

Automating Derivative Classification in Multi-Level Secure Documents

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Andrew Amack

May 2014

Major Professor: Jim Alves-Foss, Ph.D.

Authorization to Submit Thesis

This thesis of Andrew Amack, submitted for the degree of Master of Science with a major in Computer Science and titled “Automating Derivative Classification in Multi-Level Secure Documents,” has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor _____ Date _____
 Dr. Jim Alves-Foss

Committee
 Members _____ Date _____
 Dr. Paul Oman

_____ Date _____
 Dr. Daniel Conte de Leon

Department
 Administrator _____ Date _____
 Dr. Gregory Donohoe

Discipline's
 College Dean _____ Date _____
 Dr. Larry A. Stauffer

Final Approval and Acceptance by the College of Graduate Studies

_____ Date _____
 Dr. Jie Chen

Abstract

Information is often tagged with metadata that indicates access guidance and reliability criteria to protect data from unauthorized access or release. In the US government, this often takes form as a security classification. New documents are classified by an original classifier, and subsequent documents derived from the original are classified using a derivative classification process. Derivative classification is a process where a derivative classifier applies a security classification guide, generated by an original classifier, to a new document to apply the correct classification levels to this document. This is often a time consuming and tedious process.

Modern computer systems have the ability to sift through large amounts of data, dynamically generating content based on attributes of the user, including current search terms, history, location and related information. This is all derived information that needs to be correctly classified. In addition, conflicting guidelines can be present in security classification guides, and mistakes can occur in the process. Therefore there is a need for a process to automate the derivative classification process, eliminate errors and simplify the process.

The goal of this thesis is to provide a solution to the derivative classification problem by creating an automated derivative classification process and associated tool. This is demonstrated through development of a mechanism for original classifiers to create a rule file that can be used to automate the derivative classification process and is then incorporated into the UITags project, a broad research project examining different security metadata technologies by inserting metatags in XML documents.

Acknowledgements

I would like to express thanks to my parents for always believing in me and providing the support and encouragement I needed to finish my education. I would also like to thank Robert and Sandy Synstad for providing a place to crash on many nights while travelling between Cheney and Moscow and for providing much needed encouragement and support to my wife and I during the last two years. Finally this would not have been possible without the support of my loving wife who made sure I kept on task while trying to finish her own master degree.

Dedication

To my wonderful wife Valerie

Table of Contents

List of Figures	viii
List of Tables	ix
Chapter 1: Introduction	1
1.1 UITags Project.....	3
1.2 Thesis Impact.....	5
1.3 Thesis Overview	6
Chapter 2: Current Classification Methods.....	7
2.1 History of Classification	7
2.2 Original Classification	7
2.3 Derivative Classification.....	11
2.4 Compartmented Classification	13
Chapter 3: Classification Rule Builder.....	15
3.1 High Level Requirements.....	15
3.2 Proof of Concept Design Decisions	16
3.3 Proof of Concept.....	17
3.4 Proof of Concept Software Architecture and Implementation	21
3.5 Conclusion.....	29
Chapter 4: Derivative Classification Automator	31
4.1 High Level Requirements.....	31
4.2 Proof of Concept Software Architecture and Implementation	33
4.3 Derivative Classifier Class.....	34
4.4 Conclusion.....	39

Chapter 5: Analysis	40
5.1 Verifying Classification Rule Builder.....	40
5.2 Verifying Derivative Classification Automator	43
5.3 Conclusion.....	46
Chapter 6: Conclusion.....	48
6.1 Problem Areas	48
6.2 Future Work.....	49
Bibliography.....	51
Appendix A: Table of Definitions.....	53
Appendix B: Classification Rule Builder Source.....	59
Appendix C: Classification Rule Builder Manual	69
Appendix D: Derivative Classifier Source Code.....	75
Appendix E: Classification Rules Test Source Code	81
Appendix F: Classification Rules Test Xml Files.....	86
Appendix G: Derivative Classification Automator Test XML File	88
Appendix H: Derivative Classification Automator Test Logs.....	92

List of Figures

Figure 1 Security Tagged XML Example	3
Figure 2 UITags Information Flow	4
Figure 3 Example of Multilevel Classification	14
Figure 4 User Interface	17
Figure 5 Classification Rules XML	19
Figure 6 Compartment Selections.....	21
Figure 7 Classification Rules UML	24
Figure 8 Rules UML	25
Figure 9 Rule Builder Form UML	29
Figure 10 Example Entry in Error Log	33
Figure 11 DerivativeClassifier UML	34
Figure 12 Dominance Relationship from <S, {}> to <TS, {}>	37
Figure 13 Dominates Relationship.....	38
Figure 14 All Unit Test Passing.....	43
Figure 15 Test Document Confirming Software Functionality.....	45
Figure 16 Test Document XML.....	47
Figure 17 Program Default Screen.....	70
Figure 18 File Menu.....	70
Figure 19 All Options Enabled	72
Figure 20 Reference File Not Selected Error	72
Figure 21 Classification Text Missing Error.....	73
Figure 22 Clearance Level Drop Down	73
Figure 23 Program Filled Out Correctly	74

List of Tables

Table 1 Security Classification Guide Example	11
Table 2 Glossary of terms (reproduced with permission from [12]).....	54

Chapter 1: Introduction

According to many researchers, users of a system are considered to be the weak link in the security of that system and nothing technically significant has been done to address this problem [1] [2] [3] [4] [5]. Ahmed, et al, wrote, “Human error is far more likely to cause serious information security breaches than technical vulnerabilities”[2]. They also go on to say “...an overwhelming percentage of information security breaches are caused by human factors such as lack of information assurance knowledge, inadequate training, and a general failure to follow security procedures...”[2]. Users are the most unwitting security hole in our systems and processes and computer scientists should do everything we can to prevent the unintentional errors they can bring to the system. This thesis does not specifically address the intentional security problems that users cause, although the technology proposed here will also have an effect on how easy it is for them to cause problems within our system. This thesis will also help address the problem of “Insecure work practices and low security motivation” that Adams and Sasse [6] discuss, because it will help bring user’s attention to security issues that can be caused by not following security guidelines and will help automate these processes which will minimize the ability to make mistakes out of the user’s control. Of course, as Ashenden and Lawrence [7] point out “increasing awareness will not automatically lead to a change in behavior, although it can be a very useful first step.” Automating of the derivative classification process will be a way for users to verify if they are following the correct behavior in how they classify documents.

The problem addressed in this thesis is the process of derivative classification. According to Executive Order 13526 [8] derivative classification is:

The incorporating, paraphrasing, re- stating, or generating in new form information that is already classified, and marking the newly developed material consistent with the classification markings that apply to the source information. Derivative classification includes the classification of information based on classification guidance. The

duplication or reproduction of existing classified information is not derivative classification [9].

The current process of derivative classification is conducted by following a classification guide created by the original classifying authority and is done entirely by hand by the user creating the new document. Because it is an end user marking the new document and attempting to follow a guide written by another human user, there is the possibility of classification errors that can have a negative impact on national security. This thesis will not address the issue of the original classifier making a mistake in classification; that will be left as an exercise for a different thesis. This work is part of a larger data tagging research effort at the University of Idaho, UITags.

The purpose of the University of Idaho tagging project (UITags) is to investigate the use of data tags by security mechanisms in support of system security policies. There are many type of data tags being investigated, from hardware-based tags up through attribute tags on data in a cloud server. Data tags are used during both runtime execution and security decision making, as well as during system implementation and analysis.

There are two main objectives to this thesis work:

Objective 1. To create a process which allows original classifiers to enter classification guidelines for derivative classifiers. This process will allow the original classifier to select the correct classification level, source document, and classification compartment, as described by Kerr [9], and an expiration date of the classification. This process will also double check for overlapping rules and prevent the duplication of classification rules. The proof of concept software for this process will then be incorporated into the UITags project to help with automating the derivative classification process outlined in objective 2.

Objective 2. To create a solution which will use the classification guidelines from Objective 1 to automate the derivative classification process. For the purpose of this thesis each paragraph within the newly created document will be treated as an object and be classified separately. The

```

<?xml version= "1.0"?>
<mission xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation= "mission.xsd"
        label= "C" compartment= "RED">
  <starship label= "C" compartment= "RED">
    <name>Enterprise</name>
  </starship>
  <missionType label= "C" compartment= "RED">
    <nature>Exploration</nature>
  </missionType>
</mission>

```

Figure 1 Security Tagged XML Example

proof of concept for the solution will tag each paragraph following Kerr's security tagging XML process [10]. This will entail embedding XML security tags in the word document that will help automate the delivery of these documents to end-users. The proof of concept software for the solution will then be incorporated into the UITags project.

Figure 1 provides an example of security tagged XML [9]. It is important to note that the tag "label" is the classification level (in this case "Classified") while the tag "compartment" introduces the idea of security compartments, which are used as a way of implementing the "need to know" concept in security, discussed in Chapter 2.

1.1 UITags Project

The UITags project is investigating several operational scenarios where data tagging could be used. For the purposes of this work, consider a web-based information system. A user enters a query into the system, much like a browser search term. The system then automatically generates a report based on the query. The report contains summary information as well as links to information objects (i.e., reports or just portions of reports, data repositories, charts, tables). The user can select a summary, or an object, and get data about the information (i.e., creation and modification dates, original source identification, data corroboration, and security data such as release controls, and security level).

