UCIT Quality Assurance Virtual Environment

by

Julia Noelle Coleman

Submitted to
the Faculty of the School of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

© Copyright 2015 Julia Noelle Coleman

The author grants to the School of Information Technology permission
to reproduce and distribute copies of this document in whole or in part.

Julia Noelle Coleman  4/20/2015

Mark Stockman, Faculty Advisor  4/20/15

University of Cincinnati
College of
Education, Criminal Justice, and Human Services
School of Information Technology
April 2015
Contents

Abstract ......................................................................................................................................................... 1
Introduction .................................................................................................................................................... 2
  Problem .................................................................................................................................................... 2
  Solution .................................................................................................................................................. 2
  Credibility ............................................................................................................................................... 3
  Goals and Methods ................................................................................................................................. 3
  Overview ............................................................................................................................................... 3
Discussion ................................................................................................................................................... 4
  Project Concept ..................................................................................................................................... 4
  Users ..................................................................................................................................................... 5
Design Objectives ....................................................................................................................................... 6
Technical Approach .................................................................................................................................... 9
Budget ...................................................................................................................................................... 12
Time Line ............................................................................................................................................... 13
Goals ....................................................................................................................................................... 14
  Problems and Limitations ..................................................................................................................... 14
Conclusion ................................................................................................................................................ 16
Works Cited ............................................................................................................................................. 17
Appendix A: Test Plan and Results ........................................................................................................... 18
Appendix B: Software Comparisons for Decision Making ....................................................................... 23
  Sauce Labs ............................................................................................................................................ 23
  Browserstack ....................................................................................................................................... 25
Appendix C: Quality Assurance Process Document .................................................................................. 29
Executive Summary ................................................................................................................................... 31
  Objectives ............................................................................................................................................ 31
  Keys to Success .................................................................................................................................. 31
Description of Department .......................................................................................................................... 31
  Company Ownership/Legal Entity ........................................................................................................ 31
  Location ............................................................................................................................................... 31
  Hours of Operation ............................................................................................................................... 31
  Products and Services ........................................................................................................................... 31
Testing Process ............................................................................................................................................ 31
  Requesting Quality Assurance Analysis ............................................................................................... 31
    TeamDynamix Form ............................................................................................................................ 31
Accessing the Quality Assurance Environment ........................................................................................ 34
Testing Types ........................................................................................................................................... 34
  Security Testing .................................................................................................................................. 34
  Accessibility Testing ............................................................................................................................... 34
List of Tables and Figures

Figure I) QA and SDLC ................................................................................................................................................ 4
Figure II) User Profiles.................................................................................................................................................. 5
Figure III) Use Case Diagram........................................................................................................................................ 6
Figure IV) Simple Network Diagram ............................................................................................................................ 9
Figure V) Multi-Level Diagram .................................................................................................................................. 10
Figure VI) Virtual Machine Backups ........................................................................................................................... 11
Table I) Proposed Budget ............................................................................................................................................ 12
Figure VII) Gantt Chart ............................................................................................................................................... 13
Abstract

UCIT Software Applications Development, located on the medical campus of the University of Cincinnati, had a small Quality Assurance Team with loose definitions of procedures and requirements. Realizing it was important to have an organized and efficient QA process to follow in order to produce secure and quality applications for customers, certain measures needed to be taken. With efficiency and convenience in mind, Noelle Coleman researched best practices in quality assurance and created a process that fit with the UCIT QA team. Additionally, she set up a virtual environment with all of the required tools and documentation. With the approval of UCIT, Noelle implemented the virtual environment so that the team was able to start using it in its daily applications.
Introduction

Problem

Relative to the rest of the department, the Quality Assurance Team in the UCIT Software and Applications department is fairly new. There are few members of the team and several applications that request quality assurance analysis every week.

Currently, the analysis is supposed to include a security scan, a 508 compliance/accessibility scan, and a user test. Reports of all should be sent to the developer in order to start the remediation process. However, there is no set format for these reports. In fact, there is not a set process for the QA team at all. It seems more like a set of guidelines where, in order to move to production, the only requirement is that it passes a security scan.

There are several problems with the system that is in place now. For one, there is a lack of organization that, with only two or three people on the team, makes it difficult to keep track of all the parts of the quality analysis for even one application, let alone the three or four that are generally in queue. The poor organization leads to a lack of efficiency, which can compound and become overwhelming for the department. Ultimately, the biggest issue is that applications get moved into production with problems, whether functionality or usability, and, because of the cyclical nature of software and application development, it all adds up and nothing gets fixed.

Solution

The proposal to fix this was to come up with a complete QA process and environment for the department to use. The written process defines the tools that the team uses, documentation for the use of tools, the requirements of the request from the developer, and the requirements for the reports back to the developer. As far as the actual environment goes, using one virtual
environment, with all the tools included, creates organization and maximizes efficiency. While the QA team spends the most time in the environment, the developers also have access to a portion of the system to use for certain types of testing before submitting a formal request for analysis by the QA team. The documentation for this overall QA process can be seen in Appendix C of this document.

Credibility

Information obtained from project management courses at the University of Cincinnati was of great help when it comes to organizing this project since the solution lies largely in the process. Along with courses, working in the department itself for almost two years allowed for intimate knowledge of how the teams work together, what the current process is, and how the process could be changed in order to best accommodate the group.

Goals and Methods

The goal of this project was to create an environment for different quality tests to be completed for the software applications and development department at the university. In order to do this, research was be conducted on the best practices of quality assurance in a software development environment. The research was put into practice in order to find tests, tools, and an organizational scheme for the environment, as well as an organizational process for the department.

Overview

The remainder of this final report outlines in detail how the project was be completed. The report includes sections on concept, design, approach, budget, timeline, and limitations.
Discussion

Project Concept

The idea of this project is to completely overhaul and organize the way that the quality assurance team at UCIT Software Applications and Development runs itself. After working in the department for over a year, it became apparent that something needed to be done, as it was difficult to get work done because of the unorganized manner in which tasks were accomplished.

Quality Assurance is a process that should be incorporated throughout the entire Software Development Life Cycle, as illustrated in figure 1, below. By creating a single environment for the quality assurance team to complete their work in, the efficiency and productivity of the entire department will benefit.

Figure 1) Quality Assurance as it relates to the Software Development Life Cycle
Users

User Profile

<table>
<thead>
<tr>
<th>User Profile Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application: UCIT Virtual Quality Assurance Environment</td>
</tr>
</tbody>
</table>

Potential Users:


Software and Interface Experience:

The typical user will have experience with the Windows Server operating systems and Office software. They will have experience using the security tools and the compliance software.

Experience with Similar Applications:

The experience should be somewhat familiar because they will be working in a remote virtual environment like the one that is currently in place. The application that will be used here will be similar to working in any Windows environment with current software.

Task Experience:

The users should have experience with this task already. Nothing is being modified, merely consolidated so that the user has a more comprehensive view of the task in its entirety. The tasks they are completing will not be any different overall, but the path they take to get to the finished product will contain fewer trials. If a user is unfamiliar with a piece of the process, the documentation will make their experience less frustrating.

Frequency of Use:

This application will be used on a daily basis by the Quality Assurance Analyst. The developers will not use it as often, but could use it frequently.

