activity

In partnership with CST

Web Application Security Assessment Report

Acme Inc

V1.0

27 November 2012

2012-999

Copyright

Activity Information Management Limited
25 Hercules Way, Aeropark, Farnborough, Hampshire, GU14 6UU
Tel: 01252 377321 Fax: 01252 377670

Document Authorisation

Title	Version	Reference	Author	Reviewer	Date
Web Application Security Assessment Report	1.0	2012-999 RELEASE	A N Other	D. Boss	1st Sep 2012
Web Application Security Assessment Report	0.b	2012-999 DRAFT	A N Other	D. Boss	1st Sep 2012
Web Application Security Assessment Report	0.a	2012-999 DRAFT	A N Other	D. Boss	1st Sep 2012

Executive Summary

Overview

Acme Inc engaged Activity to conduct a Web Application Security Assessment of its Internet facing MyApp. The purpose of the engagement was to utilise active exploitation techniques in order to evaluate the security of the application against best practice criteria and to validate its security mechanisms and identify application level vulnerabilities.

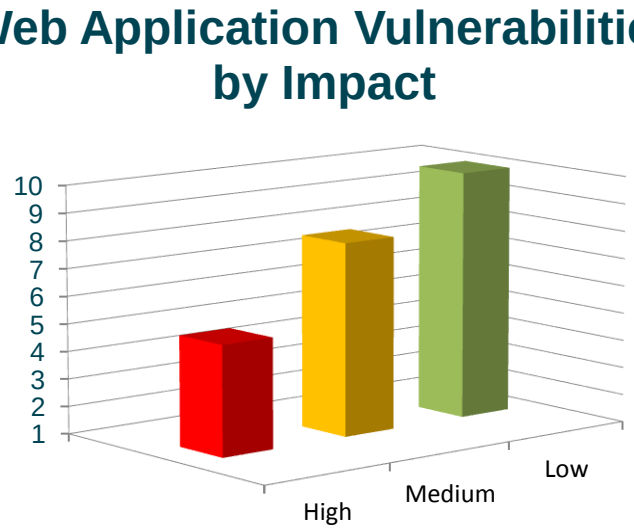
A Web Application Security Assessment provides Acme Inc with insight into the resilience of an application to withstand attack from unauthorised users and the potential for valid users to abuse their privileges and access. The assessment evaluates the security of the application against best practice criteria to validate security mechanisms and identify application level vulnerabilities.

This report details the scope of testing conducted, all significant findings along with detailed remedial advice. The summary below provides a non technical audience with a summary of the key findings and relates these back to business impacts. Section two of this report relates the key findings. Section three of this report highlights potential control areas where Acme Inc may want to invest further resources in order to improve the overall security posture of their systems. Section four of this report provides detailed narration and individual vulnerability findings that are aimed at a technical audience.

This document summarises the findings, analysis and recommendations from the assessment, which was conducted across the Internet from Activity offices in Farnborough, Hampshire.

Summary of Findings

The graph below shows a summary of the number of vulnerabilities found for each impact level for the Web Application Security Assessment. A significant number of high impact vulnerabilities were found that should be addressed as a priority.



The application has been deployed in a manner that is not in line with best practice guidelines for application and web servers facing the Internet. The application was found to be vulnerable to a number of attacks related to the authentication mechanisms and implemented authorisation controls that would result in unauthorised access to the application and compromise of the application and users' data.

Specifically, the application is vulnerable to a number of exploitable issues that are a direct consequence of either inadequate or non-existent input validation routines. The majority of these issues are a result of Cross Site Scripting vulnerabilities. The potential impacts of successful Cross Site Scripting attacks can be disclosure of user credentials and use of the site to fool users into accessing other compromised or malicious sites, which could damage brand and reputation which would ultimately have a financial cost.

In a multi user shared working environment such as the Acme e-portal, it is feasible that attackers would use Cross Site Scripting attacks to steal other users' credentials and sessions, in order to masquerade as those users or elevate their privileges to perform actions that they may otherwise be unable to perform.

Cross site scripting attacks would require knowledgeable users to perform them. Acme should review what the impact of a successful XSS attack on their users would be and its likelihood against the cost of remediation which would consist of sanitising all user supplied data on its receipt and on its use.The lack of input validation within the application also resulted in the discovery and exploitation of a number of SQL injection vulnerabilities. It is possible for an attacker to access the database with administrative privileges and manipulate the data store in order to access, modify or delete its data. It is also possible to use the same attack to execute operating system commands on the database server providing access to the underlying server resources with administrative privileges allowing an attacker the ability to attack internal systems that are not directly exposed to the Internet. Many tools are available to automate the exploitation of the application in this manner and the Internet has

many step-by-step guides to enable even the lowest skilled attacker to successfully execute an attack.

SQL injection targets the application's database and the infrastructure supporting that application and database. Acme's corporate network could be at risk depending how segregated the application is from the corporate environment. Given the ease of the attack, and what is at stake, Acme should urgently address this vulnerability by sanitising user supplied data. Acme should note that sanitising user supplied data addresses both the XSS and SQL injection issues at the same time if performed correctly.

The in house developed application should be re-engineered by Acme in order to resolve all of the identified issues. It is evident to Activity that a secure and consistent application development framework or standard has not been adopted or followed by all of the developers responsible for the application. Given this inconsistent approach to application development, Activity cannot be certain that every vulnerability within this application has been identified. To remediate this risk, Activity recommends that a full application source code review be conducted of the application's source code. Additionally, a review of HTML and active content source code could uncover a variety of vulnerabilities that are not likely to be found during a blind assessment, yet would be exploitable via insider knowledge or in the event that application source code is exposed

Contents

1 Introduction **8**
1.1 Overview 8
1.2 Scope 8
1.3 Out of Scope 8
1.4 Project Team 8

2 Key Findings **9**
2.1 Introduction 9
2.2 Key Issues 9

3 Control Areas **10**

4 Detailed Findings **12**
4.1 Web Application Security Assessment 12

Appendices **27**
Appendix A. Impact and Exploitation Definitions *28*
Appendix B. Vulnerability Details *29*
Appendix C. Exploit Code *30*
Appendix D. Port Scan Details *31*
Appendix E. Tools List *32*

1 Introduction

1.1 Overview

This report documents the findings for the Web Application Security Assessment of the Acme Inc Internet facing MyApp application.