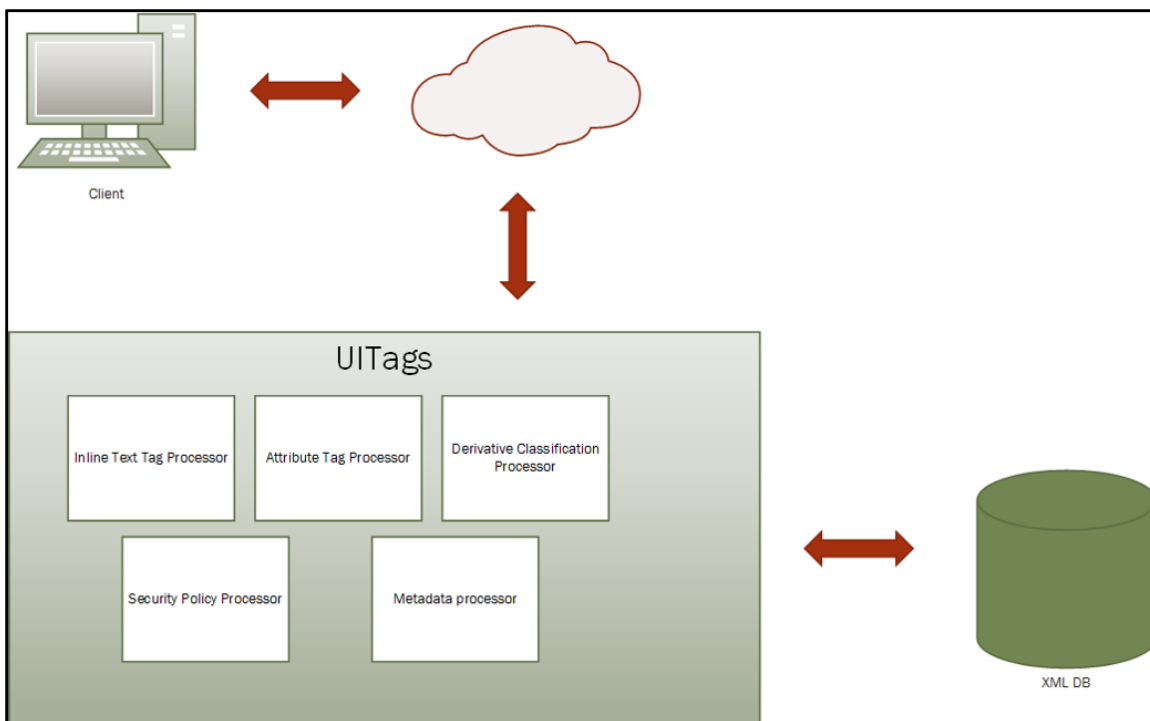


Figure 2 UITags Information Flow

Each source of information is tagged with a set of attributes that includes this metadata. These attributes are used by the system search engine, in conjunction with the user's security credentials, attributes and environmental attributes, to determine accessibility to the data. The system will also need to automatically determine the "derivative classification" of the generated data, because since combinations of data from different sources can result in composite data of a higher security classification. An example of the information flow is shown in Figure 2.

In addition to retrieving data, the user is able to add new data to the system, and to create their own reports by combining information from a variety of sources. The system must be able to propagate the metadata from the various sources, and derive the appropriate security classification of the data so that others can securely access the new data.

An enabling technology for this type of system associates metadata with each information object within a document, such as a MS Word document. The tags should be associated with each informational construct, such as sentences, paragraphs, tables, figures, etc. UITags used

paragraph-level tagging. The technology must support the ability to store a multi-level document, and selectively release parts of it to a user who can edit it. The edited material must then be securely merged back into the stored document.

1.2 Thesis Impact

It is estimated that the United States government spends billions of dollars a year on national security. The cost of running our intelligence agencies is normally classified, however a document leaked by Snowden and published by the Washington Post has shown that it is in the range of \$52.6 billion a year [11]. Intelligence leaks can cause significant damage to our country and to others that support us in our mission at home and abroad. The Taliban was known to have gone over WikiLeaks war logs to find people who had assisted the U.S. [12]. Snowden has proven that one individual with access to national intelligence data can cause reverberations throughout the political landscape of our nation, and that these types of leaks may even cause some of our greatest allies to second guess their commitment to U.S. intelligence efforts. While these issues may be grander in scale than a simple leak by government employee, who improperly derivatively classifies a document, they show how important it is to keep intelligence information in the hands of those who have a need to know, and no one else.

U.S. Executive Order 13526 [8] defines three main classification levels, based on the degree of impact if the information were to be released publicly.

- **Confidential** – should be used when unauthorized disclosure of the information could reasonably be expected to cause damage to national security.
- **Secret** – should be used when unauthorized disclosure could be expected to cause serious damage to national security.
- **Top Secret** – should be used when unauthorized exposure could be expected to cause exceptionally grave damage to national security.

Executive Order 13526 [8] requires the original classifier to identify and describe the damage that could occur if there was a breach of this information. It also requires the classifier to classify documents at a lower level of classification if there is significant doubt about the level of classification being assigned. Derivative classifiers must rely on the information provided by original classifiers and do not have a say as to which classification level can be assigned to documents that reproduce, extract, or summarize classified information.

In 2006 a Government Accountability Office (GAO) report [13] stressed the importance of improving the classification process and reducing the risk of misclassification across the Department of Defense. The automation of derivative classification of documents will have many positive impacts on the U.S. government which include: reducing the work load of derivative classifiers, preventing common mistakes that are usually made when humans manually perform a task, and helping to increase national security by making sure that documents are correctly classified. The process of automated classification could be used by any agency that has derivative classification authority, which is any government employee or contractor who has a clearance and who creates a new document from source materials that are classified at one of the levels listed above, which would also greatly improve accountability of classification decisions as requested by the GAO [13].

1.3 Thesis Overview

The remainder of this thesis is organized as follows: Chapter 2 discusses background on derivative classification and how it is currently implemented. Chapter 3 covers the proof of concept software that helps the original classifier develop classification guidelines and how to use it. Chapter 4 covers the proof of concept software that will help automate the derivative classification process. Chapter 5 contains an analysis of the process and how affectively it automates the derivative classification process. Finally Chapter 6 provides the conclusion and future work.

Chapter 2: Current Classification Methods

In order to fully understand the benefits of automating derivative classification we must first fully understand the history of classification, how original classification currently works and then by extension, how to derivatively classify a document.

2.1 History of Classification

The Atomic Energy Act of 1946 [14] is the precursor act to the issuance of Executive Orders that stipulate how sensitive information is classified and controlled. The Act stated that “It shall be the policy of the Commission to control the dissemination of restricted data in such a manner as to assure the common defense and security.” The goal was to prevent the leakage of information to our enemy the Soviet Union. Executive Order 13526 [8] replaces Executive Order 12356 [15] and is a member of a sequence of executive orders starting with Executive Order 8381 [16] that stipulates the handling and control of classified information within the U.S. Government.

2.2 Original Classification

Original classification is “an initial determination that information requires, in the interest of national security, protection against unauthorized disclosure” [17]. Documents can be originally classified, as outlined in Executive Order 13526 [8], if all of the following conditions are met.

1. An original classification authority is classifying the information
2. The information is owned by, produced by or for, or is under the control of the United States Government
3. The information falls within one or more of the categories of information
 - a. Military plans, weapon systems, or operations
 - b. Foreign government information

- c. Intelligence activities (including covert action), intelligence sources or methods, or cryptology
 - d. Foreign relations or foreign activities of the United States, including confidential sources
 - e. Scientific, technological, or economic matters relating to the national security
 - f. United States Government programs for safeguarding nuclear material or facilities
 - g. Vulnerabilities or capabilities of systems, installations, infrastructures, projects, plans, or protection services relating to the national security
 - h. The development, production, or use of weapons of mass destruction
4. The original classification authority determines that the unauthorized disclosure of the information reasonably could be expected to result in damage to the national security, which includes defense against transnational terrorism, and the original classification authority is able to identify or describe the damage.

It is important to note that “If there is significant doubt about the need to classify information, it shall not be classified” [8]. Against popular belief it is interesting to note that not just any document can be classified and the government does have clear guidelines about the appropriate use of classification authority. Executive Order 13526 also states that you cannot classify a document with the intent of covering up any violations of the law, to prevent embarrassment to agencies or individuals, to restrain competition, or to prevent the release of information [9].

Another important thing to keep in mind is who has original classification authority, this is also outlined in Executive Order 13526 [8] the following people are the only ones authorized to have original classification authority.

1. The President and the Vice President
2. Agency heads and officials designated by the President

3. United States Government officials delegated this authority pursuant to the following guidelines.
 - a. Delegation of original classification authority shall be limited to the minimum required to administer this order. Agency heads are responsible for ensuring that designated subordinate officials have a demonstrable and continuing need to exercise this authority
 - b. “Top Secret” original classification authority may be delegated only by the President, the Vice President, or an agency head or official.
 - c. “Secret” or “Confidential” original classification authority may be delegated only by the President, the Vice President, and an agency head or official provided that the official has been delegated “Top Secret” original classification authority by the agency head.
 - d. Each delegation of original classification authority shall be in writing and the authority shall not be re-delegated except as provided in this order. Each delegation shall identify the official by name or position.
 - e. Delegations of original classification authority shall be reported or made available by name or position to the Director of the Information Security Oversight Office.
4. All original classification authorities must receive training in proper classification and declassification as provided in this order and its implementing directives at least once a calendar year.
5. Exceptional Cases. When an employee, government contractor, licensee, certificate holder, or grantee of an agency who does not have original classification authority originates information believed by that person to require classification, the information shall be protected in a manner consistent with this order and its implementing directives.

It is interesting to note that although once a year training is required for proper classification and declassification, the level of training is not stated, nor does it mention what specific topics users should be trained on. As indicated earlier, human factors introduces the problem of trusting users to follow appropriate guidelines at a one hundred percent compliance level that does not occur normally. Considering the sensitive nature of these documents, if the original classifiers make a mistake in the classification process the derivative classifiers will also make the same mistake, if they are following the security classification guides exactly as written.

Finally one key thing that must be done by original classifiers is the creation of a classification guide “to facilitate the proper and uniform derivative classification of information” [8]. These guides are key to the derivative classification process and rely on derivative classifiers following them to the letter, to properly classify documents. Table 1 is an example of an entry in a security classification guide, which directs the derivative classifier to mark length of deployment as secret, so if the new document states the length of deployment anywhere within the document, the minimum classification of that new document is secret. This guide also indicates other categories that need to be marked at different classification levels and introduces the concept of combining information from two categories which results in elevating information to a higher security level, (e.g. the entry for combination of meeting location and attendees). Because of the importance of the classification guides, there is a need for a program that will allow original classifiers to create these guides electronically. A proof of concept is discussed in greater detail in Chapter 3.

Table 1 Security Classification Guide Example

“Classification String”	U	C	S	TS
Length of Deployment			X	
Exercise Dates	X			
Future Meeting Locations		X		
Future Meeting Attendees		X		
Combination of Meeting Location and Attendees				X
Point of Contact	X			
Name of exercise			X	
Intent to hold planning meetings internally			X	
Combination of Length of Deployment and Dates				X
Combination of Exercise name and dates				X

2.3 Derivative Classification

Derivative classifiers are individuals who generate new documents based on classified information and who must maintain the classification of these new documents based on the classification guidelines created by original classifiers [18]. It is important to note that unlike original classifiers there is “no specific delegation of authority required to be a derivative classifier” [19]. All government employees who possess a security clearance and who work with classified documents are granted derivative classification authority. There are only three authorized sources of derivative classification guidance that can be used to derivatively classify documents:

1. Security Classification Guide, which contains detailed information about why something was classified at its current level and the expiration date of that classification.
2. Existing properly marked source documents.
3. Department of Defense Contract Classification Specification, which is developed for contractors to properly apply derivative classification to a document.