Key Interface Design Requirements that the Profile Suggests:

- The design must be intuitive and designed with the current/old system in mind
- The process must be documented well, as there was no good process before
- The desktop experience must be the same
- Must be accessible from multiple locations

Figure II) User profiles

Seen in Figure III below, there are two main types of users for this environment and a third that is possible but is less likely. First, there is the Quality Assurance Analyst. The Analyst will be able to run testing software, view all documentation, create reports from the tests, and view the reports
created. The Developer user is the user who is submitting their application for testing. This user type will be able to run tests and view the documentation for the software. However, they will not be able to create reports, per the process. They will only be able to view the reports when they are completed. The last type of user would be a manager of the developer. These users would only be able to view the documentation and reports that have been created.

**Use Case Diagram**

![Use Case Diagram for the UCIT Virtual QA Environment](image)

**Figure III) Use Case Diagram for the UCIT Virtual QA Environment**

**Design Objectives**

**Organization**

The organization aspect of this project is both high level, covering the project as a whole by organizing the entire process of quality assurance testing, and low level, covering the way the files are organized once testing is complete.
Software

Software is used for the majority of the different types of testing. This includes compatibility (cross browser) testing, Accessibility (508 compliance) Testing, Security Testing, and Usability Testing.

Accessibility Testing

Accessibility testing is making sure that the application being tested is accommodating to users with disabilities. For example, a user who is blind must be able to navigate a website using a screen reader, meaning alt tags need to be in place for all images, heading tags need to be used in sequential order, etc. This is also known as 508 Compliance, from section 508 of the amended Rehabilitation Act of 1973. It simply requires federal agencies to ensure the technology they use is accessible to those with disabilities. Although the University of Cincinnati is not a federal agency, compliance with Section 508 is gaining momentum. It is important, especially in a university setting and even more so at a hospital, that those people with disabilities are able to access all of the necessary information from the websites that the UCIT Software Applications Development team creates.

The software that was decided on for accessibility testing is SortSite by PowerMapper. For the testing portion of the project, the company offers a free trial of its software for single use. After the trial, the department decided to buy a perpetual single use license, meaning it is only a one-time cost but can only be used on one machine.

Compatibility Testing

For the purpose of websites and web applications, compatibility testing ensures that there are no major differences between different browsers. This is also known as cross-browser or browser compatibility testing. At the University of Cincinnati, there are several ranges of computer environments used to access the university websites. They can range from Safari on a Mac, to
Firefox on a Linux machine. It is important for the websites that are created by the UCIT Software Applications Development department to function on different browsers in different environments.

In order to do this testing, there are several different choices. Currently, the analysts will go to the website on different browsers they have installed on their machines and test some of the main features and functionality. This is not efficient and will not yield the best results. This project proposes to create somewhat of a virtual browser farm using virtual machines and scripts to run the machines. This will be a home built tool. However, this portion of the project is mainly for research, as the UCIT department has decided to purchase a license for a software called SauceLabs. SauceLabs was chosen after a comparison done on different software types. This research can be seen in Appendix B of this paper. This software allows for automation of testing and side by side comparison of browsers. For the purpose of this project, this is the best decision because of the time frame, creating an efficient system quickly.

**Security Testing**

Security testing tests the website and the server configuration in order to find any security flaws that may exist so that they can be corrected as soon as possible. The UCIT Software Applications Development department currently does security testing on all new sites going into production. The information security team at UCIT is responsible for scanning the server configuration.

To scan the sites, Cenzic Hailstorm is the piece of software that is in use. Hailstorm allows for the creation of automated tests based on predefined test cases. In general, these test cases are in line with the OWASP Top 10 security risks for websites.

**Documentation**

Documentation is provided for each type of software used in the environment. This includes information on how to use the given software and how to fix the problems uncovered by the tests
that were completed. There is also documentation for the process an application should go through to be considered completely tested by quality assurance. This means submitting an application to the quality assurance team for testing, completing the testing, creating a report for the developer, and submitting the report back to the developer.

**Technical Approach**

**Design requirements**

The main design requirement for this project is that it uses one single environment inside of a Windows Server 2008 instance. This requirement exists because of current licensing in the department. Beyond the operating system, it is important that the environment is easily accessible by both the testers on the quality assurance team and by the developers in the department. The system also must be intuitive to use and not so much different than what the team is used to already. This is important because the department cannot fall behind in testing while all personnel are learning and struggling with new testing environments. By using Windows Server 2008 and working with the server team, the environment will be both familiar and accessible to all those who need access to it, Figure III below.

**Simple Network Diagram**

![Diagram](figure-iv.png)

*Figure IV) A simple network diagram for the multiple user machines and the virtual quality assurance box*
Another important technical aspect of the project is backup and disaster recovery. As seen in figure V, below, there are several different components that make up the environment.

Because of the heavy incorporation of virtual machines in this project, images and snapshots are able to be used for backup and recovery as necessary. There is an overall backup image of the system as a clean image, taken just after all the necessary components were put into place, seen in figure VI. Snapshots will be taken periodically so changes can be saved and restored to a recent version if needed. The datacenter will also be taking backups from the physical host.
Figure VI) Environment virtual machine and the clean backup image

Testing

The testing of this project was mostly completed by the QA team that it is being designed for. Several different levels of testing were implemented during the course of this project.

First, testing was done on all of the software components to ensure proper functionality that is in line with the testing requirements of the department.

Second, the documentation for the software needed to be tested. Documentation was tested for understandability, accuracy, and ease of use. Understandability was tested, much like any other piece of writing, by proof reading and making sure it has all the proper components of a technical document. Accuracy was tested by multiple members of the QA team who have used, and are familiar with, the software. They read through the documentation and made sure that the information given in the document matched the knowledge they have of the software. Ease of
use was tested by asking a new member of the QA team and a developer who was unfamiliar with certain pieces of the software, to complete a set of tasks in the different programs by reading the respective documentation available to them.

Third, testing was done to ensure that the entire environment is comprehensive and easy to use. This test sequence was performed by both a member of the QA team and by a developer in the department.

The results of the test for the documentation were the most helpful in rewording and adding extra detail. There was also some extraneous detail removed from the documents. As for the software, there was both positive and negative feedback. Some of the negative feedback from the testing was added as stretch goals for the project, as it would have been too much to complete in the scope of the current project timeline. The testing results and a more complete test plan can be seen in Appendix A at the end of this document.

**Budget**

<table>
<thead>
<tr>
<th>Item</th>
<th>License Per Year</th>
<th>License Per Month</th>
<th>Cost Per Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrustWave App Scanner</td>
<td>$26,000</td>
<td>N/A</td>
<td>$26,000</td>
</tr>
<tr>
<td>SortSite</td>
<td>$349</td>
<td>N/A</td>
<td>$349</td>
</tr>
<tr>
<td><strong>Compatibility Testing VMs</strong></td>
<td>$0</td>
<td>N/A</td>
<td>$0</td>
</tr>
<tr>
<td><strong>Sauce Labs</strong></td>
<td>$0</td>
<td>$149</td>
<td>$1,788</td>
</tr>
<tr>
<td>Virtual Box</td>
<td>$0</td>
<td>N/A</td>
<td>$0</td>
</tr>
</tbody>
</table>

| Total with VMS          |                  |                   | $26,349       |
| Total with Sauce Labs   |                  |                   | $28,137       |

** Either Compatibility Testing VMs OR Sauce Labs will be used **

Table I) The proposed budget for the project over a year
Software
For the period of time while this project was being built, all of the licensing for the software was free because trial periods are available for all of the decided upon software. However, in a real world application, the software that was chosen all either has a monthly cost or a large up-front cost to purchase a perpetual license, as seen in Table I above.