The purpose of the engagement was to utilise exploitation techniques in order to identify and validate potential vulnerabilities across all systems within scope.

1.2 Scope

Activity performed a Web Application Security Assessment of the Acme Inc MyApp application.

Acme Inc defined the following application URL and web server host as in scope:

- http://www.vulnerable-application.com/insecure
- 10.0.0.1

1.3 Out of Scope

Acme Inc defined the following URL as out of scope:

- http://www.vulnerable-application.com/live

1.4 Project Team

The engagement began on 1st September 2012 and involved contributions from the following team members:

Role	Name
Customer Project Manager	Joe Bloggs (Project Manager)
Onsite Technical Contact	John Doe (Lead Application Developer)
Activity Project Manager	Name (Title)
Activity Team Members	Name (Title)

2 Key Findings

2.1 Introduction

This section outlines a summary of the key issues identified during the course of the assessment, which Activity recommend are prioritised by Acme Inc for review and, where appropriate, initial remediation. A qualitative impact factor (High, Medium, or Low) has been assigned to each vulnerability identified. However, all of the detailed findings in section four of this report should be reviewed and the recommended corrective action implemented where appropriate.

2.2 Key Issues

Activity identified a number of vulnerabilities that have the potential to yield command or administrative level access to the underlying operating system of the web server and its supporting database server, allowing an attacker to gain unauthorised access to the application and its resources, whilst permitting authenticated and authorised users of the application to elevate their privileges and attack other users of the system.

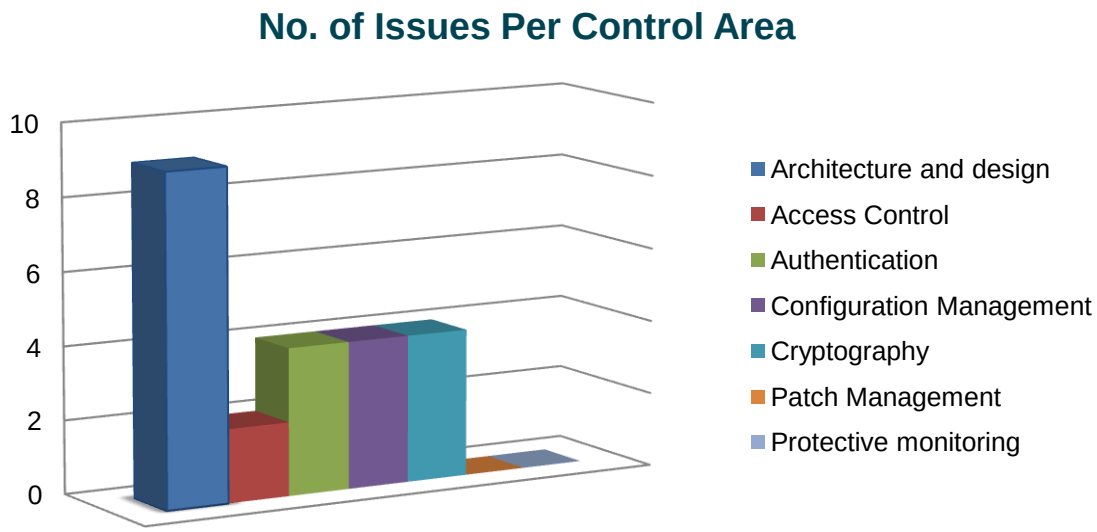
The following highlights the control areas that are the most trivial to exploit or expose the environment to the highest level of risk:

Key Issue	Control Area	Impact
A significant number of the application functions assessed were found to be vulnerable to SQL injection. An attacker can remotely execute operating system commands on the RDBMS and retrieve data from the database as well as performing modifications of the data.	**Architecture and Design**	**High**
A significant number of the application functions assessed were found to be vulnerable to Cross Site Scripting (XSS). XSS vulnerabilities are used to attack users of systems, rather than directly attacking the systems or infrastructure components themselves. The exploitable vector can be leveraged to hijack other users' sessions and access the application as the targeted user..	**Architecture and Design**	**High**
Execution of the 'Add User' functionality does not verify that the user requesting the action is an administrative user. A low-privileged application user can add new users to the system and assign administrative privileges.	**Access Control**	**High**

3 Control Areas

The findings from the test have been categorized according to the areas of control which should help prevent similar issue reoccurring. Multiple issues grouped into a single control area may indicate a root cause for those issues. Acme Inc can use this information to target effort, resource or investment to areas that will mitigate most risks.

The following diagram depicts the vulnerabilities identified as totals by control area.



The largest distribution of vulnerabilities is in Architecture & Design, with 18 issues identified.

A significant number of issues in this area may indicate systematic control failures. In addition to remediating the specific vulnerabilities highlighted, Activity recommends a thematic review of these areas is performed to improve processes and reduce the number of findings discovered in future tests.

Each of the control objectives in this area are described in the table overleaf.