Common mistakes that can be made when derivatively classifying a document include [14]:

1. Trying to classify from memory, e.g. where you recall that the source document was secret without verifying that it was in fact this classification level.
2. Relying on someone else's word about classification level, e.g. asking your co-worker what the classification of a document is because you do not want to look it up or you no longer have access to the document.
3. Making up a classification level because you believe it is the correct level, e.g. the document discusses something you feel should be "Top Secret" but you are not an original classifier and do not have the authority to classify in this manner.

The second proof of concept program for this thesis, discussed in chapter 4, is a derivative classification module that will help alleviate these common errors.

According to the derivative classification training released by the Department of Defense there are three primary ways that derivative classifiers can create new content from classified source material [20]:

1. Extracting: information is copied and stated verbatim in the new document.
2. Paraphrasing: information is reworded in a new or different document.
3. Generating: when information is taken from one authorized source and converted into a different medium for example a word document used to create a video.

Because of the complexities involved in paraphrasing and generating the derivative classification, the derivative classification prototype will only focus on extracting and will leave the intricacies of

paraphrasing and generating for a different thesis. Other areas that will not be covered in this thesis are “Classification by Compilation” where two or more unclassified items may be considered classified if they are combined together, and the concept of “revealed by” where something is revealed that is not explicitly stated. For example in our length of deployment example this would be something along the lines of “we will deploy for 6 months for security training and then deploy for an additional 8 months for guard duty.” These two facts combined reveal the total length of deployment, which is considered to be classified information. The original classifier can indirectly address some of these issues by trying to enter as many details into the rule builder as possible to help catch some of the issues that will come up from classification by compilation.

2.4 Compartmented Classification

Compartmented classification is a concept which adds an additional aspect to the normal classification structure [21]. Compartmented classification helps to introduce the concept of need to know to the current classification guidelines. “Need to know” is a concept in the security world that specifies if you have an equivalent security clearance to view information you still should not view it unless it falls under your “need to know.” Compartmented classification adds a new attribute to security clearances, called compartments. Compartments can be assigned to a document so if two individuals have confidential clearance but they are in different compartments, each individual is only allowed to read their compartments documents and not the other compartments confidential documents. For example there are three compartments, red, green, and blue, then confidential red $\langle C, \{r\} \rangle$ can only view confidential red documents and would be prevented from viewing confidential blue $\langle C, \{b\} \rangle$ documents. It is possible that someone could be cleared to view confidential red blue $\langle C, \{r b\} \rangle$, in which case they could view documents that are labeled as confidential $\langle C, \{\} \rangle$, confidential blue $\langle C, \{b\} \rangle$, confidential red $\langle C, \{r\} \rangle$, or confidential red blue $\langle C, \{r b\} \rangle$. This means a user with

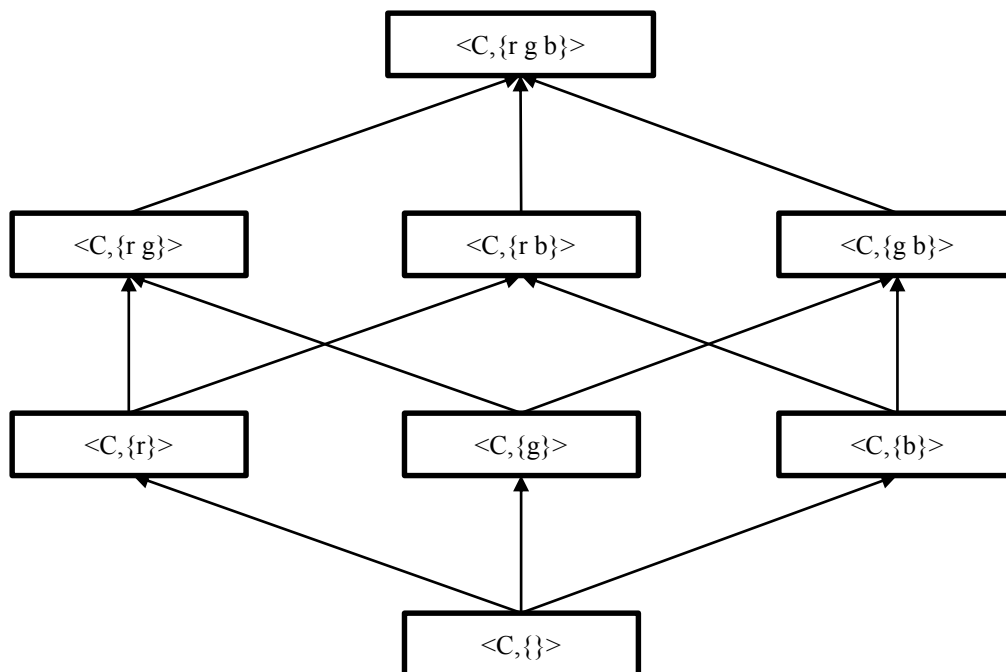


Figure 3 Example of Multilevel Classification

confidential red green black $\langle C, \{r g b\} \rangle$ would have the authority to view all documents labeled as confidential as diagrammed in Figure 3.

Derivative classifiers are also concerned about conflicting information in classification guidelines. If the information is incorrect, or seems incorrect, or if conflicting information exists from official sources like classification guides then the derivative classifier must seek guidance from someone else. The approach outlined in this thesis alleviates this problem by not allowing duplicate conflicting information, which will be discussed further in Chapter 3.

The concepts of derivative classification is clearly defined by the U.S. Government and is a prime candidate for finding a technological solution that will simplify the derivative classification process for all derivative classifiers. Products such as Radiant Mercury from Lockheed Martin [22] show that the Department of Defense is interested in cross domain solutions. A proof of concept solution that will be integrated into the UITags project is discussed in Chapters 3 and 4.

Chapter 3: Classification Rule Builder

In order to properly apply derivative classification rules there has to be a mechanism in place for original classifiers to create a security classification guide. A high level requirement for this prototype is to facilitate the creation of an electronic Security Classification Guide that can be utilized by the software prototype to automate the derivative classification process discussed in Chapter 4. We propose the introduction of software that helps create an electronic classification guide that can be used by automated derivative classifiers. This classification rule builder software is a new way of creating security classification guides that will allow for the automatic derivative classification of new and existing documents within the UITags project. As congressman Hoekstra so elegantly stated, “Just because a person has a security clearance does not give them the authority to exercise leadership in determining what should and should not be classified” and because of this, derivative classifiers will not have access to the Classification Rule Builder since they do not have this authority [23]. While we are not completely removing the human element from the derivative classification process, which as we have seen is far from reliable, the number of original classifiers is very small compared to the number of derivative classifiers. In addition, original classifiers have very strict training requirements where they must demonstrate a full understanding of the original classification process, therefore they should be considerably more reliable [8]. Two definitions that will help with understanding the software requirements are:

1. Classification Text: This is the text that the derivative classification software will look for to classify a new document. Column 1 in Table 1 shows example of classification text.
2. Compartments: These introduce the compartmented classification of our clearance levels. So a rule of $\langle \text{Secret}, \{r\} \rangle$ has classification Secret and compartment of red.

3.1 High Level Requirements

The following high level requirements must be taken into consideration and will make it easier for others to manipulate the prototype to suit their individual needs.

1. Simple User Interface. The user interface must be easy to use while giving the original classifier all of the options necessary to create a classification rule.
2. No conflicting rules. Duplicate rules cannot be allowed for the same reference document, this is to prevent any confusion in the derivative classification process. If a duplicate rule were to exist we may not apply the appropriate classification to the paragraph and we would be reintroducing errors and our goal is to eliminate errors in classification.
3. Code must be easily modified. In order to make sure that other agencies can utilize this software it is important that the source code be easy to understand and simple to modify for other uses. The two key things that can be modified in the source code are the classification levels and the compartments. For example, if NATO wanted to use the software they could add their classification levels of Cosmic Top Secret, Nato Secret, Nato Confidential, Nato Restricted, Atomal, and Nato Unclassified [24].

3.2 Proof of Concept Design Decisions

The following decisions are specific to our proof of concept and will help the proof of concept integrate well into the UITags project.

1. Use an XML flat file. The classification rules builder prototype will use an XML flat file to store the rules and will follow the same attribute tags used in the UITags project. This will allow a custom editor to be created that will follow the classification rules and compartments as outlined in the UITags project so the original classifier modifying the rules will only see rules within their classification and compartment level.

2. Must allow multilevel compartmented classification. The original classifier must have a way to specify the classification of the string entered and a compartment level of the classification. Compartments are an optional requirement since not all classification strings will have a compartment assigned to them.

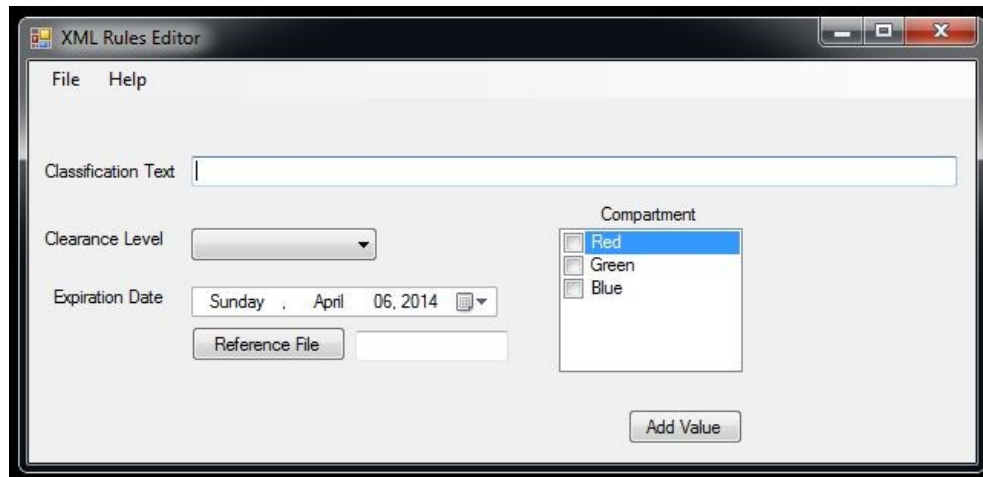


Figure 4 User Interface

3.3 Proof of Concept

The proof of concept is designed to have a simple user interface, prevent conflicting rules, an XML output file that can be integrated into an XML database, allow multilevel classification, and all of this must be integrated into the UITags project. XML was chosen as the output file format so that the rules could eventually be stored in the XML database utilized in the UITags project and allow for editing of rules by users at different classification levels which could be added to the prototype in the future.