Personnel
The cost to design this project assumed 20 hours per week for 28 weeks by a Quality Assurance Specialist, with a salary of $48,733 annually. This would mean that the budget needs to plan for $26,200 for the developer of this project if someone needs to be employed specifically for this project. Discussion about paying for testers would also need to take place. For this implementation of the project, it was assumed that the testers are already employed by the university.

Time Line
Gantt Chart

<table>
<thead>
<tr>
<th>October</th>
<th>November</th>
<th>December</th>
<th>January</th>
<th>February</th>
<th>March</th>
<th>April</th>
<th>May</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiate</td>
<td>Set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9/6/14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Wed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9/24/14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mon 9/22/14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mon 9/22/14</td>
<td>Tue 9/23/14</td>
<td>Mar 10/27/14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tue 9/23/14</td>
<td>- Fri 3/13/15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wed 2/19/15</td>
<td>Thu 4/30/15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fri 5/1/15</td>
<td>Fri 5/8/15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure VII) Proposed Gantt Chart for the beginning of the project
Goals

The project had a proposed completion time of approximately nine months. Throughout the project it had become apparent that there were changes that needed to be made and some unexpected changes that altered the timeline slightly. Setting goals became an important part of the project in order to stay on track since some changes had to occur. At the halfway point of this project several goals had been met, including creating a mock environment for the software to live on and determining all of the software that will be used. There were also several goals that had yet to be met, such as creating a browser testing platform in place of the pre-built software for sale. Stretch goals were set in place for the virtual environment because it became apparent that the current timeline would have not allowed for their completion. The stretch goals include automation of the home-built browser testing software and also for the organizational processes. Eventually, an interface for users to create reports instead of editing a word document would be extremely helpful in the environment, but it is not currently necessary for the process. Another goal that will be simple to reach is to create a web page for the documentation to be stored. This would allow for even easier access and the ability to search documentation more quickly than searching a pdf or word document. Another bonus to having a web page for the documentation is that there would be the chance for collaboration and constant feedback on how the quality assurance process is serving the department. The documentation for this project will be added to the SharePoint site for the team for now and updated as needed.

Problems and Limitations

The only major constraints to this project are that a Windows environment will need to be used because of current licensing, technologies, and personnel. There are also some tools that are predefined because of licensing as well. These are simple constraints to work with and no problems are anticipated because of this.
A couple weeks shy of the halfway point in the project, mobile testing software and load testing software were both discussed in a team meeting. These two pieces had not been integrated into the plan for this project, but it was clear that they needed to be. Meeting with the supervisors of the QA team allowed for time to come up with a plan to add these into the project. However, after testing several different application and finding that either none of them worked with the network or worked up to standards, this was moved out of the scope of this project and into a stretch goal that will continue to be researched by the department.

A week after adding the two testing components, another meeting was had with the quality assurance supervisors. The original plan for browser testing had changed from the original plan of building a home grown configuration using virtual machines. The team was now more interested in using pre-built software. To come up with a solution to this quickly, research was done on several different pieces of software commonly used to test cross-browser compatibility and then presented to the team. For the project, an open source system using virtual machines was built in order for research purposed and to show the difference between that and what exists currently. Comparisons of the different software which was considered for testing can be seen in Appendix B at the end of this document.

While continuing through this project, it became more apparent that any automation would have to be something focused on much later in the department lifetime, mainly because of the amount of applications that need to be dealt with and because of the lack of employees in the department. Unlike automation testing for a company whose QA department is only in charge of a few applications, tests would have to be built and redone almost daily and, unfortunately, UCIT does not currently have the man power for this.
Conclusion

With a proper process and documentation, this project enables the team to be much more efficient in their work. It creates a more productive environment which outputs quality applications for a relatively low cost. The UCIT Quality Assurance Virtual Environment accomplished this by using different software applications and detailed documentation to create an easy to understand system.
Works Cited


Appendix A: Test Plan and Results

Senior Design Project Management II, 15SS
School of Information Technology
College of Education, Criminal Justice, and Human Services

Testing Methods and Requirements
Noelle Coleman
Testing Phase Plan

Upon finishing the initial development of the environment, the different requirements will be tested by users in several different steps. First, the users will go through the testing as documented. The results will be looked over and any blatant problems will be resolved. After this alpha testing phase, users will sit down in a focus group type of environment in order to gather ideas on improvements that may be made based on either what is easier for the users or what fits the requirements better than the system in place. Changes will be made again and then there will be a beta testing phase in which the testers will run through the process again, but this time they will have documentation that reflects the changes that will been made based on the previous testing phases. This process may be repeated as needed.

Testing Requirements based on deliverables
1. **Windows Server 2008 Virtual Machine**
   a. Users can log in using their work ID

2. **Quality Assurance Process Document**
   a. Users understand the process that they need to go through in order to get their site through quality assurance
   b. Operating system and programs can be recovered through backups
   c. Documentation and past tests can be restored through backups

   a. Software is installed and runs as expected
   b. Users can run software based on documentation
   c. Users can fix issues by referring to documentation

4. **Quality Testing**
   a. Users understand what is expected by them based on the form they need to fill out
   b. Users can fix all issues based on documentation

User Testing Plan

Testers

A “Tester” is someone who represents a member of the Quality Assurance team. In their daily tasks, it will be required of them to use the machine and run the tests, as well as be aware of how to remedy any issues that may come up, in case a developer has questions. While much of this should be knowledge they possess as part of the job title, assistance can be found in the documentation of the software that they are using.

A “Developer” is someone who represents a member of the development team. In their daily tasks, they will not necessarily need to deal with quality assurance reports and remediation. They will need to be familiar with it, however, for moving items to production which, depending on the project, could range from once or twice a week to once a month or more.

1. **Tester user**
   a. Remote into the machine “hailstormclient.ad.uc.edu” using AD username/password combination
   b. Using test URL, run compatibility test, referring to documentation for assistance
   c. Create a report for the compatibility test, referring to documentation for assistance
d. Using test URL, run accessibility test, referring to documentation for assistance
e. Create a report for the accessibility test, referring to documentation for assistance
f. Using test URL, run security test, referring to documentation for assistance
g. Create a report for the security test, referring to documentation for assistance
h. Using test URL, test the site for quality, referring to documentation for assistance
i. Create a report for the quality test, referring to documentation for assistance
j. Follow the QA Process document in order to notify developers of their passed or failed tests

2. Developer User
   a. Fill out request for analysis for test URL, referring to documentation for assistance
   b. Obtain reports for test URL, including security, accessibility, compatibility, and quality test
   c. Correct any issues, referring to documentation for assistance
   d. Request a re-scan of the test URL
Testing Requirements Checklist