	Control Objectives	Associated Issue's
Architecture and Design *The design and implementation of network and application security controls that reduce the effective attack surface of the environment against threats from without and within the enterprise.*	• Secure Network Topology • System Documentation • Securely designed software • Secure protocol selection • Correctly configured time sources	App-2, App-3, App-5, App-10, App-13, App-14, App-15, App-19, App-21
Access Control *Systems which store sensitive data or have administrative functions should be classified and access granted on a least privilege principle. These privileges should be reviewed on a regular basis and access removed where no longer required.*	• Appropriately Defined ACL's • Role Based Access Controls • Authorisation and Approval processes • Fail secure mechanisms • Periodic reviews performed	App-4, App-6
Authentication *Users of the system are positively identified using appropriate password management policies while detecting and preventing unauthorised access.*	• Presence of default credentials • User enumeration • Legal warning messages • Secure account recovery • Risk based password policy	App-7, App-8, App-12, App-20
Configuration Management *Relevant parameters are set to recommended values which ensure optimum performance while maintaining security and are protected from tampering or misuse.*	• Correctly Configured Services • System defaults removed • System hardening performed • Security features enabled • Secure build processes	App-11, App-16, App-17, App-18
Cryptography *All sensitive data transmissions should be conducted over a cryptographically secure channel using approved algorithms with a suitable key length. Key material should be securely generated, transmitted and protected.*	• Suitable encryption algorithms • Recognised certification authorities • Certificate revocation and expiration • Secure data transmission • Key Material Protection	App-1, App-9, App-22, App-23
Patch Management *Patches for both operating systems and applications are evaluated and applied in a timely manner, ensuring minimal exposure to high risk vulnerabilities. Decisions are made using a risk based approach in line with a vulnerability management policy.*	• Unsupported software • Patch availability monitoring • Risk based prioritisation • Deployment mechanisms • Missing patch detection	N/A
Protective Monitoring *System audit logs are centralised for analysis and reporting in order to detect unauthorised access and misuse of computing resources. In addition an audit trail is maintained to facilitate forensic reviews in the event of a security incident.*	• Adequate logging detail • Centralised audit logs • Log review and correlation • Network intrusion detection • System log integrity	N/A

4 Detailed Findings

This section records the main findings from the assessment along with any recommendations on issues identified. For each of the areas assessed and analysed the vulnerabilities discovered are categorised using the definitions explained in Appendix A.

4.1 Web Application Security Assessment

The assessment provides a point-in-time security analysis and resultant recommendations for improving the security of the application and its environment and consisted of the following activities:

- **Information Gathering** techniques were used by consultants in conjunction with a review of application and support system documentation in order to gain a deep and thorough understanding of how the application works, what its purpose is and how it has been implemented.
- **Reconnaissance** involved performing active assessment techniques in order to fingerprint the technologies and versions of software in use as well as mapping the available functionality of the application.
- **Communications Security and Cryptography** implementations were analysed in order to ensure that cryptography is appropriately used to protect the confidentiality and integrity of sensitive user data. Cryptographic algorithms, ciphers, key lengths and storage strategies were assessed to ascertain their effectiveness to withstand cryptanalysis attack.
- **Authentication Mechanisms** were examined to determine the effectiveness and resilience to subversion techniques.
- **Session Management** implementations were assessed and attempts were made to violate session state to become another valid user or to escalate privileges.
- **Authorisation Access Controls** that enforce authorisation levels for the application were analysed in detail to assess the user segregation methods employed and to validate their effectiveness.
- **Data Validation** routines were subjected to tests that consist of supplying unexpected data of various types and lengths, in order to ascertain the potential for exploitation of several classes of vulnerabilities such as; Operating System (Shell

and System), Database (SQL), Data (XML) and Scripting (XSS) Injections, Buffer Overflows, Format Strings etc.

4.1.1 Technical Summary

The application was found to be vulnerable to a number of attacks related to the authentication and authorisation controls. The application does not implement an account lockout threshold or any password complexity rules. Error messages presented upon valid and invalid logon attempts differ, allowing an attacker to determine valid and invalid usernames. These issues in combination allow for the crafting of a very effective brute force attack; that would ultimately result in unauthorised access to the application and compromise of the application and users' data.

The application's session handling mechanism implementation is vulnerable to several well known classes of vulnerabilities; its identifiers are predictable, can be fixed, captured and trivially reversed due to the use of a well known encoding implementation. These issues in combination allow for an attacker to hijack users' sessions and gain unauthorised and unauthenticated access to the application and users' data.

The application is vulnerable to a number of exploitable issues that are a direct result of inadequate and non-existent input validation routines. The application heavily relies on client side validation. Client side script can be bypassed using nothing more than changing web browser security settings. There are also many tools available to automate the exploitation of the application in this manner and the Internet has many step-by-step guides to enable even the lowest skilled attacker to successfully execute an attack.

Many instances of XSS (Cross-site scripting) were identified. An effective XSS attack could be used to steal user credentials. An attacker could feasibly use the vector to embed malicious code that would be run in the security context of the user's browser. In a multi user shared working environment application such as the application assessed, it is feasible that attackers would attempt to steal other users' credentials and sessions, in order to masquerade as those users or elevate their privileges to perform actions that they may otherwise be unable to perform. Such a successful attack would result in negative publicity, and has the potential to significantly damage the Acme Inc brand

The lack of input validation within the application also resulted in the discovery and exploitation of a number of SQL injection vulnerabilities. The web application fails to properly sanitize parameters that are passed to dynamically created SQL. It is possible for an attacker to alter the construction of back end SQL statements in order to manipulate the data store and execute operating system commands on the database server.

The application has been deployed in a manner that is not in line with best practice guidelines for application and web servers deployed in a hostile environment such as the Internet. A significant amount of test and default content was found to be present on the web server's file system; the file system permissions allowed for the viewing of sensitive

information and the application has not been deployed over a cryptographically secured channel, making traffic susceptible to network sniffing or man-in-the-middle manipulation.

