**Dakota State University**

**Pen Tester**

**TextFileGuys**

**Ethan Shafer, Alex Wollman, Chris Johnson**

Table of Contents

[Summary of Project 2](#_Toc354607697)

[Vision Statement 3](#_Toc354607698)

[Functional Requirements 4](#_Toc354607699)

[Non-functional Requirements: 7](#_Toc354607788)

[Design 8](#_Toc354607789)

[Use Cases 10](#_Toc354607790)

[Misuse Cases 17](#_Toc354607791)

# Summary of Project

The TextFileGuys Pen Tester project takes security analysis scans from different programs such as NMAP, Nessus, and ZAP and organizes them into an online form that is ready to be presented to upper management and customers. This program is made by security researchers with security concepts baked into the whole development process: from design to implementation.

The Pen Tester project is driven by user supplied content: users are required to submit the scans that will compromise the final report. However the Pen Tester project requires a way to verify the scan that the user submits. This is because every program’s scans have different names for different properties, and because we realize uploading directly to servers must be done with care. This upload ability is accomplished through the use of a plugin structure: the user submits a plugin that defines how to verify scans of that type. With this plugin, the Pen Tester project requires minimal refinement and user interaction on how to define the final report layout.

To allow the program to be more feature rich, a user must register with the program. This allows a user to maintain a database of plugins that she can use over and over again. Users can also belong to groups. This allows her to share the plugins she uploaded with other members of the group. In this manner, both big security companies and one man security shops can utilize this tool to their advantage and save themselves valuable time. Finally once all of the individual scans are uploaded and verified, the final report can be generated.

As mentioned earlier this tool was created with security baked in from the design phase. The developers went to great length to utilize secure functions and classes provided by ASP.NET C# libraries. The developers also took extensive care to protect user information from unauthorized data extraction attempts such as; Cross Site Scripting, SQL Injection, Session Hijacking, Cookie Manipulation.

# Vision Statement

The Pen Tester project for Secure Software Engineering was a multipurpose endeavor. As a class project it provided us with an opportunity to work as a group and overcome challenges that face groups; scheduling difficulties, meeting conflicts, and task assignments. These challenges are well known to groups but each group has to handle them in unique ways, custom to their group. Additionally the class project gave us practical experience: programming and tools. Programming experience came from coding for ASP.NET C# MVC. MVC is a different way of programming, and having some experience in that method is invaluable as more and more languages utilize it. Experience gained from tools came from utilizing vulnerability scanning tools such as NMAP and Nessus. Being taught the commands is different from having to utilize them in a real environment.

The Pen Tester project provides all of this in the form of real world experience. A classroom setting teaches the principles and ideas, but applying the ideas to a project reveals the principles true value. Once the true value of the idea is revealed it is easier to recall and use in future projects.