- Windows Server 2008 accepts AD user login
- Windows Server 2008 has repositories for all four different testing types
- Windows Server 2008 has compatibility testing software installed
- Windows Server 2008 has compatibility testing VMs installed
- Windows Server 2008 has accessibility testing software installed
- Windows Server 2008 has security testing software installed
- Security Testing document is complete and free from error
- Compatibility Testing document is complete and free from error
- Accessibility Testing document is complete and free from error
- Quality Report template is complete
- Quality Assurance Process document is complete and free from error
- All documentation can be retrieved by backups if necessary
- Environment can be retrieved from image if necessary
<table>
<thead>
<tr>
<th>User Test Task List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Type</strong></td>
</tr>
<tr>
<td>Tester</td>
</tr>
<tr>
<td>Tester</td>
</tr>
<tr>
<td>Tester</td>
</tr>
<tr>
<td>Tester</td>
</tr>
<tr>
<td>Tester</td>
</tr>
<tr>
<td>Tester</td>
</tr>
<tr>
<td>Tester</td>
</tr>
<tr>
<td>Developer</td>
</tr>
<tr>
<td>Developer</td>
</tr>
<tr>
<td>Developer</td>
</tr>
</tbody>
</table>
Appendix B: Software Comparisons for Decision Making

Sauce Labs

Pro: You are able to choose whichever operating system and browser you want
Con: Many of these will go unused

Pro: Bug tracking
Pro: Built-in screenshot

Pro: record and share demo sessions with coworkers, customers, or users

Pro: saved sessions for version tracking

Pro: See previous screencasts
Browserstack

Live

Quick start

Test a public URL

demo.uc.edu/nightride

Windows 7  
IE 8.0  
Start testing

Desktop  
Mobile & Tablet Emulators

Windows 8.1  
Windows 8  
Windows 7  
Windows XP  
iOS 7  
OS X Yosemite  
OS X Mavericks  
OS X Mountain Lion  
OS X Lion  
OS X Snow Leopard

Pro: Live test any URL
Pro: Choose recent OS

Quick start

Test a public URL

demo.uc.edu/nightride

IE 8.0  
Start testing

Pro: Choose from many different browsers

Resolution

800 x 600  
1024 x 768  
1280 x 1024

Pro: Choose resolution for scalability
Example of issues that this catches

Can see what you’re testing while you’re testing it and have different options. Also has a bug tracker
Responsive

To test your internal web server or design files, setup local testing.

Google Nexus 4

Screen size: 5.96
Appendix C: Quality Assurance Process Document

Quality Assurance Process Document
Written by Noelle Coleman for the University of Cincinnati Information Technology
Software Development department on January 4, 2015

Last updated April 13, 2014
Table of Contents

Executive Summary..................................................................................................................................................... 31
   Objectives
   Keys to Success

Description of Department .......................................................................................................................................... 31
   Company Ownership/Legal Entity
   Location
   Hours of Operation
   Products and Services

Testing Process............................................................................................................................................................ 31
   Requesting Quality Assurance Analysis
   Accessing the Quality Assurance Environment
   Testing Types

External Resources ....................................................................................................................................................... Error! Bookmark not defined.
   For questions about this document
Executive Summary
The purpose of this document is to outline the process a site must go through to be released by Quality Assurance into production. This process is set by the Quality Assurance group of the Software Development team of the University of Cincinnati Information Technology department and covers how to request an initial quality analysis, the tests that a site will go through, and how to obtain the reports that are the product of this analysis.

Objectives
This document intends to educate developers, testers, and managers of the process needed to ensure a quality site is released from the department and into production, whether going into production for the first time or merely pushing an update.

Keys to Success
Developers, testers, and managers will need to work together to make sure communications remain open and updates remain current. This will ensure organization and timely releases.

Description of Department

Company Ownership/Legal Entity
The Quality Assurance group is a part of the UCIT Software Development team at the University of Cincinnati.

Location
The Quality Assurance department is located on the Medical Campus of the University of Cincinnati in the Medical Sciences Building at 231 Albert Sabin Way in room G005H, which is inside of the Health Sciences Library, on the second floor.

Hours of Operation
Someone will be available to process requests and run analyses Monday through Friday during the core working hours of 9:00 am to 4:00 pm.

Products and Services
The Quality Assurance Team provides testing services for websites including security testing, accessibility testing, compatibility testing, and quality testing. Some of these services use software for which the university has purchased a license and some of the services are done using reports created by the Quality Assurance Team itself.

Testing Process

Requesting Quality Assurance Analysis
Any site that is going from development into production will be required to go through Quality Assurance if it meets the needs that are defined in the Change Management process for the department. As a general rule, all sites which are being migrated into production will need a quality check (user testing) for spelling and common mistakes. Security, accessibility, and compatibility testing will only be required if a code change has been made to a site or if a site is entering the production environment that has not gone through the QA process before.

TeamDynamix Form
Any request made will go through the team collaboration site that the department is using, called TeamDynamix, either through the user or client portals.

Using the TDNext User Portal
1. Log into https://uc.teamdynamix.com/TDNext/Home
2. Choose the “Tickets” tap at the top of the page

3. Select “+ New Incident”

4. Change the “Service” to “Application Development, Partnerships and Innovation / Quality Assurance Service” and the “Ticket Type” will update automatically
5. Change the “Responsible” field from “Application Development – Quality Assurance (Group)” to the person in charge of running scans. In this case, “Noelle Coleman”

6. Fill out the rest of the form according to the information from the application that the scan is being requested for and submit.

All updates and requests from the QA team regarding the ticket request will be entered through TeamDynamix. Emails will be sent when the TeamDynamix ticket is updated, but this must be done manually so make sure to stay current on any requests that you may enter.
Accessing the Quality Assurance Environment
1. Open “Remote Desktop Application” and connect to hailstormclient.ad.uc.edu using your AD credentials

2. Once logged in, you can access the software or documentation that you are looking for by browsing from the start menu

Testing Types
Websites going from development to production are required to go through certain testing to ensure their safety and quality. The four separate tests that the QA team uses for analysis are as follows:

Security Testing
Security testing is in place to ensure that all the applications produced by UCIT Software Development are free from security holes and vulnerabilities.
For more detailed information about the security testing process, see the external document.

Accessibility Testing
Accessibility testing is in place to ensure that all the applications produced by UCIT Software Development are available to people with disabilities.
For more detailed information about the accessibility testing process, see the external document.

Compatibility Testing
Compatibility testing is in place to ensure that the applications produced by UCIT Software Development function as they are supposed to across all different browser types.
For more detailed information about the compatibility testing process, see the external document.

Quality Testing
Quality testing is in place to ensure that the applications produced by UCIT Software Development are able to be used the way they are intended, based on requirements documents, and understandable and intuitive to the end user.
For more detailed information about the quality testing process, see the external document.
Contents
Summary ..................................................................................................................................................................... 37
Testing Process ............................................................................................................................................................ 37
Software ................................................................................................................................................................... 37
Tests ........................................................................................................................................................................ 37
Reports ..................................................................................................................................................................... 37
External Resources ...................................................................................................................................................... 37
For questions about this document .......................................................................................................................... 37
Summary
Quality testing is in place to ensure that all applications produced by UCIT Software Development are viewable and functional in the way that was intended and in a way that is easy to understand and navigate by the end user.

Testing Process
Quality testing is in place to ensure that all users will have a positive experience using our websites and applications, so it is mandatory that all sites that will go out into a production environment go through this process.
For more information on requesting quality tests, see external “Quality Assurance Process” documentation.

Software
There is not a set software for this type of testing, only the application which is being tested and the browser which is it being tested on. Note that only one browser will need to be used for this testing process as it is for testing functionality and all cross browser testing will be completed using the “Compatibility Testing Process” documentation as guidelines and rules.