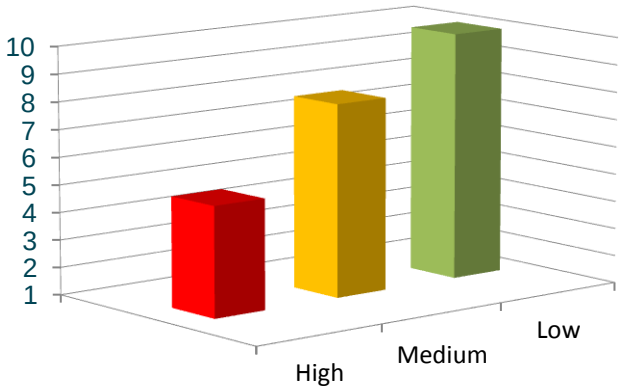
Activity recommends that a full application source code review be conducted of the application's source code. Given the limitations imposed by a blind application assessment, it is likely that certain unknown or undiscovered functionality could not be exercised. Additionally, a review of HTML and active content source code can uncover a variety of vulnerabilities that are not likely to be found during a blind assessment, yet would be exploitable via insider knowledge or in the event that application source code gets exposed.

The classes of vulnerabilities identified by Activity suggest that a consistent secure code development framework and effective quality assurance techniques have not been applied during the development life cycle. Many of the vulnerabilities discovered will require significant re-development of the application in order to remediate the exposed risk or to eradicate the vulnerability. Activity advises that as part of this process Acme Inc investigates the feasibility of producing a secure development strategy that incorporates a coding framework that is built from a secure coding standard is subjected to a quality assurance process. A good quality assurance programme can be effective in identifying and eliminating vulnerabilities during the development stage and would include activities such as penetration testing, fuzz testing, and source code audits. This would provide Acme Inc with a consistent approach to the security of developed applications and help reduce the number of vulnerabilities present in the MyApp application and other applications developed in the future by Acme Inc. There are many Commercial-Off-the-shelf (COTS) software packages that are available to support software security assurance activities. However, before they are used, these tools must be carefully evaluated and their effectiveness must be assured.

4.1.2 Vulnerability Table

The graph below shows a summary of the number of vulnerabilities found by impact level for the Web Application Security Assessment. A significant number of high impact vulnerabilities were found that should be addressed as a priority.

Web Application Vulnerabilities by Impact



The table below lists each of these vulnerabilities in detail, with recommendations on remedial action to remove the vulnerability along with an indication of the ease with which these vulnerabilities could be exploited.

ID	Vulnerability/Impact	Ease of Exploit	Recommendation
App-1.	**High** *Predictable Session Identifiers* The application utilises a custom 'SecurityToken' within a cookie to identify a users session. The token is generated using a predictable algorithm. The generation mechanism increments the value numerically. The 'SessionToken' cookie is Base64 encoded.	**Trivial** Base64 can be trivially decoded. There are many tools and coding examples available online. The session identifier is generated and set when the logon page is loaded. An attacker who can request the page can decode, observe and reverse engineer the algorithm easily. Several application security tools have specific functionality designed to discover and exploit this vulnerability.	The simplest method of implementing a session management mechanism for a given application is to utilise an existing session management library. These libraries have undergone a degree of public scrutiny and conformance testing. Most development frameworks and languages offer such mechanisms. ASP.NET provides a number of libraries to maintain user state, the most powerful of which is session state. This feature provides a convenient programmatic interface for associating arbitrary application state with a user session, and securely handles back-end state storage and client session management for the application. If the use of the ASP.NET session state features is not possible; re-engineer the application to use session identifiers that are opaque, unpredictable, and are

ID	Vulnerability/Impact	Ease of Exploit	Recommendation
			unique references to server-side state in order to help prevent session hijacking.
App-2.	**High** *Over Reliance on Client Side Validation* The application, in many places does, not enforce server side validation of client side data. Disabled fields can be enabled and edited. Application defined and populated variables can also be manipulated and corrupted. Throughout the application the reliance on JavaScript client side validation routines allows a user to corrupt data stored in the RDBMS, the bypassing of business and application logic rules, as well as providing exploitable vectors of injection vulnerabilities.	**Trivial** It is a trivial task to send a crafted request that bypasses client side script validation. An attacker simply needs to change their browser security settings to not allow JavaScript to bypass the validation routines.	Do not rely solely on client-side validation to ensure the data that is expected from the browser is valid. Implement data validation routines that contain a default deny policy and restrict character classes to known good values.
App-3.	**High** *SQL Injection Allowed Arbitrary Database Access* The application allows for the injection of arbitrary SQL queries via a number of user supplied input fields. SQL injection occurs when user input is not filtered for escape characters and is then passed into a SQL statement. Several functions within the application dynamically create SQL statements from user input and perform no validation on the user input before doing so. Attackers could inject arbitrary SQL statements that modified the execution of the statements. This vulnerability allows an attacker read and writes access to any and all of the data stored within the database. An attacker	**Trivial** Many tools exist to automate the exploitation of SQL injection vulnerabilities and their functions range from mining database access to gaining command execution via specific database packages.	Implement data validation routines that contain a default deny policy and restrict character classes to known good values. Single-quotes should be escaped to prevent misinterpretation. Re-engineer the application so that it implements a least privilege model, i.e. use different database users to perform SELECT, UPDATE, INSERT etc. commands. In the event of an attacker injecting code into a vulnerable statement the privileges afforded would be minimised. This remediation step does not solve the SQL injection vulnerability, but it may limit the potential damage. Re-code the application so that it uses parameterised stored procedures to prevent variables from being interpreted as SQL commands. Stored procedures encapsulate reusable database procedures that are called with

