



External Network & Web Application Assessment

For

The XXX Group LLC

October 2012

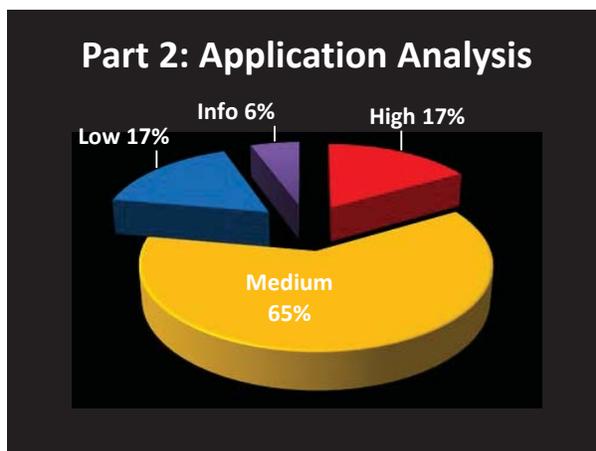
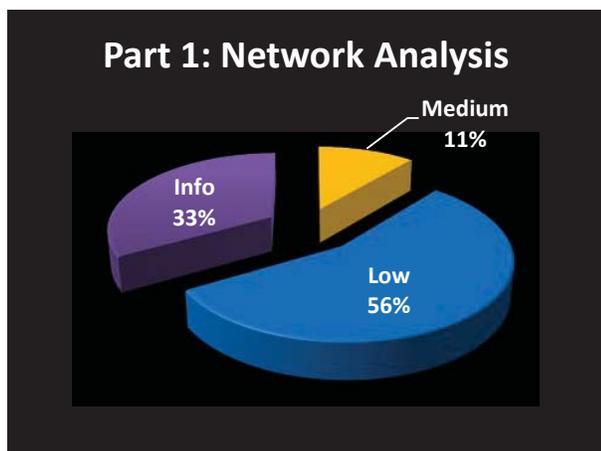
This report is solely for the use of client personal. No part of it may be circulated, quoted, or reproduced for distribution outside the client organization without prior written approval from 2Secure Corp.



Executive Summary Report

The web application will be the main target for any attacker; it's essential to have numerous security measures (protection layers), protecting the application.

We see an improvement from our last audit from June 2010; some of the layers implemented are old and new measures should be implemented such as Web Application Firewall (WAF) that examines the content of the traffic or the use of Database Firewall that examines requests before they arrive to the database.



The total estimated of time needed to repair all findings is 62 hours, see below Findings Summary.