# Functional Requirements

# 1.)Create an account in the program and log in

# 1.1.)User enters registration information

# 1.1.1.)Email address

# 1.1.2.)Password

# 1.1.3.)Confirm password

# 1.1.4.)Submit

# 1.2.)Default group is created for new user in group folder

# 1.2.1.)Folders are created in group folder

# 1.2.1.1.)Reports

# 1.2.1.2.)Plugins

# 1.2.1.3.)Scans

# 1.3.)Confirmation email is sent out

# 1.3.1.)Send Verification code to registered email

# 1.3.2.)Receive verification code to activate account.

# 1.4.)Log-in to the application

# 2.)Create new groups

# 2.1.)Log in to program

# 2.2.)Enter new group name and submit.

# 3.)Utilize a plugin structure to support file formats.

# 3.1.)Create a parser to verify supported and custom Document Type Definition (DTD)

# 3.1.1.)Nmap

# 3.1.1.1.)DTD already exists

# 3.1.1.1.1.)http://nmap.org/book/nmap-dtd.html

# 3.1.2.)Nessus

# 3.1.2.1.)DTD discontinued

# 3.1.3.)Zap

# 3.1.3.1.)Potential DTD

# 3.1.3.1.1.)http://code.google.com/p/zaproxy/source/browse/trunk/src/xml/drivers.dtd

# 3.1.4.)Custom

# 3.1.4.1.)Data organized correctly

# 3.1.4.2.)Attributes listed

# 3.1.4.3.)Entities defined

# 3.2.)Plugin uploaded to the user's selected group plugin folder.

# 3.3.)Specify what attributes relate to what section in the report to be generated.

# 3.3.1.)Required List

# 3.3.1.1.)Report name

# 3.3.1.2.)Host List

# 3.3.1.3.)Host

# 3.3.1.4.)Host IP

# 3.3.1.5.)Open Port List

# 3.3.1.6.)Open Port

# 3.3.1.7.)Port Number

# 

# 4.)Process one or multiple input scan documents and save to group folder

# 4.1.)Accept output from security tools as input into the program

# 4.1.1.)Input will be accepted from

# 4.1.1.1.)Nmap

# 4.1.1.2.)Nessus

# 4.1.1.3.)ZAP

# 4.1.1.4.)Custom formats

# 4.1.2.)Ensure input is well formed and properly formatted

# 4.1.2.1.)Check against the plugin for the uploaded file's type that was selected.

# 4.1.3.)Data not formed correctly will be reported as malformed and rejected

# 4.2.)Upload security tool's scan and name it

# 4.2.1.)Select security scan from computer

# 4.2.2.)Provide a name for it

# 4.2.3.)Select which group to upload it to

# 4.2.4.)Submit it

# 4.3.)Verify it against the correct plugin

# 4.3.1.)Select plugin from list

# 4.3.2.)Submit it

# 5.)Process selected security scan(s) into the output document, view, and delete

# 5.1.)Create Report

# 5.1.1.)Gather Scans into report document

# 5.2.)View scans in user report document

# 5.2.1.)Deseralize files and add to temporary report

# 5.2.2.)Order reports

# 5.2.2.1.)IP address

# 5.2.2.2.)Open ports

# 5.2.2.3.1.)Number

# 5.2.2.3.)Severity

# 5.2.3.)Present scans to user

# 5.3.)Delete Report

# 6.)Manage user names and passwords

# 6.1.)Change password

# 6.1.3.)Enter old password

# 6.1.4.)Enter new password

# 6.1.5.)Confirm new password

# 6.2.)Recover password

# 6.2.1.)Provide email to send link too

# 6.2.2.)Send link to email to continue password recovery process

# 7.) Ability to search data

# 7.1.) Search data with Keywords used

# 7.1.1.) Search by Metadata

# 7.1.2.) Search Entire Sheet

# 7.1.3.) Multi-search

# 7.1.3.1.) Search by String of words

# 7.3.) Search data by Tools used

# 7.3.1.) Nmap

# 7.3.2.) ZAP

# 7.3.3.) Nessus

# 7.3.4.) Custom

# 7.4.) Search data by User submitted

# Non-functional Requirements:

1.) Securely store data

1.2.1.) Safe escaping of special characters

1.2.2.) Avoid SQL syntax

2.) Works on the multiple browsers

2.1.) Chrome

2.2.) Firefox

2.3.) Internet Explorer

3.) Secure session handling

3.1.) Login

3.2.) Page navigation

3.3.) Form submission

4.) Store some data in a database for future use

5.) Store some data in server folders with database references.

# Design

1.)Account creation

1.1.)

1.1.1.)Allow input into a page, escape special characters

1.1.1.1.)EX: !@#$%^&\*(){}[]/=\?+|'",.><

1.1.1.2.)Don't allow duplicate user names

1.2.)C# creates folder for users Plugins/Scans/Reports in their group folder

1.3.)User must verify account through verification email

1.3.1.)Verificaion code is generated and stored in user profile

1.3.2.)Code sent to user through registered email in the form of a link

1.3.3.)User clicks link and verifys account.