Tests
In order to test the application, the tester will load the website or application in the browser. Depending on the project, the tester will have requirements, instructions, or test cases which will need to be tested against. This testing will include making sure that all functionality of the application works as intended. Checking for understandability is also important in this process. While user acceptance testing may take care of this and is not in the scope of the QA department, unless a QA member is requested to help guide user acceptance testing, going through the site as though the tester is a normal user is important before going to production.

Reports
The reports for quality testing are manually created by the QA department tester. There will be a set format to follow, which can be found as an external document called, “Quality Testing Report”, as a guideline and which can be viewed by the developer and management. The report will include results of testing against test cases, if the project called for them, or requirements and will include comments on any other usability features which may have not been covered by other documentation. Screenshots and detailed comments are important in these reports, as they will be used to reproduce any errors.

External Resources
For questions about this document:
Noelle Coleman, Software Applications Developer I, UCIT Software Development
colemaj6@ucmail.uc.edu, (513) 558 – 6048
Summary
Security testing is in place to ensure that all applications produced by UCIT Software Development are free from security holes and vulnerabilities. This document is in place to explain the testing process, including the software used for testing, the types of tests that are done, the reports that are used, and instructions for use of the software and reports, and understanding the results of the test.

Testing Process
Security testing is an important and mandatory process in the UCIT Software Development department. For information on requesting a scan, see external “Quality Assurance Process” documentation.

Software
The software that is used to perform security testing is known as Hailstorm. The desktop application is powered by Cenzic, a company recently acquired by Trustwave. The web application is named as Trustwave App Scanner. The two applications are completely compatible and use the same login credentials.

Tests
The tests that are used to scan the security of websites and applications going through the QA process are based on the OWASP Top 10. The name given to these tests that are developed for Hailstorm are “SmartAttacks”. The QA department has identified 42 different SmartAttacks to be used and has broken them up into lettered sets to maximize efficiency and organization of the security scan process.

Set A
- Form Caching: HTTPS Web servers supporting weak ciphers are vulnerable to a breach of secure communication. Such servers, if they are using expired certificates, expose users of applications hosted on them to Phishing attacks. This SmartAttack reports HTTPS Web servers which support weak ciphers. The SmartAttack also checks if SSLv2 is supported. The SmartAttack reports hosts for which the certificate has expired or is not issued to the domain of Web server.
- Forms Submitted: Forms submitted without using POST is a vulnerability where a form with sensitive information is submitted via an HTTP GET request. This type of submission can result in disclosure of the submitted values. The SmartAttack reports all form submissions made using a GET request. The significance of such findings is dependent on the sensitivity of the submitted data.
- Login Redirect: This SmartAttack reports vulnerability if 302 redirect is not found on login page.
- Non-SSL Form: Non-SSL Form is a vulnerability caused by allowing submission of sensitive form data without using SSL encryption. The SmartAttack observes and reports any form that is not submitted via SSL. The significance of such findings is dependent on the sensitivity of the submitted data.
- Non-SSL Password: Non-SSL Password is a vulnerability caused by a failure to submit passwords via SSL. The SmartAttack inspects all requests where passwords are submitted and reports those where SSL is not used for the submission.

Set A2
- Non-masked Password: Non-Masked Password is a vulnerability typically caused by a failure to use the HTML password input field type which masks passwords from being displayed. The SmartAttack matches form fields (by name) which appear to be asking for passwords, and reports any which are not masked.
- Password Autocomplete: Password Autocomplete is a vulnerability caused by allowing caching of passwords by browsers. The SmartAttack inspects responses from the application to identify if autocomplete is explicitly set to “off”.

Set B
• Blind SQL Injection: Blind SQL Injection is a vulnerability caused by a Web application sending user input into a SQL query without validation. It is a type of SQL Injection vulnerability, where an attacker infers information from differences in responses observed for different injections. This SmartAttack injects pairs of SQL injections and looks for evidence of different responses.

• DOM Based Cross-Site Scripting: DOM based Cross-Site Scripting is an Cross-Site Scripting (XSS) variant where the malicious script is executed as a result of modifying DOM in a victim’s browser. The malicious injected script is run by client side code, modifying the DOM environment. This attack is a client side code flaw while reflected and persistent Cross-Site Scripting are server side flaws.

• SQL Disclosure: SQL Disclosure is a vulnerability caused by a Web application sending user input into a SQL query without validation. It is a type of SQL Injection vulnerability, where an attacker is able to extract much more information than he is authorized for, simply by using SQL statements that always evaluate to true in the input. This SmartAttack injects such statements and looks for evidence of extra information getting revealed in the response.

• SQL Error Message: SQL Error Message, or SQL Exception, is a vulnerability caused by a Web application inserting user input in a SQL query without validation and failing to suppress error messages that may result from use of such input. This SmartAttack injects SQL characters in order to cause errors in SQL execution, and looks for evidence of such errors.

Set C (Field at a time)
Cross-Site Scripting: Cross-site Scripting vulnerabilities allow malicious scripts to execute in the context of a trusted session with a web site. The SmartAttack alters the inputs to the web application to send benign versions of such malicious scripts, and detects the actual execution or unfiltered reflection of such scripts.

Set D
• HTTP Response Splitting: HTTP Response Splitting is a vulnerability allowing an attacker to structure a request that results in two responses, the second of which is totally under the control of the attacker. The SmartAttack sends benign requests designed to fool the server into returning a response setting a custom Cenzic Cookie where normally it would not and reports vulnerabilities if successful.

• LDAP Exception: This SmartAttack reports each page on which it finds LDAP exception vulnerabilities, that is, where bad input leads to an internal LDAP processing error.

• LDAP Injection: This SmartAttack reports each page on which it finds LDAP Injection vulnerabilities.

• Remote File Inclusion: Remote File Inclusion is a vulnerability where a submitted value is used directly, without sanitization, to reference any specified URL of a file for processing. The SmartAttack submits its own URL to a Cenzic hosted file, as parameter values, and reports each occurrence where it can detect evidence of the Cenzic file being processed.

• Unix Command Injection: This SmartAttack reports each page on which it finds UNIX command injection vulnerabilities.

• Unix Relative Path: This SmartAttack reports each page on which it finds UNIX relative path vulnerabilities.
• Windows Command Injection: This SmartAttack reports each page on which it finds Windows command injection vulnerabilities.

• Windows Relative Path: This SmartAttack reports each page on which it finds Windows relative path vulnerabilities.

Set E

• Application Exception: Application Exceptions are vulnerabilities where unexpected inputs can trigger inappropriate exceptions, or error responses disclosing implementation information, such as a stack trace. The SmartAttack sends various unusual inputs and looks for text in responses evidencing poor error handling.

• Application Path Disclosure: This SmartAttack reports each page on which it finds path disclosure vulnerabilities, that is, where bad input can lead to an internal application error which reveals the path information. This SmartAttack exploits the application business logic to reveal path disclosures.

• Authentication Bypass: This SmartAttack uses SQL injection strings in an attempt to bypass login mechanism.

• Frame Injection: This SmartAttack reports each page on which it finds Frame Injection vulnerabilities.

• Open Redirect: This SmartAttack reports each page on which it finds Open Redirect vulnerabilities.

Set F

• Browse HTTP from HTTPS: Browse HTTP from HTTPS is a vulnerability allowing HTTPS pages to be accessed via HTTP, thus disclosing potentially sensitive information. The SmartAttack attempts to access pages using HTTP, which it observes to be originally accessed via HTTPS. If allowed to access what appears to be the same content, it reports vulnerability.