3.3.1 Simple User Interface

The user interface (Figure 4) was designed to be minimalistic and easy for the end user to understand. The interface uses elements of a standard windows application and only presents the bare minimum options, so as to not confuse the end user and to prevent clutter in the application menu. The following options under the file and help menu are standard program options and provide the same functionality you would find in a traditional windows application:

1. File -> New: This allows the user to create a new classification rule file. If the original classifier was making a new classification guide for a new document they might want to select this option.
2. File -> Open: This opens an existing classification rule file and is recommended if work is being continued on an existing classification rule set.
3. File -> Close: This will close the current classification rule file that is being worked on after prompting the user to save if the file is not saved.
4. File -> Save: This will present a save dialog to allow the original classifier to save the classification rule file that contains all of the rules.
5. File -> Save As: This will present a save dialog option which allows the original classifier to make a duplicate of an existing classification rule file or to name and save a new classification rule file.
6. File -> Exit: This will exit the program and if the classification rule set has not been saved a prompt will ask the user to save their work.
7. Help -> About: This will show a small window with version information and a brief description of the application.

3.3.2 No Conflicting Rules

Prevention of conflicting rules is of utmost importance for derivative classification to work properly. There cannot be any doubt as to the accuracy of the information and if conflicting rules exist the original classifier will need to do some more in depth analysis to discover why the discrepancy exists and how it should be appropriately addressed. Because of this design decision the application will not allow any rules that have the same classification text and reference file name. If the original classifier attempts to create two entries that contain the same classification text and reference file the user will be informed that the information already exists in the classification rules file and to double check their text entry. The information will not be added to

the classification rule set since it violates the principal of no duplication. If the original classifier feels this is in error they can open the rules file and do some further investigation to fix this problem. Future work will need to be done to add an editor to this prototype that will work within the UITags project but that is out of the scope of this thesis.

If the original classifier adds a new rule that contains the same classification text but a different reference file, the user will not be prompted regarding a conflicting entry since it is possible that two separate documents exist with different classification rules for the same text. It is up to the original classifiers to follow the standard classification rules so they can avoid any potential conflicts in the rule sets.

```
<ClassificationRules Level="SECRET" Compartment="">
  <ComputerName>SFS-WORKSTATION</ComputerName>
  <UserName>Scott Amack</UserName>
  <String>quick style gallery</String>
  <Expiration>2022-04-02T12:03:38</Expiration>
  <CreationDate>2014-04-17T12:04:18.9158349-07:00</CreationDate>
  <FileReference>test.docx</FileReference>
</ClassificationRules>
```

Figure 5 Classification Rules XML

3.3.3 Computer Readable Flat File

As shown in Figure 5 the classification rules XML file is fairly easy to read and understand once you understand how the file is setup. The tags in the XML file contain all of the rule information and are defined as:

1. <ComputerName>: This is the name of the workstation that the user who entered the rule was logged into. This is for auditing purposes.
2. <UserName>: This will record the user name of the user who entered the rule. This is also for auditing purposes.
3. <ClassificationRules>: This tag encloses the entire rule
4. <String>: This tag is the classification text that we will look for in the new document
5. <Level>: This attribute is the security level of the classification text

6. <Compartment>: This attribute lists all of the compartments for the classification text
7. <Expiration>: This is when the rule expires.
8. <CreationDate>: This is the time date stamp for when the rule was created.
9. <FileReference>: This is the reference file for the original classification

The XML file is generated by the C# XmlSerializer that is provided in the standard Microsoft System.Xml.Serialization library. This provides the end user with a XML file generated from our classification rules list that follows the standard rules for XML encoding. The importance of this is twofold and helps us to meet the requirement of generating an easy to read flat file for the end user. First, since it follows the standards outlined in generating a traditional XML document, it allows any developer with XML familiarity to generate an application to take advantage of this file. If the developer uses the ClassificationRules.cs class file that is shown in Appendix B they will be able to serialize or deserialize the XML file into a rule set without any further modification to the code. Secondly the XML format creates a XML formatted flat file that works within the UITags XML schema and will allow future development of a custom rules editor that can serve up the rules to an original classifier and their current clearance level. Three auditing fields in the XML file ComputerName, UserName, and CreationDate are for auditing purposes in case there is a problem with a rule and the user who created the rule needs to be identified. The Computer and User names are pulled from the windows environment variables that are available and will show who the current user is that is logged into the machine and the machine name.

3.3.4 Must Allow Multilevel Classification

In order to work within the specifications of the UITags project, and implement multilevel compartmented classification, this program must of course support multilevel classification [10]. As shown in Figure 2 we must account for the fact that any user can be assigned none, one, two, three, or more compartments within the multilevel classification guidelines as set by the agency

using this software. For our example there are only three compartments and the software provides check boxes to the original classifier so they specify the compartments as needed as shown in Figure 6.



Figure 6 Compartment Selections

3.3.5 Code Must be Easily Modified

Our final requirement is that the code base must be easily modified so that other agencies can customize the software to their individual needs. The two items that will need modifying by individual agencies are the classification levels and the compartments. Classification levels are accessed through a drop down list in the user interface that is populated from the string array called `classificationStrings`. While the typical Department of Defense classification levels are set out in guidelines via Executive Order 13526 there are other agencies like the North Atlantic Treaty Organization (NATO) which use different classification levels [8] [24]. Along with clearance levels, compartments must also be easily modified and these are populated from the string array called `compartmentStrings`. The array `compartmentStrings` generates a checked list box on the form that allows the end user to select from 0 to n compartment elements.

3.4 Proof of Concept Software Architecture and Implementation

The architecture and implementation of this application were created with simplicity in mind for the end users and future developers who might need to modify this project to suit their own individual needs. The code was kept as modular as possible so as to not obfuscate the meaning or intent of the code base. The main attributes and methods in the two main classes in the code base, the `RuleBuilderForm` and the `Classification Rules` class are outlined below. The code for both of

these is detailed further in Appendix B. The UML diagrams in Figure 7, 8, and 9 show the main attributes and operations focused on in the description. Common windows forms attributes like the file menu strip will not be explained in this section because they serve a common purpose are not modified beyond their standard usage.

3.4.1 Classification Rules Class

The Classification Rules class contains the base class that the rule list is generated from and contains all of the information that is needed to properly create a rule. The UML for the ClassificationRules class is shown in Figure 7 and the code can be found in Appendix B.

3.4.1.1 Classification Rules Attributes

The main attributes in the classification rules class are:

1. ClassificationCompartment String: The value of the classification compartment is stored in this string. These values are selected from the compartment check box on the main windows form; these values do not require a selection.
2. ClassificationLevel String: The value of the classification level is secret, top secret, classified, etc. is stored in this string. These values are selected from the drop down list on the main windows form and cannot be empty.
3. ClassificationString String: The value of the classification string is stored here and is limited to a maximum length of 2000 characters, this is a required value and cannot be an empty string.
4. ExpirationDateTime DateTime: The expiration date is stored in this value, the user will only be able to enter a date from the current date to the current date plus 50 years, this is a required value. If the agency using this software requires a longer time they can modify the date requirements in the RuleBuilderForm class.

5. ReferenceFile String: This contains the file name of the original classification document and is a required field. This does not store the path to the original file only the file name to prevent manipulation of the rules that could allow a malicious entry.
6. ComputerName String: This is the name of the computer that the user who created the rule was logged into at the time of rule creation. This is for auditing purposes.
7. UserName String: This is the name of the user that ran the rule builder software, this is for auditing purposes. Because government employees are our main target audience and because they use a common access card to logon to their workstations, we can be reasonably assured that the person logged onto the machine is the user that is currently using the Rule Builder software.
8. CreationDate DateTime: This is the date that the rule was created, this is for auditing purposes.

3.4.1.2 Classification Rules Operations or Methods

The classification rules class only requires three main methods or operations to provide us the functionality we need. These are:

1. ClassificationRules: This is the default constructor for the ClassificationRules class. Because this is a helper class to create our rule set we do not need to initialize any default values here.
2. CompareTo: This method runs a compareTo operation on the incoming ClassificationRule object. This method first compares the classification string and if that is equivalent it then compares the reference file and returns the integer value (a representation of equality, 0 being equal and any non-zero number indicating distance from equality) of the compareTo on those two ReferenceFile strings. If the ClassificationString compareTo value is not equal to zero then this method returns the compareTo value of just the classification strings. For the purpose of how the code functions the other attributes are not important for comparison.

3. Equals: this method returns a Boolean value indicating if two ClassificationRules are equal.

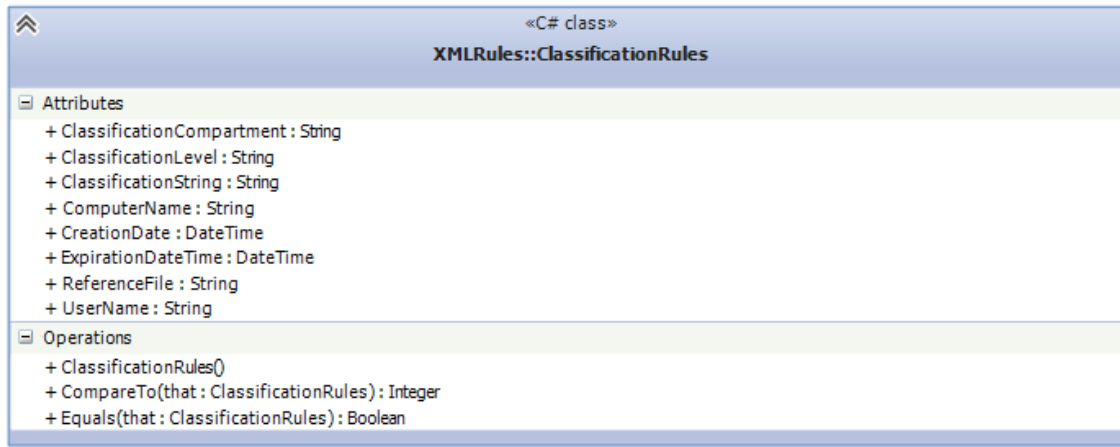


Figure 7 Classification Rules UML

3.4.2 Rules Class

The rules class is nested within the classification rules class, and provides some added functionality including adding new rules to the List and opening and saving the XML file. The UML for this class diagram is shown in Figure 7 and the code can be found in Appendix B.

