial password assigned to a user is formed of 8 digits. The search space in this case is suitable for a brute force attack and is applicable from the moment the account of a new user is registered till the legitimate user accesses the application for the first time and changes the initial password.

2.2.4 Insufficient Authorization

Operating in a black box mode we stressed the application to see if it is possible to access certain resources which should be accessed only after providing a valid credential. The application does not allow access to protected resources if the request does not include valid cookies and valid cookie values, which in turn are provided only if the user authenticates with success. The application handles correctly the cookies, therefore it has not been possible to bypass the authorization scheme. From a white box mode we tried to access certain application resources as another user, and these attempts were not successful as the application makes a proper use of sessions. Nonetheless, we found out that the application allows simultaneous access from different terminals. Generally it is a common security practice not to allow such a simultaneous access to an application to be protected.

2.2.5 Weak Password Recovery Validation

In a black box mode we tried several SQL and LDAP injection attacks against the password recovery procedure. These attacks were blocked from their very first step by the application firewall. We tried to identify functional weaknesses in the aforementioned procedure. We deem that an attacker who personally knows a legitimate user could bypass the first phase of the password recovery procedure since he could easily construct the first 8 characters of the user name and brute force the last two digits. Furthermore, an attacker defined as above could obtain personal data of such a legitimate user, for example the complete date of birth. Nevertheless, we believe the attacker cannot reach the combination of a secret question and the related response, thus an attacker would be blocked at the second phase of the password recovery procedure.

2.2.6 Insufficient Session Expiration

After authenticating to the application we logged out and verified that our previous session was invalidated by the application. In this sense the application behaves correctly. We did another test on the session expiration issue. We authenticated to the application and did nothing on it for around 30 minutes. Although there was a relatively long period of inactivity, the application did not invalidate the current session like it should

2.2.7 Content Spoofing / Cross-Site Scripting

Operating in a white box mode we searched for a possibility to insert malicious data and latter have other users view it. This is a common scenario for a cross site scripting attack, and the malicious data in this case would be code written in a client side language such as JavaScript. The application is protected from these attacks. There are some points in which an attacker could try to insert malicious data, but the application does not allow viewing it at all. Furthermore, it has not been possible to even insert malicious data as they are rejected by the application firewall.

2.2.8 Command Execution & Low-level Coding Vulnerabilities Exploitation

Server side code in various applications happens to call operating system commands through functions such as *system()*. By injecting malicious characters it is possible to extend the string which represents the command to be executed and make it contain the original command and other attacker defined commands. We tried such an attack both in black box and in white box and observed that the application firewall rejects the characters needed for carrying out a command injection attack. From the selective code auditing did not result the existence of a function which could be abused with for the purpose of implementing a command injection attack.
With regard to a possible exploitation of low-level coding vulnerabilities such as buffer overflows, well, this attack is not applicable in Java code. In fact Java does not suffer from buffer overflows. A potential target for this kind of exploitation could be the web server hosting the target application, but it has not been possible to identify a way to carry out this kind of exploit with the Teros application firewall in the middle.

2.2.9 SSI Injection

Server side include (SSI) attacks exploit the fact that a web server may execute several commands by following preliminarily defined directives to generate dynamic content. The commands that may be executed through these directives vary from including a certain file to a direct execution of an operating system command. We injected server side include directives to the application but the application firewall rejects them. The server side include attack attempt has been performed both in black box mode and in white box mode.

2.2.10 Directory Listing

The web server hosting the on line application of XXXXX does not allow a listing of any directory within its root directory.

2.2.11 Information Leakage

There have not been found in the on line application of XXXXX any ways of leaking information other than the information which may be easily deduced by observing the general client side architecture of such an application.

2.2.12 Path Traversal

These attacks consist in sending HTTP requests for resources which are not part of the web server's root directory. It has not been possible to access resources other than those explicitly allowed by the application.

2.2.13 Abuse of Functionality

In a white box mode we tried various attempts to abuse with the application, especially for assessing the possibility of a financial fraud. The application is well programmed, therefore there have not been identified any functional weaknesses which could enable an attacker to carry out a financial fraud.

2.2.14 Denial of Service

In black box mode we tried several script injection attacks against the application. These attacks consist in injecting code written in the language of the server side code and if successful could lead to the processing of the injected script code. We operated on the fields of the authentication form and on several cookies. In front of code injection attacks the web server hosting the application fell into internal error.

2.2.15 Forcefull Browsing

This attack consists in a direct request for a resource without following the execution path defined by the application programmer. In white box mode it has been possible to access the source code of certain scripts. According to the application logic these files should never be requested directly by a user. In fact they are to be included in other script files and execute as an integral part of those other script files. By requesting the aforementioned script files directly the web server returned their source code as they had an extension which does not recognize them as server side scripts, therefore the web server considers them as simple text files and returns their content.

2.2.16 HTTP Request Smuggling

The HTTP request smuggling attack takes advantage of the lack of standardization in parsing HTTP traffic when one or more HTTP entities like a cache server, proxy server, or application firewall are in the data flow between a client and the target web server. An HTTP request smuggling attack consists in sending multiple HTTP requests that cause the two attacked entities to see different sets of requests enabling an attacker to smuggle a request to one device without the other device noticing it. We analyzed the way the Teros application firewall and apache web server parse HTTP traffic and ended up with the result that there are no parsing discrepancies in these two entities.