ID	Vulnerability/Impact	Ease of Exploit	Recommendation
	could steal, modify or destroy any or all of the data using SQL injection. In addition an attacker can use SQL injection to remotely execute operating system commands with SYSTEM privileges on the RDBMS. This would allow an attacker to access the underlying operating system on the database server and make modifications or use this as a platform to attack other internal systems.		typed parameters. This provides several security advantages, by parameterising input parameters and type-enforcing them, user input is effectively filtered. In addition stored procedures can be executed under different security privileges from the database user. Using stored procedures does not eradicate the vulnerability completely; if user input is not parameterised or filtered, an attacker could still inject statements.
App-4.	**High** *Unauthorised Execution of Administrative Functions* Access to administrative functionality requires that a user be authenticated and in possession of a valid session token, however execution of the 'Add User' functionality does not verify that the user requesting the action is an administrative user. The application does not correctly check that the user is authorised to execute the actions requested, meaning that the different privilege levels are not effectively enforced. A low-privileged application user can add new users to the system and assign administrative privileges simply by logging into the application as a standard user and then directly accessing the "add user" asp page.	**Moderate** The functionality is exposed only to Administrative users after successfully authenticating to the application. An attacker would need to be in possession of valid user credentials, an associated session token and have knowledge of the correct URL for the administrative functionality. A valid user accessing the application script directly will allow trivial execution of the functionality.	Re-engineer the application logic and authorisation/access controls to ensure that requests to execute privileged functions require that the calling user have sufficient permission to do so.
App-5.	**High** *Persistent and Reflective Cross-Site Scripting* The application is vulnerable to "Reflective" and "Persistent" Cross-site scripting (XSS). A number of XSS vectors can be used to inject malicious code into	**Moderate** XSS attacks are widely publicised and automated exploitation tools are freely available on the Internet that would allow an attacker to steal cookies and session information. To craft targeted and bespoke attacks, an attacker would	The best way to protect a web application from XSS attacks is ensure that the application performs validation of all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be

ID	Vulnerability/Impact	Ease of Exploit	Recommendation
	the web pages that is displayed when viewed by other users of the application. XSS attacks occur when an attacker uses a web application to upload malicious code, generally in the form of a browser side script into a message box or other persistent application function that can be viewed and executed by another user or administrator of that application XSS vulnerabilities are used to attack users of systems, not systems or infrastructure components themselves. The malicious script can access any cookies, session tokens, or other sensitive information retained by the users browser and used with that site. In addition XSS vulnerabilities can be used to perpetrate Phishing attacks against users of the application through launching pop, up boxes and spoofed authentication pages for the application. If session credentials were retrieved, the attacker would then be able to access the application as the targeted user. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programmes.	require a higher level of skill and extensive knowledge of the HTML and scripting languages, such as (vbs and js), in order to exploit the vectors to their true potential.	allowed (white listing). The validation should not attempt to identify active content and remove, filter, or sanitize it (blacklisting). There are too many types of active content and too many ways of encoding it to get around filters for such content. Data that is retrieved from untrusted sources or user input, which is then displayed within the application, should be passed through an input sanitisation function that either escapes or converts the data into appropriate HTML. Using this method to encode user-supplied data can defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. .NET applications can use the HttpUtility. HtmlEncode method to encode output. HtmlEncode replaces characters that have special meaning in HTML-to-HTML variables that represent those characters. Encoded data does not cause the browser to execute code. Instead, the data is rendered as safe HTML. Limit the ways in which input data can be represented. This can help prevent malicious users from using canonicalisation and multi-byte escape sequences to bypass input validation routines. Use the appropriate encoding scheme for the context in which the data is being presented. Setting the character encodings for each page outputted could help reduce exposure to some variants of an XSS attack.ASP.NET allows you to specify the character set at the page level or at the application level by using the <globalization> element in the Web.config file.
App-6.	**Medium** *File Upload Functionality Allows*	**Trivial** Custom Trojans, virus and	Restrict the file types to only those

ID	Vulnerability/Impact	Ease of Exploit	Recommendation
	Potentially Dangerous File Types Activity identified application functionality that would allow an attacker to upload arbitrary files and have users of the system download the files to their local machine. There was no evidence of file type filtering or any content or virus checking of the uploaded files. The consequences of unrestricted file upload mechanisms can vary, including complete system compromise, an overloaded file system, forwarded attacks to backend systems, and simple defacement. This could potentially be used as a vector to distribute Trojans and attack users of the system, including for example Visual Basic Script (VBS) macro viruses. Embedded in MS word documents.	canned XSS exploits are readily available in the public domain to exploit this issue.	dictated by business requirements. Ideally this function should be implemented using code that actively checks and allows only valid file types based on a complete file inspection rather than a check of the file extension name. Implement on disk access virus checking and warn users of the potential risk of downloading content and educate on the enabling and running of non trusted macros.
App-7.	**Medium** *No Account Lockout for End-User Accounts* The application does not impose an account lockout threshold. An attacker could perform unlimited login attempts. An attacker able to successfully brute force entry into an account would be able to masquerade as that user within the application.	**Trivial** Automated Brute-force utilities exist for the majority of protocols that provide an authentication context, especially HTTP. They are fairly simple to configure and use and would require no specialist IT or hacking skills to successfully run. Without an account lockout mechanism to restrict the number of authentication attempts, an attacker would have an unlimited number of attempts to guess the configured password and unless their brute force attempt was discovered they would eventually guess even a strong password.	Implement an account lockout mechanism that restricts the number of login attempts. Industry best practise suggests a maximum figure of failed logon attempts to be between 3 and 5 attempts. The account lockout should either be implemented as a random period of time i.e. between 1 and 5 minutes to defeat automated brute force tools or should remain locked until unlocked by an administrator or through an account reset process that can be initiated by a user the implementation of which is subject to security policy requirements. All unsuccessful login attempts should be tracked in a database or other stateful component and should not be tracked in a session variable.
App-8.	**Medium** *Weak Password Complexity*	**Trivial** The application allows for the	Re-engineer the application to