• Credit Card Disclosure: Credit Card Disclosure vulnerabilities are disclosures of credit card numbers in responses from the Web application. The SmartAttack observes all responses and reports any pages where a full credit card number is observed. The significance of such findings is dependent on the intention and other security factors related to this display.

• Platform Path Disclosure: This SmartAttack reports each page on which it finds path disclosure vulnerabilities, that is, where it finds path information of any components in the original response or in the response for a modified URL. This SmartAttack examines each page and makes additional exploits to cause the platform (for example, ASP.NET, PHP) to reveal path disclosures.

• Revealing Login Message: Revealing Login Message is a type of vulnerability, where an attacker can determine whether a login failed due to a bad username or a bad password, making it easier to identify good usernames and passwords. This SmartAttack runs the traversal three times. In the first it determines if the traversal can login successfully. In the second run it changes the traversal username and keeps the password intact. Then in the third run it changes the password and keeps the username intact and then looks for evidence of different responses with different messages for bad usernames and bad passwords.

• Username or Password in HTTP Request: Username or Password in HTTP Request is a vulnerability reporting inappropriate disclosure of these credentials. The SmartAttack observes all
requests sent to the Web application and reports any requests that carry the username or password in an unsafe manner.

Set G
- Directory Browsing: Directory Browsing is a vulnerability caused by unintentionally disclosing directory listings to users. The SmartAttack attempts to retrieve and identify such listings and reports them as vulnerabilities based on the assumption that the listings are unintended.
- File and Directory Discovery: File and Directory Discovery is a collection of vulnerabilities all related to being able to access resources that should not be accessible. The SmartAttack asks for thousands of known files and backup files and reports anything it can inappropriately access.

Set H
- Cross Site Request Forgery: Cross-site Request Forgery vulnerabilities allow unauthorized requests from a victim’s machine to improperly initiate transactions using an existing authenticated session. The SmartAttack, in combination with utilities used to identify session ids, identifies cases where the application is using only cookies to maintain and communicate session ids, and is therefore vulnerable.

Set I
- Buffer Overflow: This SmartAttack reports each page on which it finds buffer overflow vulnerabilities.
- Cross-Frame Scripting: If a page is allowed to be embedded inside an HTML frame by another page, this may be used by a phishing site to replicate the look and feel of this page. Older browsers even allowed script from the outer page to access content from the page inside the frame. Hence, a page allowing it to be embedded in a frame is a vulnerability, which we call Cross-Frame Scripting. This SmartAttack identifies pages which can be embedded in frames under the control of an attacker.
- Format String: This SmartAttack reports each page on which it finds format string vulnerabilities.
- Integer Overflow: This SmartAttack reports each page on which it finds integer overflow vulnerabilities.

Set J (Inject All Fields)
- Cross Site Request Forgery: Cross-site Request Forgery vulnerabilities allow malicious scripts to execute in the context of a trusted session with a web site. The SmartAttack alters the inputs to the web application to send benign versions of such malicious scripts, and detects the actual execution or unfiltered reflection of such scripts.

Set K
- Check HTTP Methods: A Web application supporting a few potentially dangerous HTTP methods like PUT, DELETE, COPY, MOVE, OPTIONS can aid an attacker to tamper the Web server of the application or craft further advanced attacks. Hence, this is a vulnerability we will call as Check HTTP Methods. This SmartAttack checks the HTTP Methods supported by the server and reports a failure when the web server responds to dangerous HTTP methods.
- Cookie Vulnerabilities: A failure to specify proper attributes for cookies may result into stealing of cookie information through various attacks like Cross-Site Scripting (XSS) or a Man-In-The-Middle attack. Hence, this is a Vulnerability we call Cookie Vulnerabilities. This SmartAttack optionally reports each page where session cookies are set insecurely, persistently, without proper caching directives or without HTTPOnly attribute.
- Web Server Vulnerabilities: Web Server Vulnerabilities are a variety of CVE style vulnerabilities regarding known security flaws in known versions of various software infrastructures, such as
Apache, PHP, Oracle, etc. This SmartAttack does hundreds of tests looking for versions and resources with evidence pointing to known CVEs.

Set L
• Session ID Length: Session ID Length is a vulnerability where session ids are considered too guessable. Session ids with less than 128 bits are not recommended. The SmartAttack observes the character types and length of detected session ids to compute the bit size, and reports those with a bit length less than 128.
• Session ID Randomness: Session IDs with insufficient randomness are vulnerable to brute force session guessing attacks. This SmartAttack performs a “Runs Test” algorithm on observed session IDs and their predictive values (p-values) are calculated. The SmartAttack reports each session ID that does not found to be sufficiently random.

Set M
• Report Broken Links: The Broken Links SmartAttack checks whether all pages visited are valid and do not result in a “Page Not Found” errors. If any page referenced by a link shows such an error message or has a 404 HTTP response status code, the SmartAttack will report the link as a broken link.

Reports
The reports, which are generated into pdf form via the Hailstorm desktop software or web application, will be created when the scan is complete, whether passed or failed, or when requested by a developer or a manager. All reports will be saved in the proper directory on the QA environment server and will also be uploaded to the TeamDynamix ticket which was used to request the scan. Vulnerabilities and remediation tips, as well as much more detail about the scan, will be apparent in the report.
Instructions for Use

Running a Security Scan
Logging into the application

Creating an application

Create New Application
Running application analysis

Cenzic Halstorm Enterprise ARG

File  View  Report  Action  Settings

New
Close
Run Assessment...

Analyze Application...

Save  Ctrl+F5
Delete
Import Application(s)
Export All Applications

Exit  Alt+F4
This analysis lets Haistorm discover information about your web application to optimize subsequent security assessments. It spiders and submits forms for up to 35 minutes, but does not attack or report vulnerabilities.

**URL:** "http://application.com"

**Max Pages:** 100

**Username:**

**Password:** ********

**Confirm Password:** ********

**Max Times to Fetch a Page:** 1

**Page Load Delay (sec):** 30

**Max Unresponsive Host Tries:** 5

**Javscript Links Drops (ms):** 300

**Ignore JavaScript Links:**

**Ignore Form Data:**

**Do Not Load Images:**

**Ignore Queries in URLs:**

**Switch to Interactive Mode When Spider Completes:**

**Fill Form with User Train Data:**

**Automatically Add Related Additional Hosts:**
**Next Steps:**

1. **Review/ Edit the analysis findings.**
   
   Your quality of results can be dependent on these settings. It is suggested, but not required, that you carefully review/edit the settings on the following pages. Note that these settings may be reviewed and edited at anytime using the Application Settings.

2. **Run an Assessment**
   
   Use the "Run Assessment..." link to run an assessment.

---

**Application Analysis: Complete**

---

48
Advanced Spider Settings

These spider settings are used to fine tune the behavior of the spiders. Changing these settings will affect how the spiders crawl your site.

- Perform mouse events for all links
- Site generates AJAX HTTP requests

Application Analysis: Observations/Recommendations

Based on the responses you provided and observations during the analysis, Halicom has generated assessment recommendations for your application.

Halitorm Observes:
- Specific observations:
- No specific IE content was found (10 pages examined).
- Pages observed:
- 10 pages
- Plugins observed:
- 0 plugins

Halitorm Recommends:
- Recommendation: Injectable fields found. We recommend testing with input Validation against:
- Specific settings:
  - Internet Explorer Required fields:
    - Mozilla User-Agent String: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9)
    - Gecko/20090923 Firefox 3.0
    - Mozilla/5.0 [Compressor]
Running pre-defined SmartAttack sets

Tasks

- Run Assessment...
  - View Application Settings...
  - View Reports...
  - Run Interactive Results Assessment...
  - New Spider...
  - New Interactive Recording...
  - New Custom Assessment...