3.4.2.1 Rules Attributes

The Rules class only contains one attribute the List type of ClassificationRules that holds all of the classification rules for the current xml file.

3.4.2.2 Rules Operations or Methods

The rules class contains 5 methods which are helper methods that make is easy to manage the list of rules. These methods can be modified to meet the needs of the agency utilizing this software. The methods are as follows:

1. Rules: This is the default constructor for the rules class whose only purpose is to initialize the RuleSetList attribute which holds all of the rules.
2. AddNewRule: This method takes the incoming values for the new rule and creates a new ClassificationRules object and then adds it to our list of Classification Rules.

3. InList: This method has a Boolean return type that searches to see if a classification rule is already in our list of rules.
4. OpenXml: This method takes an incoming filename and attempts to open this file into the classification rules list. If the XML file is not in the correct format, the method will generate an InvalidOperationException and the user will be prompted that they need to select the correct XML file. This method can be modified to suit the needs of the different agencies using this software for example instead of opening a .XML file from a specific directory code could be added here to open the file from an XML database such as exist-db.
5. SaveXml: This method takes an incoming filename and uses the XmlSerializer to write the ClassificationRules list out to a properly formatted XML file. This method can also be modified to store the XML file in an exist-db or another format.

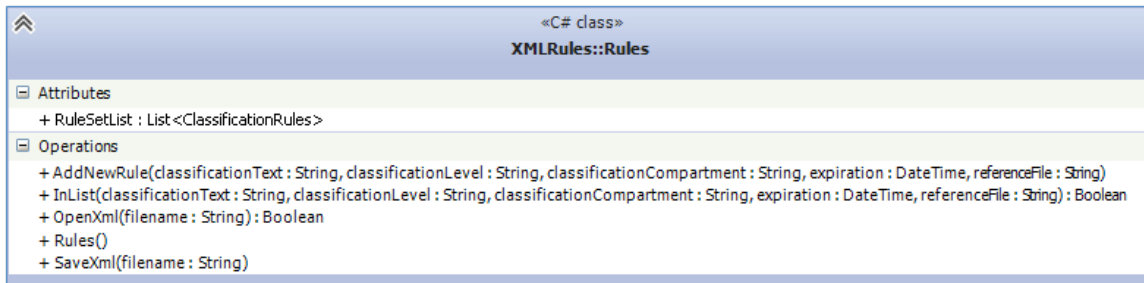


Figure 8 Rules UML

3.4.3 RuleBuilderForm Class

The Rule Builder Form class contains the methods that help control the user interface shown in Figure 3. Some of the default methods that are necessary for windows forms will not be discussed here, as they are not unique to this solution. The UML diagram for this class is shown in Figure 8 and the code for this class can be found in Appendix B.

3.4.3.1 RuleBuilderForm Attributes

The attributes for the RuleBuilderForm class serve two main purposes, first they help keep track of the state of different events like whether the file has been saved and secondly they

provide helpers so we can provide the correct options to the end user when the code builds the windows form. The Attributes are as follows:

1. Saved Boolean: This value is a helper Boolean which indicates if the current RuleSet list has been saved to file or not.
2. classificationStrings string array: This array contains all of the string values for the classification choices that appear in the classification drop down box. This list can be modified to customize this program for other agencies.
3. compartmentString string array: This array contains all of the string values for the classification compartments which appear in the checked list box. These values can also be modified to provide customization for different agencies.
4. filename string: This string contains the full file name of the xml file that we are currently working on and is used when saving the file. If a name does not exist here the user will be prompted to select a file or type in a new file name to be saved with a standard file chooser dialog window.
5. ReferenceFile string: This string contains the value of the Reference file for purposes of building our classification rules. That way when a custom program is used to help automate derivative classification, as discussed in Chapter 4, we can use the reference file list provided by the author to help build a custom rule set. For example if there is a references section in the document we would like to derivatively classify we could build a custom rule set using the names of the files provided in this list.
6. Rule set Rules: This is the Rules object that holds our list of classification rules for the XML file the user is currently working on.

3.4.3.2 RuleBuilderForm Operations or Methods

The operations and methods in the RuleBuilderForm class are mostly helper methods that enable or disable controls, and event handlers that control what happens when a button or menu option is selected. The key operations and methods in the RuleBuilderForm are:

1. RuleBuilderForm: this is the default constructor that builds our form and presents it to the end user. There are a few important things done here that should be noted by future developers. First there are the standard windows form options which set the default behaviors for how the form works and is presented to the end user, these are not covered in detail since they are well documented in the Microsoft developer documentation for windows forms. The program does limit the date options for the date time picker so that you cannot classify a document for longer than 50 years. All of the values of the form are reset so they are not selected and all of the entry boxes are disabled, so users are forced to open an existing file or start a new one.
2. aboutToolStripMenuItem1: This presents the traditional about box to the end user which contains a brief description of the program and version information.
3. addValueButton: This button will add the new value entered by the user once it has been verified that all required fields are filled out correctly, if the required fields are not filled out the user will be prompted to supply the information. Required fields are ReferenceFile, classificationText and classificationLevel. The user will also be prompted if a duplicate entry is attempting to be made and asked to correct the information. Once valid information is presented the new rule will be added to the list and the values of all input boxes will be reset.
4. closeToolStripMenuItem: This tool strip option fires the Close() event that triggers the RuleBuilderForm_FormClosing method to run.

5. `DisableAllBoxes`: This will disable all input on the form and only allow the user to open an existing XML file or create a new one. Once the user has selected a correct file to open or a new file the `EnableAllBoxes` method will be called.
6. `EnableAllBoxes`: This enables all entries on the form.
7. `InitializeComponent`: This is a built in windows form method that initializes all of the controls available to the form.
8. `newToolStripMenuItem`: This allows the end user to start a new form without typing in a file name. The form will have all input boxes enabled and allow the user to create a new XML rule set.
9. `openToolStripMenuItem`: This provides a standard windows file browser to appear for the end user and allows them to select the XML file they wish to open. The file selected will be checked to make sure it is in the correct format before allowing the user to create new rules.
10. `referenceFileButton`: This provides a standard windows file browser which allows the user to select the reference file that they are creating the classification rules for.
11. `ResetValues`: This is a helper method which resets all of the default values of the windows form.
12. `RuleBuilderForm_FormClosing`: This is an internal method provided in windows forms that I have modified to make sure the file has been saved prior to exiting. If the file has not been saved the user is prompted if they wish to save and if they chose to save are given a file browser dialog to type in the name of the XML file they wish to save. The user can also overwrite an existing XML file if they so desire.
13. `saveAsToolStripMenuItem`: This method allows the user to save the XML rule file they are working on into a new file with a different name.

14. `saveToolStripMenuItem`: This method saves the rule file if the filename string contains a value otherwise it will prompt the user to type in a file name using the standard windows save as dialog box.

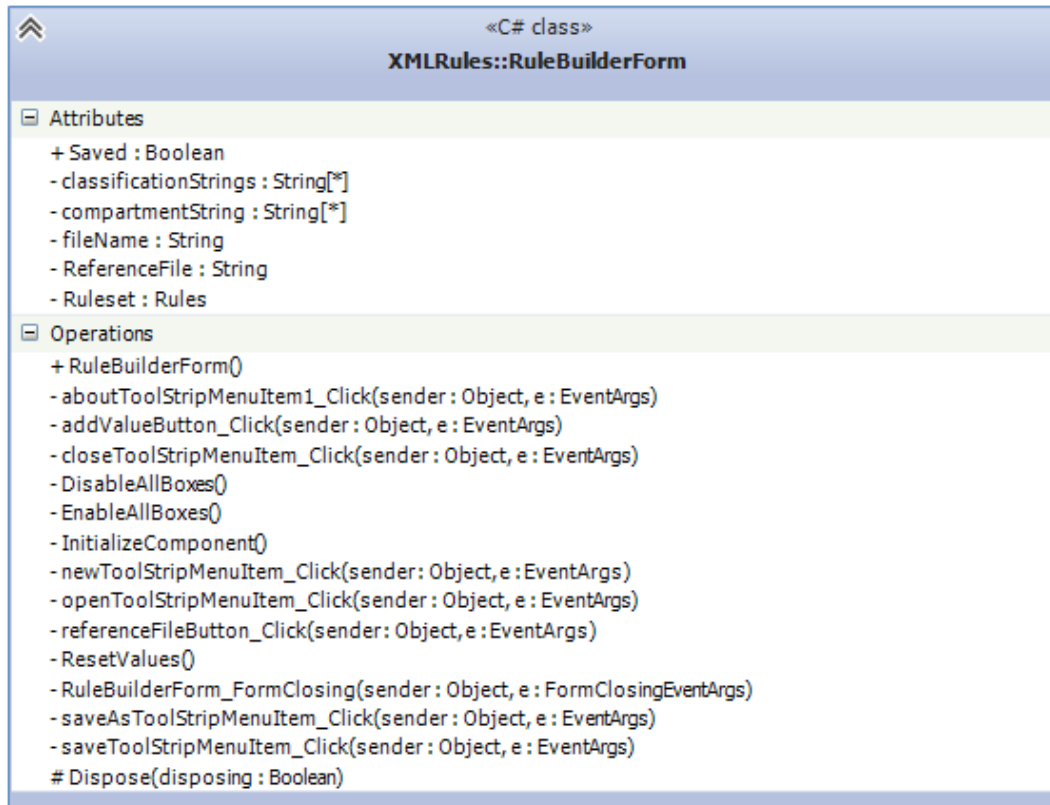


Figure 9 Rule Builder Form UML

3.5 Conclusion

The Rule Builder code was designed to be modular and easily modifiable by any agency that needs to customize it for their specific needs. We have taken advantage of standard windows forms to present the user with an interface that should be intuitive and easy to use while attempting to limit the number of mistakes that could be made in data entry.

It is important to note that in order to provide the most accurate derivative classification it is recommended that all document rules be kept in one XML file to maximize the efficiency of the derivative classification process and for the purpose of this thesis we are not allowing multiple

files to be opened. If multiple files were used the rule set would create additional work for an original classifier because of conflicting rule which would require manual classification of the conflicts, the automatic derivative classification process will be discussed in depth in Chapter 4.