ID	Vulnerability/Impact	Ease of Exploit	Recommendation
	Rules The application does not implement minimum complexity rules for passwords. This results in the user being able to select weak and easily guessed passwords that could be as simple as “password” or “123” If an attacker were able to guess a user's password, they would be able to masquerade as that user within the application.	use of weak passwords; depending on the strength of the user's chosen password, brute-force guessing could take an unskilled attacker as little as a few seconds to guess.	introduce functionality that allows the implementation of a minimum password strength that can be set in line with corporate security policy and best practice The policy should support passwords consisting of required characters (such as must have at least one lower and uppercase letter and a number); minimum length strings and prevents the use of illegal words (e.g. user name, dictionary words), etc. Generally a password policy for this type of web application would be to enforce passwords of at least 8 characters that contain upper case and lowercase letters as well as at least 1 number. The accounts should be routinely audited in order to prevent and detect the use of weak passwords. Passwords should be set to expire on a regular basis and the application should maintain a history of at least the last 3 passwords to prevent password repetition. Expire any temporary or default account passwords after first logon and force all users to change their password upon first successful authentication.
App-9.	**Medium** *Passwords Stored Unencrypted in Database* Passwords for users of the application are stored unencrypted in the database. Activity used a SQL injection vector in order to extract user credentials and password values. An attacker could use the same vector to impersonate authorised users and gain access to potentially sensitive information.	**Moderate** An external attacker would need the ability to query the backend database, via an attack vector such as SQL Injection or access to unencrypted database backup media. However an internal database or server administrator could access the database and gain access to users credentials and then masquerade as these users when accessing the application. Many tools exist to automate the discovery and exploitation of	Use a one-way cryptographic hash function to store passwords in the database rather than using plain text.

ID	Vulnerability/Impact	Ease of Exploit	Recommendation
		SQL injection vulnerabilities but some level of skill is required to perpetrate this specific attack.	
App-10.	**Medium** *Username Enumeration* The user log on page error messages presented when attempting to authenticate with valid and invalid usernames, presents an overly informative error page to a user that could be used to identify valid and invalid accounts through error based brute force techniques. Specifically, when an invalid account (i.e. one that does not exist within the application) is used, the login failure error message specifies that the account does not exist.	**Moderate** An attacker with an understanding of HTTP and web server services as well as a degree of coding/scripting skill could craft a program to automate the process and harvest valid user account names; however this could also be manually performed using a web browser.	Amend the application functionality to ensure that error reporting on failed log on attempts does not provide the facility for an attacker to use error reporting to enumerate valid account credentials. An example of an effective solution to this issue would be to provide an error message for failed logon attempts that regardless of whether the account or password is invalid simply presents an authentication failure message back to the user.
App-11.	**Medium** *No Session Timeout* Authenticated sessions within the application do not appear to time out within a reasonable period of time. Activity consultants found that user sessions were still valid after over 2 hours of inactivity. An individual user may be targeted by a deliberate or opportunistic attempt to 'hijack' an active session either through exploiting an unattended unlocked workstation or through a session hijacking attack.	**Moderate** An unattended workstation or logged in session cookie acquired through either inadequate desktop security or through a session hijacking attack would be required to take advantage of this vulnerability.	Automatically time out sessions after a period of inactivity less than fifteen minutes, or less in the case of administrative accounts.
App-12.	**Medium** *Forgotten Password Feature Open to Abuse* The application automatic password generator uses a predictable mechanism for generating passwords. The passwords generated use incremental values.	**Difficult** An attacker must have an account in order to discover this vulnerability and to have identified and have access to a valid email address that is associated with a valid user account.	Re-develop the application so that generated passwords are random and not predictable. Consider implementing a function that once a user has authenticated, forces the user to set their password to a value that conforms to a suitable password

ID	Vulnerability/Impact	Ease of Exploit	Recommendation
	The application also does not require a user to change the password upon first use. The forgotten password feature of the application resets the password of an account if the email address supplied matches a valid user account on the system and the new password can be used immediately. Due the predictability of application-generated passwords, an attacker could potentially reset an accounts password to a known/predicted value and gain unauthorised access.		complexity policy. Re-engineer the application so that it only resets an account password once the identity of the requester has been confirmed. Consider using a short cryptographically unique time limited token that is supplied to the application when authorising the password reset request. Consider requiring that the requester supply personal information that can be compared to data captured and stored during the registration process, and is required to be submitted before the password can be changed.
App-13.	**Medium** *Persistent Session Identifier* The application issues a session identifier to users if the client browser does not present a cookie containing a session identifier. The session identifier does not expire and is not destroyed upon session termination. An attacker could perform a 'session fixation' attack where by the attacker retrieves a valid session identifier and uses an XSS vector in conjunction with a social engineering attack in order to 'trick' a user into visiting the application and authenticating the session identifier set by the attacker. The attacker can then hijack the authenticated session.	**Difficult** Launching an effective 'session fixation' attack against end users of a website requires a thorough knowledge of JavaScript and web application technologies. The session token is written to a cookie. An attacker would need the ability to write arbitrary data to the cookie within the constraints of the Document Object Model. This can be achieved via a Cross Site Scripting (XSS) vector.	Destroy all session identifiers upon session termination and do not allow the re-use of identifiers. Issue different session tokens authentication for pre and post authentication session handling.
App-14.	**Low** *Previous Sessions Not Displayed to End Users* The application does not display the time or source of previous logins or login attempts to the user. In the event that an attacker compromises account credentials; by not displaying previous logins and login	**Trivial** An attacker does not need to take any action to benefit from this issue.	Provide the user with access to a list of their most recent successful and unsuccessful logins showing the time and duration of those sessions, with the source and a log of activity if possible. Following a successful login, display the time (and source, if meaningful) of the last successful