Run Assessment

1. Choose Assessment Type:

   - UC Pre-Test - List All Pages and All Pages with Forms
   - OWASP-2016 (Set L) - Report Broken Links
   - OWASP-2013 (Set A) - SSL/Forms
   - OWASP-2013 (Set A2) - SSL/Forms
   - OWASP-2013 (Set B) - Unvalidated Input SOLIDOM
   - OWASP-2013 (Set C) - XSS inject a field at a time
   - OWASP-2013 (Set D) - Unvalidated Input Injection
   - OWASP-2013 (Set E) - Application/Unhandled Errors
   - OWASP-2013 (Set F) - Disclosure
   - OWASP-2013 (Set G) - Directory Discovery/Browsing
   - OWASP-2012 (Set H) - XSRF
   - OWASP-2013 (Set I) - Overflow
   - OWASP-2013 (Set J) - XSS inject All Fields
   - OWASP-2013 (Set K) - Server Settings and Cookies
   - OWASP-2013 (Set L) - Sessions

Advanced >>

Run | Cancel | Help
Adding an additional host for login page
Viewing the run of the scan
Running the scan on the web client

http://scan.webcentral.uc.edu/3t/
Select a type of assessment to be performed

Assessment Type

Halstorn offers a variety of types of assessments targeting different aspects of security and/or aspects of the target application. Please select a type of assessment that you would like to queue to run.

OWASP-2010 (Set L) - Report Broken Links

Changing the selection will reset fields below

Configure assessment options

Traversal Settings - http://scan.webcentral.uc.edu/3t/ - 150 pages

Options - no recurrence; no constraints; no notification

Advanced Options - any ARC execution engine

Queue Assessment [Uncheck this option if you are not yet ready to queue this assessment for execution. Assessments that are added but not queued will retain the settings specified here and can be easily queued at a later time.]

Schedule Assessment Cancel
## Re-running a Security Scan

### Desktop Application

### Assessment Results

<table>
<thead>
<tr>
<th>Name</th>
<th>Last Run</th>
<th>HARM Score</th>
<th>View</th>
<th>Report</th>
<th>Run</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP-2013 (Set K) - Server Settings and Cookies</td>
<td>4/9/2015 11:35:35 PM</td>
<td>205</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2013 (Set G) - Directory Discovery/Browsing</td>
<td>4/9/2015 11:18:31 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2013 (Set J) - XSS Inject All Fields</td>
<td>4/9/2015 11:32:20 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2013 (Set L) - Sessions</td>
<td>4/9/2015 11:42:20 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2013 (Set G) - XSS inject a field at a time</td>
<td>4/9/2015 10:46:12 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2013 (Set I) - Overflow</td>
<td>4/10/2015 12:23:33 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2013 (Set A) - SSL/Forms</td>
<td>4/9/2015 10:15:32 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2013 (Set B) - Unvalidated Input SQL/DBM</td>
<td>4/9/2015 10:36:06 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2013 (Set D) - Unvalidated Input Injection</td>
<td>4/9/2015 10:46:50 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2013 (Set F) - Disclosure</td>
<td>4/9/2015 11:08:47 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2010 (Set L) - Report Broken Links</td>
<td>4/9/2015 11:49:01 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2013 (Set H) - XSRF</td>
<td>4/9/2015 11:21:07 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2013 (Set A) - SSL/Forms</td>
<td>4/9/2015 10:19:21 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>UC Pre-Test - List All Pages and All Pages with Forms</td>
<td>4/9/2015 8:25:22 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
<tr>
<td>OWASP-2013 (Set E) - Application/Unhandled Errors</td>
<td>4/9/2015 11:00:13 PM</td>
<td>0</td>
<td>View</td>
<td>Report</td>
<td>Run</td>
<td>Edit</td>
</tr>
</tbody>
</table>

### Web Client

### Assessment Status

<table>
<thead>
<tr>
<th>Assessment Type</th>
<th>Priority</th>
<th>Affected (s)</th>
<th>Last Run</th>
<th>Next Run</th>
<th>Recurrence</th>
<th>Status</th>
<th>Page Visited</th>
<th>Edit</th>
<th>Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWASP-2013 (Set D) - Command Injection_Turbo</td>
<td>Medium Base</td>
<td>03/31/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2010 (Set L) - Report Broken Links</td>
<td>Medium Base</td>
<td>04/06/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2013 (Set A) - SSL/Forms</td>
<td>Medium Base</td>
<td>03/30/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2013 (Set A) - SSL/Forms</td>
<td>Medium Base</td>
<td>03/30/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2013 (Set B) - Unvalidated Input SQL/DBM</td>
<td>Medium Base</td>
<td>03/27/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2013 (Set C) - XSS inject a field at a time</td>
<td>Medium Base</td>
<td>03/30/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2013 (Set E) - Application/Unhandled Errors</td>
<td>Medium Base</td>
<td>04/09/2015</td>
<td>Ready</td>
<td>recur</td>
<td>queued</td>
<td>complete</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2013 (Set F) - Disclosure</td>
<td>Medium Base</td>
<td>04/02/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2013 (Set G) - Directory Discovery/Browsing</td>
<td>Medium Base</td>
<td>04/02/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2013 (Set H) - XSRF</td>
<td>Medium Base</td>
<td>04/05/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2013 (Set I) - Overflow</td>
<td>Medium Base</td>
<td>04/05/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>running</td>
<td>complete</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2013 (Set J) - XSS Inject All Fields</td>
<td>Medium Base</td>
<td>04/05/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2013 (Set K) - Server Settings and Cookies</td>
<td>Medium Base</td>
<td>04/06/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>OWASP-2013 (Set L) - Sessions</td>
<td>Medium Base</td>
<td>04/06/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
<tr>
<td>UC Pre-Test - List All Pages and All Pages with Forms</td>
<td>Medium Base</td>
<td>03/30/2015</td>
<td>N/A</td>
<td>no recur</td>
<td>not queued</td>
<td>add to queue</td>
<td>Pages Visited</td>
<td>edit</td>
<td>report</td>
</tr>
</tbody>
</table>

56
Understanding the Results

![Assessment screenshot]

57
Web Server Vulnerabilities enable an attacker to launch more efficient and accurate attacks on the infrastructure of a Web application. The first step towards such attacks is usually reconnaissance using Web Server Vulnerabilities, to which the attacker gains valuable knowledge about the infrastructure and hosting environment. The attacker sends normal requests to a Web server by browsing a few pages of the Web application, but uses an interception proxy such as Foca or WebSeek to observe the requests and responses with their headers. If a response discloses a version number, the attacker may further try to access common files present for that version and then later try to exploit known vulnerabilities in the server.

Exploit

Although this is mainly a helper attack, thus making an exploit unimportant, a Web Server Vulnerability enables an attacker to gain valuable information such as the version number of the server or technology used and access to certain configuration files. This may be considered as an exploit in this context.

For example, if the attacker comes to know that his target Web application is using IBM WebSphere Application Server 6.1.2.1, then a quick search of the BugTraq archives informs him that this server has a Cross-Site Scripting vulnerability in a specific scenario. If he can trick a user of the application into running up in such a scenario, then he may be able to exploit the Cross-Site Scripting vulnerability and steal the victim's session information, etc.