Chapter 4: Derivative Classification Automator

The derivative classification automator is where the rules generated with the classification rule builder software are applied to the word document in the UITags project. This is where the usefulness of the rules created comes to light because without these rules the software would not be able to automatically apply derivative classification to the new document. One very important concept that needs to be discussed is the idea of rule dominance. When a rule is a valid upgrade $\langle S, \{r\} \rangle$ to $\langle TS, \{r b\} \rangle$ then we can say that the rule $\langle TS, \{r b\} \rangle$ dominates the rule $\langle S, \{r\} \rangle$ and is therefore a valid upgrade path. If we had the rule $\langle S, \{r\} \rangle$ to $\langle TS, \{g b\} \rangle$ then we would say that the $\langle TS, \{g b\} \rangle$ rule does not dominate the $\langle S, \{r\} \rangle$ rule because it does not share the same compartments and is therefore not a valid upgrade path. Dominance is how the derivative classification software determines an upgrade and is further discussed in Section 4.3.1.

4.1 High Level Requirements

There are a few key requirements that must be met in order for the derivative classifier software to work properly.

1. Integrated within the UITags project: The derivative classifier software must fit seamlessly within the UITags project and function within the rules of the project.
2. Correctly label classification on new paragraphs: Paragraphs must be properly labeled with their correct classification, either keep the old classification, upgrade the classification to a new one, or mark the paragraph for review because of a classification conflict.
3. Generate an error log: The derivative classifier software must generate an error log so that classification conflicts can be reviewed.
4. Code must be easily modified: The derivative classifier software will need to be modifiable to fit within the needs of the agency utilizing the software.

4.1.1 Integrated With the UITags Project

The derivative classifier was specifically designed for the UITags project and therefore it must fit seamlessly within the code base. The design of the software is compatible with the UITags project, which will be modifying the XML of word documents and applying multilevel classification at the paragraph level. Therefore when new tags are added to the document they are inserted at the paragraph level and if an old tag needs to be replaced the classification rules and dominance of these rules as they related to other classifications in the UITags project is also supported.

4.1.2 Correctly Labeled Classification

The whole point of automating the derivative process is to eliminate the human error element, therefore when the Derivative Classifier software was developed it was very important to make sure all labels are applied correctly. The security lattice [25] shown in Figure 2 shows one example of classification levels with compartments and how they are related to each other. If the classifications are mislabeled this will lead to a spillage of information which would invalidate the use of the derivative classification software within the UITags project.

4.1.3 Error Reporting

Error reporting in the derivative classification software must provide enough information to an original classifier so that they can track down any errors in the derivative classification process. The error reporting file will contain the classification text being classified, the classification of the rule and the paragraph being evaluated, the compartment of the rule and paragraph, the date of the audit attempt, the expiration date of the rule, the file that contains the rules, the source document that the rule was developed from, and the reason why the rule has failed. Figure 10 shows an example entry in the error log that contains all of the information described above. This file location and name can be edited in the constants (Cnst.cs) file in the UITags project and should only be viewed by anyone who has the appropriate authority to see

```
Classification string to be checked is: on the insert tab
Current Node Classification level is: TOPSECRET
Rule classification level is: SECRET
Node compartments are:
Rule compartments are: r g b
Current Date is: 4/19/2014
Rule Expiration Date is: 4/5/2019
File containing XML rules is ..\..\DerivativeClassification\Test.xml
Reference document for this rule is CentOS.docx
Rule Values Did Not Dominate Current Paragraph Values
```

Figure 10 Example Entry in Error Log

this information, an original classifier with the appropriate clearance and compartment (need to know).

4.1.4 Code Must be Easily Modified

Like the classification rule builder software we want this code to be easily modified to fit the needs of the agency who will be utilizing it. The only two items that will need modifying for custom use in the UITags project is the name and location of the rules file we are using to apply derivative classification and the name and location of the error reporting log. The code base also relies on conventions established in the UITags project so if those parts are modified it may require additional modification of the Derivative Classifier code.

4.2 Proof of Concept Software Architecture and Implementation

The architecture and design of the derivative classification software was also created with simplicity in mind. The main purpose of this was to not obfuscate the meaning of the code base because we expect future developers to customize everything based on their specific needs. Keeping this in mind the code was heavily commented to make sure the steps being performed are very clear to the developer viewing the code. The attributes and methods in the DervativeClassifier class file are outlined below. The source code for the derivative classification software is available in Appendix D and the UML diagram is shown in Figure 11. The

`_classificationRules` attribute will not be covered because that is discussed in detail in Chapter 3 and the functionality is the same.

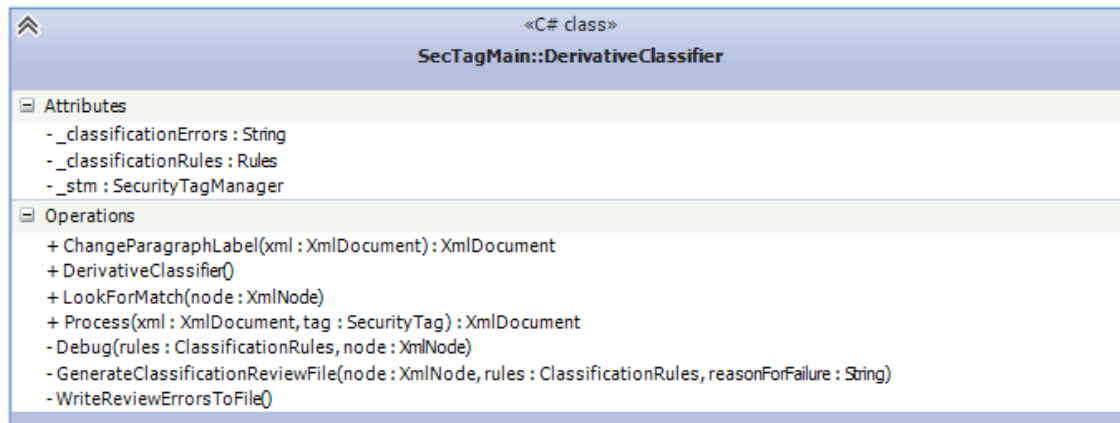


Figure 11 DerivativeClassifier UML

4.3 Derivative Classifier Class

The Derivative Classifier class file uses the `ITaggingModule` interface that is specified in the `UITags` project. The only requirement of this interface is that a `process` method is implemented that accepts the XML document being processed and the security tag of that document being edited, more information about these specific files can be found in the `UITags` documentation.

The main attributes of the Derivative Classifier class are:

1. `_classificationErrors` String: This is the string built to create error log entries.
2. `_classificationRules` Rules: This is the rules class that contains the classifications rules used to derivatively classify the paragraphs.
3. `_stm` SecurityTagManager: This is the tag manager that help us to establish rule dominance.

The operations and methods in the `DerivativeClassifier` class are helper methods to run the derivative classification process and a few more helpers for log building and debug purposes. The key operations and methods in the `DerivativeClassifier` are:

1. **ChangeParagraphLabel:** This is a helper method with runs after the derivative classification software has checked all of the rules for a match within the document. This checks all of the current tags on the paragraph and correctly labels the paragraph with the appropriate classification. This value is obtained from the xml attributes that were changed during the classification process. This is done last because the software does not want to keep modifying the text if a paragraph were to be upgraded in classification multiple times.
2. **DerivativeClassifier:** This is the default constructor for the class and it opens the rules file whose location is stored in the UITags project constants file, it also gets an instance of the security tag manager from the UITags project.
3. **LookForMatch:** This method searches the current XML node in the document and looks for any rule matches. If a match is found the xml attributes are changed to correctly match the new rule if dominance is established. More details about this method are provided below in section 4.3.1.
4. **Process:** This is the main driver method for the derivative classifier software that processes all the steps required to run the derivative classification process.
5. **Debug:** This is a helper method for debugging the code during development and is not used in the production environment.
6. **GenerateClassificationReviewFile:** This helps build the error string using the built in String Builder because it provides a mutable string object that has less overhead on the system. A normal string was not used because with it a string must be destroyed and rebuilt every time it is added to which can be taxing to system resources.
7. **WriteReviewErrorsToFile:** This method writes the error log to disk. The location of the error log is stored in the UITags projects Constants class (Cnst.cs) under Cnst.ErrorFile and should be changed to a secure location.

4.3.1 The LookForMatch method

The LookForMatch method is what does the actual rule lookup and comparison. It is broken down further here since it does the most important work of this class. If any modification needs to be done to customize the derivative classification software this is most likely going to be where the code base is changed. This method takes a correctly tagged XML node that contains the current classification of the paragraph, the current compartments assigned to the classification, and the contents of the paragraph.

In order to search for every rule that may exist in the paragraph the application needs to check the paragraph to see if it contains an exact match for a rule. This is done by iterating through all of the rules and doing a case insensitive regular expression search on the paragraph text. If a match is not found then continue searching through the rest of the rules. This needs to be done for every rule to make sure that each rule is tested against the current paragraph and to appropriately apply derivative classification to the paragraph. For the purpose of this prototype we are only looking at exact rule matches other search types will be deferred to future work and discussed in Chapter 6.

Now that a match is found rule expiration needs to be checked. A check is run against the system clock of the machine and if the rule is still valid then we continue. If the rule is not valid an entry will be made in the errors log indicating that a match was found but the rule was expired. Most classifications have fixed expiration dates but it is possible that this date should be extended, so an original classifier will need to review this information and make sure the rule does not need to be modified [8].

After the derivative classification software has confirmed that the rule is not expired the software checks to see if a previous rule has marked this rule for review. If a previous rule has marked this paragraph for review a message is generated in the error log and the software will continue the rule processing.