ID	Vulnerability/Impact	Ease of Exploit	Recommendation
	attempts to users, users may not be aware that their credentials have been compromised or that attempts have been made to break into their account. Successful attacks may go unnoticed and unreported, as the user has no ability to track historical account activity.		login. Also display the number of unsuccessful login attempts since the last successful login.
App-15.	**Low** *HTML Comments Information Disclosure* Comments within the HTML content delivered to the browser reveal information about the underlying platform and technologies in use. Each application page contains information pertaining to the developer, version of the application, last date and time revised and the operating system version and framework version required. This vulnerability constitutes information leakage and did not directly impact the security posture of the server. However the information is useful to an attacker in looking for other exploits and vulnerabilities with the application and its underlying infrastructure.	**Trivial** By simply selecting the "view page source" function in any web browser a user/attacker can see the comment information.	Review all application code before deployment into a production environment. All comments should be removed and where possible variable names changed to generic indices.
App-16.	**Low** *Web Server Default/Test Content* The application web server had default and development content present and accessible to users. The application web servers' default content contained exploitable vulnerabilities and revealed sensitive configuration information.	**Trivial** Valid requests for known default content locations will return the content. Example and test/development scripts can be executed by navigating to their location.	Remove all default, outdated, development and test content from production servers. There are a number of web server hardening tools available for different web server types that automatically remove default content such as scripts and help pages i.e. IIS lockdown which is available from Microsoft.
App-17.	**Low** *.NET Version and HTTP Server Type and Version Information Leakage in HTTP Response*	**Trivial** Valid HTTP requests result in HTTP responses containing	If possible, configure the server to provide false or no information. Remove all default content from

ID	Vulnerability/Impact	Ease of Exploit	Recommendation
	Headers The web server HTTP header responses displayed the type and version of software running on the host. Knowledge of the specific web server version allows an attacker to quickly identify known vulnerabilities that may be present in that particular server. This vulnerability constituted information leakage and did not directly impact the security posture of the server however it is useful and time saving to an attacker.	specific versioning information. An attacker can use common scanning tools or directly send valid HTTP requests to the server to cause it to reveal version information.	the system in order to reduce its known signature footprint. Use the IIS Lock down tool from Microsoft and utilise the built in URL scan tool within IIS6.0 to prevent the information leakage; this can also be achieved by editing the machine.config configuration file: Specify the enableVersionHeader directive as false (enableVersionHeader=false)
App-18.	**Low** *The Web Server Supports the HTTP TRACE Method.* The TRACE method is typically used for debugging; however it may also be used in cross-site tracing (XST) attacks. An XST attack can be used to compromise user cookie and session information. The presence of the method alone does not constitute a direct threat to the security of the system. An attacker must first locate an exploitable cross-site scripting (XSS) vector within an interactive application served from the web server in order to execute a cross-site tracing (XST) attack. Cross-site tracing attacks do not target the web server directly; attacks are launched against a third party or users of the system however a successful attack using the web server would damage reputation and customers trust in the site.	**Moderate** Launching an effective XST attack against end users of a website requires a thorough knowledge of JavaScript and web application technologies.	The 'URL Scan' tool available from Microsoft can be used to remove support for superfluous HTTP methods. Activity used the HTTP OPTIONS verb to enumerate the web server's supported options, removing support for this verb prevents enumeration of supported methods.
App-19.	**Low** *Cookie Secure Flag Not Set* The cookie provided by the	**Difficult** Engineering a connection between client and server communications in order to	Configure the application environment to only provide session cookies that are marked

ID	Vulnerability/Impact	Ease of Exploit	Recommendation
	server for session maintenance is not marked as secure. An attacker able to compromise a valid session cookie could masquerade as that user within the application without having knowledge of their credentials.	capture the cookie in transit is not a trivial task and is dependent on several environmental configurations; however once achieved, the extraction of the session data from the communications is trivial.	as secure.
App-20.	**Low** *No Password Ageing Policy* The application did not support a password ageing policy, allowing users to keep the same password indefinitely. This vulnerability greatly increases the chances of compromise of a users password through either unintentional disclosure by a user or through other electronic compromise.	**Difficult** An attacker with access to a user's credentials would be able to indefinitely compromise an account.	Implement a password ageing policy that requires users to change their passwords at least every 90 days.
App-21.	**Low** *Cookie HTTP-only Flag Not Set* The cookie provided by the server for session maintenance is not marked for transmission over HTTP only. The HTTP-only attribute for cookies prevents them from being accessed through client-side script. Firefox 2.0.0.5 and later, Internet Explorer 6 Service Pack 1 and later support the HTTP-only cookie attribute. Web browsers that do not support the HTTP-only cookie attribute either ignore the cookie or ignore the attribute, which means that it is still subject to cross-site scripting attacks. An attacker able to compromise a valid session cookie could masquerade as that user within the application without requiring knowledge of the users credentials.	**Difficult** For an attacker to gain access to the document.cookie object an exploitable XSS vector must exist. Launching an effective XSS attack against end users of the application requires a thorough knowledge of JavaScript and web application technologies.	Configure the application environment to only provide session cookies to that are marked as HTTP-only. Ensure that the application is free of XSS vulnerabilities and support for the TRACE verb is disabled. If the TRACE verb is enabled on the server and an XSS vector is identified and can be used to issue a TRACE request it is possible for an attacker to gain access to the cookie data as the data is passed in the HTTP request. Current browsers block TRACE requests using the XMLHttpRequest object.
App-22.	**Low** *SSLv2 Enabled on Web Servers*	**Difficult** In order to exploit an SSLv2 weakness, an attacker needs to be able to intercept and modify	Disable SSLv2 on the web servers. Microsoft's IIS web server secure communication channel