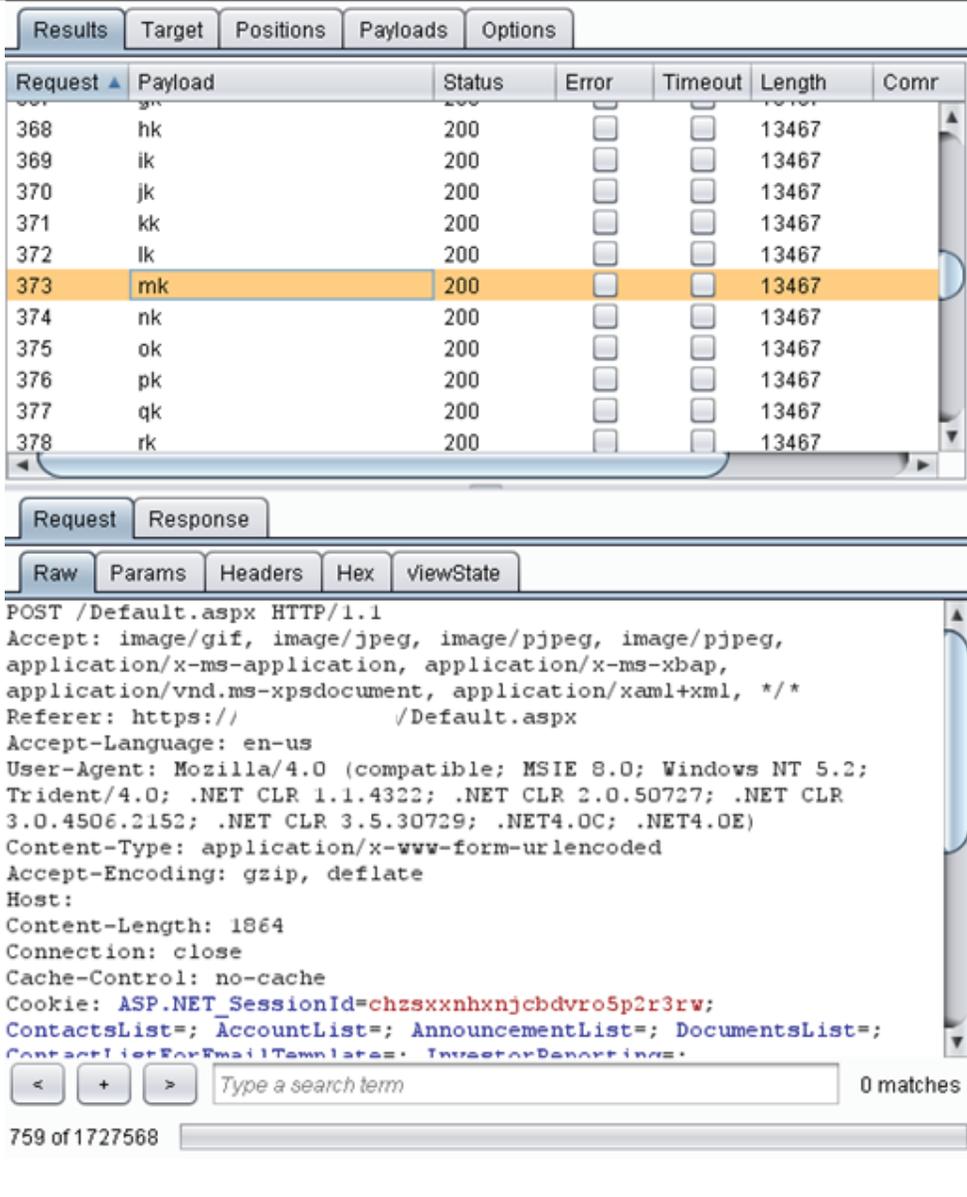


DEFAULT OR GUESSABLE (DICTIONARY file) USER ACCOUNT	
Summary	In the test we will use the forgotten password function to facilitate a dictionary attack; using a large file known user accounts we can enumerate a valid user by monitoring application responses.
Description	There are some ways that can be employed to find user accounts. The application provide us the option to reset account password, based on application response we can deduce that.
What was tested	<p>In this test we are looking to capture a different message from “Please contact Investor Services for further assistance” that may identify a valid account.</p> <p>This test exposed one account “kayne”</p>
Risk	Medium
Recommendation(s)	<ol style="list-style-type: none"> 1. It is recommended that if the account does not exists; the application will present erroneous message telling the user to check his email, this way an attacker won't know if accounts are valid or not, since all of them are valid, otherwise continue with the reset process. 2. Implement the use of CAPTCHA mechanism to block this type of attack. 3. In addition, the management application should verify that none of



	<p>below items exist in dictionary files :</p> <ul style="list-style-type: none">a. Creation of new accountb. Current accounts <p>These checks will reduce the probability to find valid accounts using dictionary files</p>
--	---



TESTING FOR BRUTE FORCE	
Summary	We tried to brute force password user accounts on forgot password page
Description	This is common attack where large lists of user accounts are being used to harvest many account as possible.
What was tested	 <p>The screenshot shows a table of test results with columns: Request, Payload, Status, Error, Timeout, Length, and Comr. The row for Request 373 with Payload 'mk' is highlighted in orange, showing a Status of 200 and Length of 13467. Below the table, the 'Request' tab is selected, showing the raw HTTP request details for a POST to /Default.aspx.</p> <pre> POST /Default.aspx HTTP/1.1 Accept: image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xaml+xml, */* Referer: https:// /Default.aspx Accept-Language: en-us User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.2; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; .NET4.0C; .NET4.0E) Content-Type: application/x-www-form-urlencoded Accept-Encoding: gzip, deflate Host: Content-Length: 1864 Connection: close Cache-Control: no-cache Cookie: ASP.NET_SessionId=chzsxxnhxnjcbdvro5p2r3rw; ContactsList=; AccountList=; AnnouncementList=; DocumentsList=; ContactListForEmailTemplates= InvestorReporting= </pre> <p>Results: test was successful; we have retrieved one account "amk"</p>
Risk	High



Recommendation(s)	User name is too short; enforce users to have longer user names to be at least 8 characters long in addition to above recommendations.
--------------------------	--

TESTING FOR VULNERABLE REMEMBER PASSWORD, PASSWORD RESET & AUTO COMPLETE	
Summary	Password Reset
Description	In this test we try to how this mechanism works and how it can be manipulated.
What was tested	Password Reset The password reset in this application requires entering login ID. To test this we tried to use "amk" This initiated a process of asking the user secret questions.
Risk	None
Recommendation(s)	None

TESTING FOR VULNERABLE REMEMBER PASSWORD, PASSWORD RESET & AUTO COMPLETE	
Summary	Remember Me
Description	In this test we try to how this mechanism works and how it can be manipulated.
What was tested	Remember Me This parameter will disable from the browser to save credentials in the browser cache, if a hacker is able to access this cache, he could read the password in clear text.
Risk	None
Recommendation(s)	None