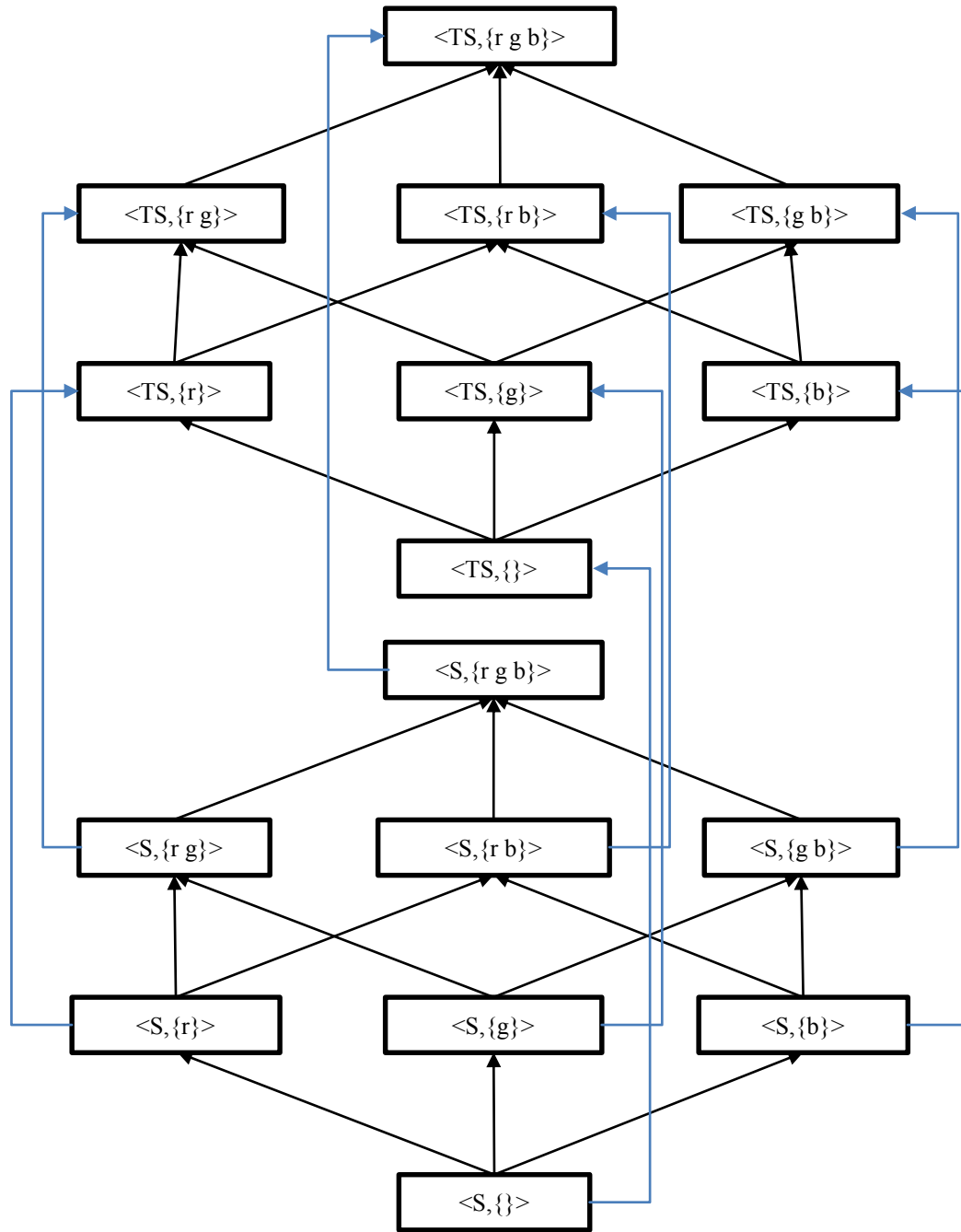


Figure 12 Dominance Relationship from $\langle S, \{\} \rangle$ to $\langle TS, \{\} \rangle$

Now the derivative classification software will use the dominates check in the security tag manager to see if the current paragraph classification and compartments need to be replaced by

the rule classification and compartments. The dominance relationship from $\langle S \rangle$ to $\langle TS \rangle$ is shown in Figure 12 and the arrows indicate the upgrade path.

The dominance relationship illustrated in Figure 12 shows the checks made by the security tag manager dominates function. To follow the figure you can only increase classification if the arrows show a path from the starting classification to the ending target classification. A few rules to illustrate this are:

1. $\langle S, \{\} \rangle$ to $\langle TS, \{\} \rangle$: Evaluating this rule with the chart is easy since we can see a direct connection from $\langle S, \{\} \rangle$ to $\langle TS, \{\} \rangle$, so this rule is a valid rule and $\langle TS, \{\} \rangle$ would be said to dominate $\langle S, \{\} \rangle$ which means a valid upgrade.
2. $\langle S, \{g\} \rangle$ to $\langle TS, \{gb\} \rangle$: This rule is a little trickier to verify since we now how to go through more steps to verify dominance. For this rule we start at $\langle S, \{g\} \rangle$ then connect to $\langle TS, \{g\} \rangle$ and then finally connect to $\langle TS, \{g b\} \rangle$, so this is also a valid upgrade.
3. $\langle S, \{g\} \rangle$ to $\langle TS, \{rb\} \rangle$: This rule is not a valid upgrade because $\langle S, \{g\} \rangle$ is not dominated by $\langle TS, \{rb\} \rangle$. We can see this if we follow the chart. Thusly: $\langle S, \{g\} \rangle$ connects to $\langle TS, \{g\} \rangle$ but we do not have a connection from $\langle TS, \{g\} \rangle$ to $\langle TS, \{rb\} \rangle$ because $\langle TS, \{g\} \rangle$ is not dominated by $\langle TS, \{rb\} \rangle$. That is $\langle TS, \{rb\} \rangle$ will dominate ($\langle S, \{\} \rangle$, $\langle S, \{r\} \rangle$, $\langle S, \{b\} \rangle$, $\langle TS, \{\} \rangle$, $\langle TS, \{r\} \rangle$, $\langle TS, \{b\} \rangle$).

$$\langle l_1, c_1 \rangle \text{ is dominated by } \langle l_2, c_2 \rangle \text{ iff } l_1 \leq l_2 \text{ and } c_1 \subseteq c_2$$

Figure 13 Dominates Relationship

Dominance of a rule is determined by the formula shown in Figure 13 [21]. If the derivative classifier software determines that the rule in the rule list dominates the node then the existing node attributes for classification level and compartments will be changed to the levels of the rule. If the rule does not dominate the existing rule in the node an entry is made in the log saying dominance was not established and why.

4.4 Conclusion

The combination of the Classification Rules builder software and the Derivative Classification software are the contribution of the thesis to the problem of automating the derivative classification process in the UITags projects. When the correct rule set is created and the UITags project is utilized appropriate multi-level classification will be applied to any word document processed by the UITags software.

Chapter 5: Analysis

There are two approaches to verifying the functionality of all of the software modules in this thesis. Unit tests were used for the ClassificationRules, Rules, and RuleBuilder class in the Classification Rule Builder software. The Derivative Classification software evaluation started out with a document with $\langle U, \{\} \rangle$ and changed the classification levels to check for dominance. The rule that all other rules are a dominated by for testing purposes is $\langle TS, \{r\ g\ b\} \rangle$. We then checked all of the rules and made sure that the upgrade was performed correctly and that errors are being properly reported. Each method will be discussed in detail in this chapter.

5.1 Verifying Classification Rule Builder

In order to confirm that the Classification Rule Builder software is working as intended unit tests were written for each method to verify functionality. The source code for the unit tests is located in Appendix E and the XML files that were used are in Appendix F. Setting the value of Attributes are all tested but getting the values are not because in order to test their values we have to be able to get the values first so if getting the value was not working the test would never pass. The way visual studio tests decide if a test works is through assert statements that compare set values with expected values, so the code line “Assert.IsTrue(rules.Equals(rulesTwo));” verifies that a Boolean value of true is returned for rules.Equals(rulesTwo)). As long as it is true a green light is shown as illustrated in Figure 14. The tests performed are:

1. Verify adding of rules: This test gets the rule set and then adds a new rule and confirms that the size of the list has increased by 1.
2. Verify bad XML files will not open: This test attempts to open an XML file that is not properly formatted and verifies that an error is generated and the file is not opened. The Microsoft deserializer is passed a parameterized type when the XML file is opened so we are guaranteed that any XML file not matching this parameterized type will throw an exception and fail the test.

3. Verify classification compartment set: This test creates a new classification rule in the list, changes the classification compartment, and then confirms that the change was applied.
4. Verify classification level set: This test creates a new classification rule, sets the classification level, and then confirms that the level was set correctly.
5. Verify classification string set: This test creates a new classification string, sets the classification string, and then confirms the string was set correctly.
6. Verify compareto equal: This test creates two separate identical classification rules and then runs a comparison on the rules to verify that 0 is returned, because 0 represents equality in a compareTo method (which is a standard defined in other languages as well including C and Java).
7. Verify compareto not equal: This test creates two separate classification rules that are not identical and verifies the compareTo method is returning a non-zero value.
8. Verify rule set size when newly created: This test creates a new rule set and confirms it is empty on creation.
9. Verify Equals when equal: This test creates two rules that are identical and tests to make sure the equals method correctly returns a true value.
10. Verify equals when not equal: This test creates two rules that are not identical and tests to make sure the equals method correctly returns a false value.
11. Verify expiration date set: This test creates a DateTime object with a verification date set in the future and confirms that this date is set correctly.
12. Verify Rule not in list: This test creates a test rule and verifies that the inlist method correctly states the rule is not in the rule set.
13. Verify rule in list: This test creates a new rule and adds it to the list and then verifies that the inlist method correctly reports that the rule is in the rule set list.

14. Verify XML rules file is opened: This test creates a new rules list object and then deserializes the rule XML file into the list, it then checks the list to make sure all the rules were added.
15. Verify Contents of XML rules file: This is a series of tests that creates a new rules list object and then deserializes the rule XML file into the list, after this we check all of the values in the XML file based on what the test file contained and confirm that they are correct.
16. Verify reference file name is set: This test creates a new rule, sets the reference file name of the rule and then confirms the name is set correctly.
17. Verify XML file is saved: This test confirms that a save file is being created by deleting the existing save file in the test environment, creating a new save file, and then confirming that the file was saved to the disk.

All of these tests confirm the functionality of the Classification Rules Builder software and can be modified in the future if the file is customized for different agency use. Once the file is modified the test will produce a red light and the code should be tested until all green lights appear like in Figure 14.

Three things were not tested with this class because they do not have public accessibility. UserName, ComputerName, and CreationDate which are not publicly accessible because the values stored in them are windows environment variables that are unique to each machine the tests are run on. We tested that the values were being correctly set by generating output to the console in a separate piece of c# code so we could verify that the values shown were the same values being saved in the rules file that are serialized out to disk.