ID	Vulnerability/Impact	Ease of Exploit	Recommendation
	This version of SSL has several weaknesses that were addressed in v3 of the protocol. The most serious vulnerability is the lack of protections in the initial protocol handshake. This makes a man-in-the-middle attack possible. By exploiting weaknesses in SSLv2, an attacker could view or potentially modify the encrypted payload, resulting in a loss of integrity and confidentiality. The contents of entire sessions with the web server could be compromised, including passwords, session identifiers, and any information exchanged between the server and its clients.	the cipher text of a web server session, such as in a man-in-the-middle scenario. The attack is conducted by causing the client and server to negotiate a null, or a weak cipher.	configuration information is available in the following knowledge base article 187498.
App-23.	**Low** *Weak Ciphers Enabled in Web Server SSL Configuration* The web server is configured to allow the negotiation of weak SSL ciphers. By exploiting weak ciphers, an attacker could view or potentially modify the encrypted payload, resulting in a loss of confidentiality and integrity. The contents of entire sessions with the web server could be compromised, including passwords, session identifiers, and any information exchanged between the server and its clients.	**Difficult** An attacker would have to be in a position to observe the communications and then attempt to crack the negotiated key.	Disable ciphers with fewer than 64 bits of private key material on the web servers. Microsoft's IIS web server cipher suite configuration information is available in knowledge base article 216482.

Appendices

Appendix A. Impact and Exploitation Definitions 28
Appendix B. Vulnerability Details 29
Appendix C. Exploit Code 30
Appendix D. Port Scan Details 31
Appendix E. Tools List 32

Appendix A. Impact and Exploitation Definitions

A qualitative impact factor (High, Medium, or Low) has been associated with each vulnerability as well as an arbitrary assignment of the degree of skill that an attacker would require to exploit the vulnerability (Trivial, Moderate, Difficult). This should assist Acme Inc in their risk management efforts through the prioritisation of any remedial activity.

Activity's impact and ease of exploitation categorisation can be illustrated in the tables below:

Impact	Definition
Low	Minimal impact on the business if exploited. Information disclosed is of no significant detrimental value, no repudiation or legal consequence, minimal to no impact in regards to regulatory or standards compliance.
Medium	Moderate financial impact, possible legal consequences and reputational ramifications.
High	Significant financial loss, damage to brand and business identity through potential media involvement, exposure and compromise of data.

Ease of Exploit	Definition
Trivial	Publicly available exploit code or easily available tools to automate discovery and exploitation. Vulnerability is exploitable with the aid of minimal knowledge and or research.
Moderate	Requires some level of skill in basic programming or scripting and or a degree of knowledge and understanding of the target platform/software/application etc.
Difficult	Exploitation requires a highly skilled, motivated and apt attacker.

These technical factors do not equate to the actual risk, which must be calculated with due consideration of non-technical factors (such as physical security and the accessibility of the network) and the business impact of a compromise.

All risks and recommendations should be reviewed by Acme Inc and contextualised by persons with a more detailed knowledge of the environment and the risk management process of the organisation.

Appendix B. Vulnerability Details

This is an example report and as such does not contain screenshots or information on how to exploit the reported vulnerabilities.

This section of the report would contain supporting evidence and further technical detail for individual vulnerabilities discovered that warranted a more in depth analysis, demonstration of exploitation and more detailed explanation of what and how to remediate a given vulnerability.

Appendix C. Exploit Code

The example Proof Of Concept (PoC) code below was authored during the assessment to demonstrate the impact of the discovered vulnerability.

```
#!/usr/bin/perl
# Title: PoC.RemoteCodeExec.pl
# Purpose: Proof Of Concept Script to demonstrate the exploitation of a vulnerability.

my $var;
$var = “Activty Web Application Example Report”;
print $var x9000;
```

Appendix D. Port Scan Details

TCP and UDP port scans were conducted on all hosts within the scope of testing to establish running services on the hosts the results of which are detailed in the tables below.

IP Address	TCP Ports
10.0.0.10	80 (http), 443 (https)

Appendix E. Tools List

The table below is a list of commercial, open source and proprietary scripts, tools, applications and software suites that were used to perform the assessment.

Tool	Description
amap	Amap is a next-generation tool for assisting network penetration testing. It performs fast and reliable application protocol detection; independently of the TCP/UDP port they are being bound to. http://freeworld.thc.org/thc-amap
Bespoke	A collection of custom and proprietary scripts and programs authored by Activity consultants were used to perform the assessment.
Burp	Burp Suite is an integrated platform for attacking web applications. http://portswigger.net/suite
Nessus	Nessus is an automated vulnerability scanner. http://www.nessus.org/nessus/
netcat	Netcat is a featured networking utility, which reads and writes data across network connections, using the TCP/IP protocol. http://netcat.sourceforge.net/
Nikto	Nikto is an Open Source (GPL) web server scanner that performs comprehensive tests against web servers for multiple items. http://cirt.net/code/nikto.shtml
Nmap	Nmap ("Network Mapper") is a free and open source (license) utility for network exploration or security auditing. http://nmap.org
Paros	A Security tool for performing web application vulnerability assessments. http://www.parosproxy.org
SSLScan	SSLScan queries SSL services, such as HTTPS, in order to determine the ciphers that are supported. http://sourceforge.net/projects/sslscan
tcpdump	Tcpdump prints out a description of the contents of packets on a network interface that match the Boolean expression. http://www.tcpdump.org
@stake WebProxy	@stake WebProxy is a cross-platform/browser security tool for use in auditing web sites. No longer available.

Tool	Description
WebScarab	WebScarab is a framework for analysing applications that communicate using the HTTP and HTTPS protocols. http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project