2.)Groups

2.2.)Create and manage groups on the server

2.2.1.)No duplicate group names

2.2.2.)Special characters are escaped

2.2.2.1.)EX: !@#$%^&\*(){}[]/=\?+|'",.><

2.2.3.)Groups are saved on server, refered to by unique group number

3.)Plugin

3.1.)Upload file from folder that was selected by user

3.3.)Select values that relate to important fields

3.3.1.)Provide form to user to select required and optional fields needed by the report generator

3.3.1.1.)Verify required fields are present

3.3.1.1.1.)Reject if any required field isn't present.

4.)Process scans into folders

4.1.)POST submitted files to server for handling

4.2.)Values are POSTed through a web form

4.3.)The plugin is attached to the scan and verified.

5.)Process selected security scan(s) into the output document, view, and delete

5.1.)Values are POSTed through a web form

5.1.1.)Queries get required information from scans and added to report

5.2.)scans are selected by user and output is displayed in HTML format

6.)MVC 4/ASP has some built-in user management

6.1.)Create web page for input that contained appropriate fields

6.2.)Use query parameters to pass verification code/email so user doesn't have to enter it.

6.2.1.)Lock account unless verification code is received.

# Use Cases

Test Case ID: 1.1

Description: Selection of a file and uploading it to a database

Module/Class/Function: /FileUploader/Upload

Test Activities

Step 0(Pre): Assume user is logged in, and valid

Populate Known file formats selection

Dropdown of accepted file types populated

Step 1: Select type of Tool used from dropdown

Step 2: Browse for file (File browser)

Step 3: Select file intended for upload

File path shown outside of file browser

Step 4: Click upload button

File passed to validation

Step 5: Entry created in database

===================================================================

Test Case ID: 1.1.2

Description: Validates the file

Module/Class/Function: FileValidation -- NMAP

Test Activities

Step 0(Pre): Assume file exists

Step 1: Get file passed from uploader

Step 2: Verify NMAP Against DTD

User is prompted of success or failure

Step 3: Return to FileUPload

===================================================================

Test Case ID: 2.0

Description: UploadDTD

Module/Class/Function: Plugin/Upload

Test Activities

Step: 0 (pre)

User is logged in as admin and at management panel

Step: 1

Step Description: Enter PLugin name in "Plugin Name" Field

Expected Results: Name entered in field

Step: 2

Step Description: Select Plugin file from file selector

Expected Results: File selected from file system

Step: 3

Step Description: Sumbit Form

Expected Results: Validate paths and name entered into plugin database

Step: 4

Step Description: Save PLugin file in directory

Expected Results: File exists in server directory

===================================================================

Test Case ID: 2.1.1

Description: CheckNmapDTDCorrectness

Module/Class/Function: Plugin/NmapVerify

Test Activities

Step: 0 (pre)

Step Description: NmapDTD has already been uploaded and a generalDTD exists

Expected Results: None

Step: 1

Step Description: Verify a type field exists

Expected Results: Nmap is present in the first line

Step: 2

Step Description: Verify data fields exist

Expected Results: NmapDTD is not an empty document

Step: 3

Step Description: Save NmapDTD

Expected Results: NmapDTD saved to server directory

===================================================================

Test Case ID: 2.1.2

Description: CheckNessusDTDCorrectness

Module/Class/Function: Plugin/NessusVerify

Test Activities

Step: 0 (pre)

Step Description: NessusDTD has already been uploaded and a generalDTD exists

Expected Results: None

Step: 1

Step Description: Verify a type field exists

Expected Results: Nessus is present in the first line

Step: 2

Step Description: Verify data fields exist

Expected Results: NessusDTD is not an empty document

Step: 3

Step Description: Save NessusDTD

Expected Results: NessusDTD saved to server directory

===================================================================

Test Case ID: 2.1.3

Description: CheckZapDTDCorrectness

Module/Class/Function: Plugin/ZapVerify

Test Activities

Step: 0 (pre)