Similarly, if the attacker comes to know that his target application is using the Apache Web server with Tomcat service container having version 6.0.9, then he can quickly find out that this version is vulnerable to a very easily exploitable directory traversal flaw. He can then traverse directories not exposed otherwise and extract sensitive data from the application.

Injections and requests

For each page in the traversal, the SmartAttack creates a modified URL that contains a specially constructed string that can be used to exploit or detect a flaw. The SmartAttack makes a request with the modified URL and analyzes the HTTP response to detect any information disclosures or anticipated responses to a known exploit. Each vulnerability is described in the data file specified in the "Vulnerability Reference File" parameter. This data file contains a list of references and...
Creating a Report

Creating a Report in the desktop application

A solution is described in the vulnerability report made by the SmartAttack. The solution might involve configuring the web server appropriately to eliminate the vulnerability. Other solutions are to upgrade the web application component to a new version in which the vulnerability has been removed, or to apply a patch released by the vendor.
Creating a report on the web application

- Executive Report
- Assessment Report
- Assessment Report w/Remediation
- Assessment Report w/Remediation & HTTP
- Assessment Trends
- Weekly Status Report
- Delta Analysis
- AMEX SecureCode Compliance Report
- BITS Compliance Report
- FISMA Compliance Report
- FTC 16CFR314 Compliance Report
- GLBA Compliance Report
- HIPAA Compliance Report
- MCARD SDP Compliance Report
- NIST Compliance Report
- OWASP-2010 Compliance Report
- OWASP-2013 Compliance Report
- PCI Compliance Report
- PIPEDA Compliance Report
- REGSP Compliance Report
- SB1336 Compliance Report
- SOX Compliance Report
- STIG Compliance Report
- VISA CISP Compliance Report

Assessments

Selected Applications/Assessment Results that will be reported:

- _colmajo 2011 Test
- _colmajo 2013 Test
- _colmajo 2013.02.24 med/ci
- _colmajo 2014.05.26 Pharmacy
- _colmajo 2014.10.10 UCIT App Dev
- _colmajo 2014.10.14 CHI
- _colmajo 2014.10.28 SMTPR
- _colmajo.a 2015.02.05

Generate Report
External Resources
For more documentation about the software:
https://www.trustwave.com/Products/Application-Security/App-Scanner-Family/App-Scanner-Enterprise/
For questions about this document:
Noelle Coleman, Software Applications Developer I, UCIT Software Development
colemaj6@ucmail.uc.edu, (513) 558 – 6048
Summary
Accessibility testing is in place to ensure that the applications produced by the UCIT Software Development department are available to people with disabilities. Compliance with Section 508 of the US Rehabilitation Act is mandatory at the University of Cincinnati and so this process is in place to test all sites and applications for said compliance.

Testing Process
Accessibility testing is required for all applications or websites that are in the production environment at the University of Cincinnati.

Software
In order to run accessibility tests on all of the websites produced at UCIT, the desktop application called SortSite by PowerMapper is used. This software allows the department to scan single pages or entire site, with or without a login required. This software also allows the ability to create a report with versioning by saving in the application or just a word document report by exporting it.

Tests
The tests that SortSite uses are based on Section 508 of the US Rehabilitation Act, W3C WCAG 1 Levels A, AA, AAA, and W3C WCAG 2 Levels A, AA, AAA. SortSite also checks for functional problems such as broken links, missing/corrupt images, SSL problems. As a bonus, SortSite includes a spell checker and some browser compatibility check functionality. However, browser compatibility will be handled in another document.

Reports
SortSite allows the use of reports that are broken down into Summary, Issues, and Pages. These reports can be exported into HTML, Word RTF, or Excel XML documents to be viewed and archived by developers and management.
Instructions for Use
The software is available in the remote QA environment. For access to the environment, refer to the external “Quality Assurance Process Document”

Running a Compatibility Test
SortSite
one-click site testing

Start Here

Quick Start
1. Navigate to the site you want to check using the address bar.
2. Click Check.

Checking Sites with a Login
When you want to check a site that requires a password.

Tip of the Day
You can disable rules using the Options beside each issue in the site report.

Update Watch
You are running version 5.7. This is the most recent version of SortSite.

Top Support Issues
Checking password protected pages

Help and Support
Email Support
Get help from the support team.
Tip of the Day

TOP-RANKED CO-OP & INTERNSHIPS
AND 50 MORE RANKINGS in the TOP 50
(Now that's CINCINNATI SMART.)
Understanding the Results

<table>
<thead>
<tr>
<th>Category</th>
<th>Benchmark against sites</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Quality</td>
<td>104 pages with quality issues</td>
<td>better than average</td>
</tr>
<tr>
<td>Errors</td>
<td>20 pages with broken links or other errors</td>
<td>better than average</td>
</tr>
<tr>
<td>Accessibility</td>
<td>104 pages with accessibility problems</td>
<td>better than average</td>
</tr>
<tr>
<td>Compatibility</td>
<td>68 pages with browser specific issues</td>
<td>better than average</td>
</tr>
<tr>
<td>Search</td>
<td>70 pages with search engine issues</td>
<td>better than average</td>
</tr>
<tr>
<td>Standards</td>
<td>98 pages have W3C standards issues</td>
<td>better than average</td>
</tr>
<tr>
<td>Usability</td>
<td>96 pages with usability issues</td>
<td>better than average</td>
</tr>
<tr>
<td>Totals</td>
<td>249 pages and files checked (1638 pages remaining when 'Stop' clicked)</td>
<td></td>
</tr>
</tbody>
</table>
Summary
Compatibility testing is in place to ensure that all applications produced by UCIT Software Development are viewable and functional in different browser types that people may use. This document is in place to explain the reasoning for testing, the process of compatibility testing, the software that will be used by the department, and the reports that will be a product of testing.

Testing Process
Compatibility testing is in place to ensure that all users will have a positive experience using our websites and applications, so it is mandatory that all sites that will go out into a production environment go through this process.
For more information on requesting compatibility tests, see external “Quality Assurance Process” documentation.

Software
Instead of purchasing an existing software for compatibility testing, the department is using virtual machines with different operating systems on them and with different browsers installed on each one. For now, there are only Windows machines with browsers such as Firefox, Chrome, Internet Explorer, and Safari running on them. In addition to the virtual machine use, the software which is in use for accessibility testing, SortSite, also has a component to check compatibility of websites and applications.

Tests
In order to test the application, the tester will load one of the virtual machines and test on different browsers to make sure everything is displaying correctly and functioning as it should. The SortSite software, mentioned previously and in the external “Accessibility Testing Process” documentation, runs tests that check for browser specific code and settings in the websites that it runs the tests against.

Reports
The reports for browser compatibility are made manually by the QA department when running tests using the virtual machines. The report will contain a section for each browser that the site was tested on and a screenshot of each main page that was tested. There is a format that will be followed in order to fill out the report and it will be updated as browser standards are updated. The format for the report can be found in the external document “Compatibility Testing Report”.
SortSite will also include the browser compatibility tests in its report, but the report will still be included with the accessibility testing documentation.

External Resources
For questions about this document:
Noelle Coleman, Software Applications Developer I, UCIT Software Development
colemaj6@ucmail.uc.edu, (513) 558 – 6048