SQL INJECTION	
Summary	A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application.
Description	A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file existing on the DBMS file system and, in some cases, issue commands to the operating system.
What was tested	<p>Our target is the "forgotten your password" function; entering our test user "ybehar"</p> <p>.....</p> <p>Retrieve Password</p> <p>To receive an email with a temporary password, Please enter your Login ID and click Go to complete your password reset request.</p> <p>User Name: <input type="text" value="ybehar"/> <input type="button" value="Go"/></p> <p>.....</p> <p>Next on the security question we entered "(select%201)" this caused the application to bring us to the next security question.</p> <p>.....</p> <p>Login</p> <p>Question: What was the last name of your third grade teacher? Answer: <input type="text" value="(select%201)"/> <input type="button" value="Go"/></p> <p>.....</p> <p>To make sure we did not have a false positive was to restart this process and enter wrong answer for the first security question instead of the injection as described above.</p> <p>The application brings the next security question:</p>



	<p>Login</p> <p>Question: In what city or town was your first job? Answer: (select%201) <input type="button" value="Go"/></p> <p>.....</p> <p>And the application stopped here we need to change the injection to (select%202) to pass this...</p> <p>.....</p> <p>Please contact Investor Services for further assistance.</p> <p>Back to Login page</p> <p>.....</p> <p>Result: there is a possibility for SQL injection therefore mitigation is needed, see below recommendations.</p>
<p>Risk</p>	<p>High</p>
<p>Recommendation(s)</p>	<p>The most effective way to prevent SQL injection attacks is to use parameterized queries (also known as prepared statements) for all database access. This method uses two steps to incorporate potentially tainted data into SQL queries: first, the application specifies the structure of the query, leaving placeholders for each item of user input; second, the application specifies the contents of each placeholder. Because the structure of the query has already defined in the first step, it is not possible for malformed data in the second step to interfere with the query structure. You should review the documentation for your database and application platform to determine the appropriate APIs which you can use to perform parameterized queries. It is strongly recommended that you parameterized every variable data item that is incorporated into database queries, even if it is not obviously tainted, to prevent oversights occurring and avoid vulnerabilities being introduced by changes elsewhere within the code base of the application. You should be aware that some commonly employed and recommended mitigations for SQL injection vulnerabilities are not always effective:</p> <ul style="list-style-type: none"> ➤ One common defense is to double up any single quotation marks



appearing within user input before incorporating that input into a SQL query. This defense is designed to prevent malformed data from terminating the string in which it is inserted. However, if the data is being incorporated into queries is numeric, then the defense may fail, because numeric data may not be encapsulated within quotes, in which case only a space is required to break out of the data context and interfere with the query. Further, in second-order SQL injection attacks, data that has been safely escaped when initially inserted into the database is subsequently read from the database and then passed back to it again. Quotation marks that have been doubled up initially will return to their original form when the data is reused, allowing the defense to be bypassed.

- Another often cited defense is to use stored procedures for database access. While stored procedures can provide security benefits, they are not guaranteed to prevent SQL injection attacks. The same kinds of vulnerabilities that arise within standard dynamic SQL queries can arise if any SQL is dynamically constructed within stored procedures. Further, even if the procedure is sound, SQL injection can arise if the procedure is invoked in an unsafe manner using user-controllable data.



XSS - Injections	
Summary	Cross-Site Scripting (XSS) attacks are a type of injection problem, in which malicious scripts are injected into the otherwise benign and trusted web sites.
Description	<p>XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user in the output it generates without validating or encoding it.</p> <p>An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site. These scripts can even rewrite the content of the HTML page.</p>
What was tested	<p>Injection code of <code><ScRiPt>alert(testme)</sCriPt></code> to user name field & password: foo</p> <p>Apparently the application has detected a general error which may indicate that the .NET triggered that error.</p>  <p>url encoding: <code>%3c%53%63%52%69%50%54%3e%61%6c%65%72%74%28%74%65%73%74%6d%65%29%3c%2f%73%43%72%69%50%74%3e</code></p> <p>Result: We got an error page, see above picture.</p>



	<p>Base-64 encoding: PFNDUKIQVD5hbGVydChYU1MgYXR0YWNrKTwwc2NyaXB0Pg==</p> <p>Result: We got the login screen again.</p> <p>Html encoding: &#x3c; &#x53; &#x63; &#x52; &#x69; &#x50; &#x54; &#x3e; &#x61; &#x6c; &#x65; &#x72; &#x74; &#x28; &#x74; &#x65; &#x73; &#x74; &#x6d; &#x65; &#x29; &#x3c; &#x2f; &#x73; &#x43; &#x72; &#x69; &#x50; &#x74; &#x3e;</p>
Risk	High
Recommendation(s)	Ensure that proper sensitization steps are taken on the server, as well as on the client.

Sincerely,

Yigal Behar
Principle IT Security Consultant
2Secure Corp