Step Description: ZapDTD has already been uploaded and a generalDTD exists

Expected Results: None

Step: 1

Step Description: Verify a type field exists

Expected Results: Zap is present in the first line

Step: 2

Step Description: Verify data fields exist

Expected Results: ZapDTD is not an empty document

Step: 3

Step Description: Save ZapDTD

Expected Results: ZapDTD saved to server directory

===================================================================

Test Case ID: 2.1.4

Description: CheckCustomDTDCorrectness

Module/Class/Function: Plugin/CustomVerify

Test Activities

Step: 0 (pre)

Step Description: CustomDTD has already been uploaded and a generalDTD exists

Expected Results: None

Step: 1

Step Description: Verify a type field exists

Expected Results: User supplied name is present in the first line

Step: 2

Step Description: Verify data fields exist

Expected Results: CustomDTD is not an empty document

Step: 3

Step Description: Save CustomDTD

Expected Results: CustomDTD saved to server directory

===================================================================

Test Case ID: 3.1.a

Description: Registration Form as a Manager

Module/Class/Function: /Account/Register

Test Activities

Step: 1

Step Description: Enter Email address in form.

Expected Results: None

Step: 2

Step Description: Enter password and password confirmation.

Expected Results: Password confirmation is validated against the first password.

Step: 3

Step Description: Specify that user is a manger.

Expected Results: None

Step: 4

Step Description: Select Submit.

Expected Results: User receives Verification Email, User Account created in database.

===================================================================

Test Case ID: 3.1.b

Description: Registration Form as an Employee

Module/Class/Function: /Account/Register

Test Activities

Step: 0 (pre)

Step Description: User is on Registration page.

Expected Results: None

Step: 1

Step Description: Enter Email address in form.

Expected Results: None

Step: 2

Step Description: Enter password and password confirmation.

Expected Results: Password confirmation is validated against the first password.

Step: 3

Step Description: Specify that user is \*not\* a manager.

Expected Results: Manager Email prompt shows.

Step: 4

Step Description: Enter Manager Email address.

Expected Results: None

Step: 5

Step Description: Select Submit.

Expected Results: User receives Verification Email, User Account created in database, ManageeApproval is created in database.

===================================================================

Test Case ID: 3.2.a

Description: Verify User Account as a Manager

Module/Class/Function: /Account/ConfirmToken

Test Activities

Step: 0 (pre)

Step Description: User creates Account (Test Case 3.1.a)

Expected Results: User Receives Verification Email to Account Email Address

Step: 1

Step Description: Click link in email

Expected Results: User is taken to Confirmation page, User Account is confirmed in database.

===================================================================

Test Case ID: 3.2.b

Description: Verify User Account as an Employee

Module/Class/Function: /Account/ConfirmToken

Test Activities

Step: 0 (pre)

Step Description: User creates Account (Test Case 3.1.b)

Expected Results: User Receives Verification Email to Account Email Address

Step: 1

Step Description: Click link in email

Expected Results: User is taken to Confirmation page.

User Account is confirmed in database.

Manager receives an email for Confirmation as Employee.

===================================================================

Test Case ID: 4.1

Description: Change User password

Module/Class/Function: /Account/Password

Test Activities

Step: 0 (pre)

Step Description: User logs into Account (Test Case //TODO) and is on /Account page.

Expected Results: User has an active session.

Step: 1

Step Description: User enters Old Password into Form

Expected Results: None

Step: 2

Step Description: User enters New Password and Password Confirmation.

Expected Results: Password Confirmation is validated against the first Password.

Step: 3

Step Description: User clicks Submit.

Expected Results: Successful password change is submitted.

Entry is updated in database.

===================================================================

Test Case ID: 4.3

Description: Change User Manager

Module/Class/Function: /Account/Manager

Test Activities

Step: 0 (pre)

Step Description: User has verified and is logged into an Employee account (Test Cases 3.2.b, //TODO)

User is on /Account page.

Expected Results: None

Step: 1

Step Description: User enters new email address into form.

Expected Results: None

Step: 2

Step Description: User clicks Submit.

Expected Results: User receives confirmation message.

Current Manager relation is deleted from database.

User is dropped from Current Manager's Employee list.

New Manager receives an Email.

===================================================================

Test Case ID: 5.3

Description: Search files by tool(s) used

Module/Class/Function: /File/Search

Test Activities

Step 0(Pre): User is logged in, and validated

User on search page

Step 1: Populate types of tools used

Dropdown of available tools populated

Step 2: Select tool to search by

Step 3: Click Search

Returns file entries that used selected tool

# Misuse Cases

Test Case ID: 1.1.1

Description: Upload of a non-known file type

Module/Class/Function: /FileUploader/Upload

Test Activities

Step 1:

Step Description: User enters unknown file format in upload

Expected Results: File inserted into file browser

Step 2:

Step Description: User Selects file format from known format dropdown

Expected Results: Format box is populated

Step 3:

Step Description: User submits invalid report

Expected Results: Report is not submitted, error message is shown that format does not

match selected.

===================================================================

Test Case ID: 1.4

Description: Upload file larger than 30MB

Module/Class/Function: /FileUploader/Upload

Test Activities

Step 1:

Step Description: User enters large file in upload

Expected Results: File inserted into file browser

Step 2:

Step Description: User submits to large of report

Expected Results: Report is not submitted, error message is shown that file is too large.

===================================================================

Test Case ID: 2.0a

Description: User enters an invalid file name

Module/Class/Function: /Plugin/Upload

Test Activities

Step 0: Assume the user is logged in and at the Plugin Management Page

Step 1:

Step Description: User enters a file name containing illegal special characters (‘)

Expected Results: None

Results: None

Step 2:

Step Description: User selects a file to upload

Expected Results: File Browser window pops up and file is selected

Results: File Browser window pops up and file is selected

Step 3:

Step Description: User presses submit.

Expected Results: The file name is properly escaped to avoid database problems

===================================================================

Test Case ID: 2.0b

Description: User uploads an inappropriate DTD file

Module/Class/Function: /Plugin/Upload

Test Activities

Step 0: Assume the user is logged in and at the Plugin Management Page

Step 1:

Step Description: User enters a proper file name

Expected Results: None

Results: None

Step 2:

Step Description: User selects a file to upload that is larger than 30 Mbps.

Expected Results: File Browser window pops up and file is selected

Results: File Browser window pops up and file is selected

Step 3:

Step Description: User presses submit.

Expected Results: The file is rejected and an error is returned to the user.

===================================================================

Test Case ID: 2.1.1

Description: Improper Nmap DTD uploaded

Module/Class/Function: /Plugin/NmapVerify

Test Activities

Step 0: NmapDTD has already been uploaded and a generalDTD exists

Step 1:

Step Description: NmapDTD type field is not present

Expected Results: generalDTD flags an error and bubbles it up

Step 2:

Step Description: Display error to user

Expected Results: Display error to user

Results

===================================================================

Test Case ID: 2.1.2

Description: Improper Nessus DTD uploaded

Module/Class/Function: /Plugin/NessusVerify

Test Activities

Step 0: NessusDTD has already been uploaded and a generalDTD exists

Step 1:

Step Description: NessusDTD type field is not present

Expected Results: generalDTD flags an error and bubbles it up

Step 2:

Step Description: Display error to user

Expected Results: Display error to user

===================================================================

Test Case ID: 2.1.3

Description: Improper Nmap DTD uploaded

Module/Class/Function: /Plugin/ZapVerify

Test Activities

Step 0: ZapDTD has already been uploaded and a generalDTD exists

Step 1:

Step Description: ZapDTD type field is not present

Expected Results: generalDTD flags an error and bubbles it up

Step 2:

Step Description: Display error to user

Expected Results: Display error to user

Results

===================================================================

Test Case ID: 2.1.4

Description: Improper Custom DTD uploaded

Module/Class/Function: /Plugin/CustomVerify

Test Activities

Step 0: CustomDTD has already been uploaded and a generalDTD exists

Step 1:

Step Description: CustomDTD type field is not present

Expected Results: generalDTD flags an error and bubbles it up

Step 2:

Step Description: Display error to user

Expected Results: Display error to user

Results

===================================================================

Test Case ID: 3.1.a

Description: User enters an invalid email Address

Module/Class/Function: /Account/Register

Test Activities

Step 1:

Step Description: User enters an invalid email address in form

Expected Results: none

Step 2:

Step Description: User fills out rest of form correctly and clicks submit.

Expected Results: Form does not submit and email address is flagged as invalid.

User has to re-enter password but no other information.

===================================================================

Test Case ID: 3.1.b

Description: User enters an invalid password (not long/complex enough)

Module/Class/Function: /Account/Register

Test Activities

Step 1:

Step Description: User enters an invalid password in form

Expected Results: none

Step 2:

Step Description: User fills out rest of form correctly and clicks submit.

Expected Results: Form does not submit and password is flagged as weak.

User has to re-enter password but no other information.

===================================================================

Test Case ID: 3.1.c

Description: User enters an incorrect password confirmation

Module/Class/Function: /Account/Register

Test Activities

Step 1:

Step Description: User enters password in first ‘Password’ box.

Expected Results: none

Step 2:

Step Description: User enters different password in ‘Confirm Password’ box.

Expected Results: none

Step 3:

Step Description: User fills out rest of form correctly and clicks submit.

Expected Results: Form does not submit and password is flagged as incorrect.

User has to re-enter password but no other information.

===================================================================

Test Case ID: 3.1.d

Description: User does not fill out registration form completely

Description: /Account/Register

Test Activities

Step 1:

Step Description: User does not fill out a portion of the form.

Expected Results: none

Step 2:

Step Description: User fills out rest of form correctly and clicks submit.

Expected Results: Form does not submit and password is flagged as weak.

User has to re-enter password and any other missed information.

===================================================================

Test Case ID: 3.1.e

Description: User enters an invalid password (not long/complex enough)

Description: /Account/Register

Test Activities

Step 1:

Step Description: User enters an email address for a manager that is not in our system.

Expected Results: none

Step 2:

Step Description: User fills out rest of form correctly and clicks submit.

Expected Results: Form submits, user is created, email is not sent to the entered email address.

===================================================================

Test Case ID: 3.2

Description: User confirms account

Description: /Account/ConfirmToken

Test Activities

Step 1:

Step Description: User changes the URL from the confirmation email.

Expected Results: Page prompts with incorrect verification information.

===================================================================

Test Case ID: 3.2

Description: User confirms account

Description: /Account/Login

Test Activities

Step 1:

Step Description: Enters incorrect login information, clicks submit.

Expected Results: Returned to login page with generic failure message.

Step 2:

===================================================================

Test Case ID: 4.1

Description: User confirms account

Description: /Account/Password

Test Activities

Step 1:

Step Description: User changes the URL from the confirmation email.

Expected Results: Page prompts with incorrect verification information.

===================================================================

Test Case ID: 4.3

Description: User changes manager to invalid manager email address.

Description: /Account/Manager

Test Activities

Step 1:

Step Description: User enters an email address for a manager that is not in our system.

Expected Results: none

Step 2:

Step Description: User clicks submit.

Expected Results: Email is not sent to the incorrect email address.

===================================================================

Test Case ID: 5

Description: SQL injections

Description: /File/Search

Test Activities

Step 1:

Step Description: Attempt to inject SQL statement in the input box.

Expected Results: SQL Injection is aborted. User sees invalid search criteria message.