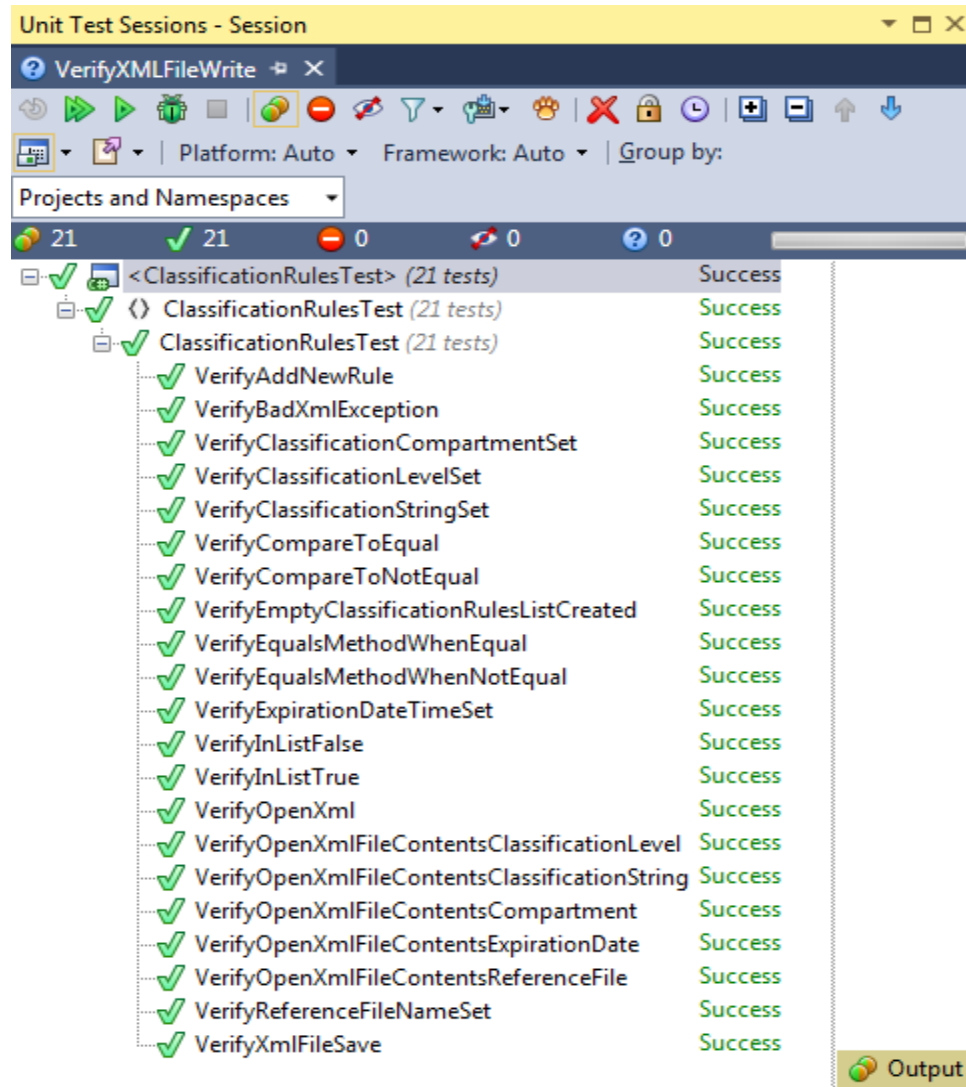


Figure 14 All Unit Test Passing

5.2 Verifying Derivative Classification Automator

In order to verify the functionality of the derivative classification automator software custom rules files were built to test all possible classification decisions. The rule files contained identical rules which elevated the classification of the paragraph from $\langle C, \{\} \rangle$ all the way to $\langle TS, \{r\ g\ b\} \rangle$ as shown in Figure 12 and following the dominance relationship shown in Figure 13. Rule files were also created to test downgrading from $\langle TS, \{rgb\} \rangle$ to $\langle C, \{\} \rangle$ to confirm that

rules were not being incorrectly downgraded in classification and compartment. The rule set used for testing is found in Appendix G and an upgrade log function was added to the derivative classification code base to confirm upgrades were being handled correctly. Logs for upgrades and error logs for these tests are also found in Appendix G. The base word document used for testing purposes has all classification levels and compartments set to <UNCLASS, {}> so every first upgrade will be valid. Classification levels and compartments were then changed to a specific case to be tested to make sure that errors were being recorded correctly in the error file or that the rule is upgraded correctly. The word document and underlying XML were also examined to verify that the security label was correctly updated in the document.

The tests in the XML file that were verified to be working correctly in the test log file are:

1. Upgrade from <UNCLASS, {}> to <UNCLASS, {}>
2. Upgrade from <UNCLASS, {}> to <SECRET, {}>
3. Upgrade from <SECRET, {}> to <TOPSECRET, {}>
4. Upgrade from <TOPSECRET, {}> to <TOPSECRET, {b}>
5. Upgrade from <TOPSECRET, {b}> to <TOPSECRET, {g b}>
6. Upgrade from <TOPSECRET, {g b}> to <TOPSECRET, {r g b}>

The tests in the XML that were verified to correctly produce errors are:

1. Downgrade from <TOPSECRET, {r g b}> to <SECRET, {r g b}>
2. Downgrade from <SECRET, {r g b}> to <UNCLASS, {r g b}>
3. Downgrade from <UNCLASS, {r g b}> to <UNCLASS, {r b}>
4. Downgrade from <UNCLASS, {r g b}> to <UNCLASS, {b}>
5. Downgrade from <UNCLASS, {r g b}> to <UNCLASS, {}>
6. Change compartments incorrectly from <UNCLASS, {b}> to <UNCLASS, {r}>
7. Change compartments in upgrade from <UNCLASS, {b}> to <SECRET, {r}>
8. Change compartments in upgrade from <UNCLASS, {b}> to <TOPSECRET, {g}>

9. Change rule check with an expired rule

As shown in the test results in Appendix H these tests generate the correct error messages and correct log entries when the rule was successfully applied. This confirms that the rules are getting applied to the nodes while the Derivative Classification software is running. Next I investigate the word document that was created by the UITags software to confirm that the tags are properly inserted into the document at the start of each paragraph. The test document shows that tags were correctly added as shown in Figure 15.

[UNCLASS] On the Insert tab, the galleries include items that are designed to coordinate with the overall look of your document.

[TOPSECRET] You can use these galleries to insert tables, headers, footers, lists, cover pages, and other document building blocks.

[UNCLASS] When you create pictures, charts, or diagrams, they also coordinate with your current document look.

[UNCLASS] You can easily change the formatting of selected text in the document text by choosing a look for the selected text from the Quick Styles gallery on the Home tab.

[UNCLASS] You can also format text directly by using the other controls on the Home tab.

[UNCLASS] Most controls offer a choice of using the look from the current theme or using a format that you specify directly.

[UNCLASS] To change the overall look of your document, choose new Theme elements on the Page Layout tab.

[NEEDS REVIEW] To change the looks available in the Quick Style gallery, use the Change Current Quick Style Set command.

[UNCLASS] Both the Themes gallery and the Quick Styles gallery provide reset commands so that you can always restore the look of your document to the original contained in your current template.

[UNCLASS] On the Insert tab, the galleries include items that are designed to coordinate with the overall look of your document.

Figure 15 Test Document Confirming Software Functionality

Figure 15 also shows the TOPSECRET label and NEEDS REVIEW labels applied correctly to the paragraphs. TOPSECRET label is applied to the second paragraph and if we review the error logs we will see an entry for the attempts to downgrade the classification on that paragraph incorrectly. The NEEDS REVIEW label is applied to the eighth paragraph because of an attempt to classify this paragraph from <UNCLASS,{b}> to <UNCLASS,{r}>. The reason the paragraph is being tagged for review is for an original classifier to manually investigate the compartments being applied and make a determination if the current classification (<UNCLASS,{b}>) should be kept or possibly even upgraded to a new classification of

<UNCLASS,{r b}>, this may even indicate a need to investigate the rules file and modify the rule to the new determination.

Finally the last thing that needs to be checked is the contents of the XML within the word document itself. This check will confirm that the tags are correctly applied and saved outside of the derivative classification software. Since Word documents are a compressed XML document with special Microsoft formatting applied, you can view the XML in the document by renaming it from a *.docx to a *.zip file and then extracting the new *.zip file. This allows you to view the underlying XML. After the .zip file is extracted you can view the document's XML contents in the word folder and then the document.xml file. As you can see from the XML in Figure 16 the label tag, for classification level and the compartments tag are correctly marked with the TOPSECRET and r g b compartments. It should be noted that the tags will not stay if you open the document using Word and then edit the document. This is because Microsoft Word does not allow custom XML tags to be inserted into their documents, the solution to this problem is being addressed by the UITags project [26].

5.3 Conclusion

Extensive testing was done on both the Classification Rule Builder and the Derivative Classification Automator software. The tests show that both software projects are fully working as intended and these tests can be used to further test functionality of both projects if any modifications are made in the future. Unit tests from the Classification Rule Builder will not work correctly if modifications are made to the functionality of the software and will be helpful for future developers for troubleshooting any issues in the code base. Automated unit tests are not used in the Derivative Classification Automator software because of needing to verify XML tags were being correctly added to documents and this can only be done manually because Microsoft does not allow modification to the XML files of word documents and it could not be guaranteed

that opening the files programmatically via a unit test would not cause the Microsoft office software development kits to strip out the tags added by the UITags project.

```
<w:p w:rsidR="006246C2" w:rsidRDefault="006246C2" w:rsidP="006246C2" stox:label="TOPSECRET"
stox:compartments="r g b " stox:preserve="PRESENT">
  <w:r>
    <w:t xml:space="preserve">[TOPSECRET] You can use these galleries </w:t>
  </w:r>
  <w:bookmarkStart w:id="0" w:name="OLE_LINK1"/>
  <w:bookmarkStart w:id="1" w:name="OLE_LINK2"/>
  <w:r>
    <w:t>to insert tables, headers, footers</w:t>
  </w:r>
  <w:bookmarkEnd w:id="0"/>
  <w:bookmarkEnd w:id="1"/>
  <w:r>
    <w:t>, lists, cover pages, and other document building blocks.</w:t>
  </w:r>
</w:p>
```

Figure 16 Test Document XML

Chapter 6: Conclusion

This thesis presents a solution to the problem of derivative classifiers, the human element, introducing errors in the derivative classification process. The original objectives of this thesis were to create a program to allow original classifiers to enter classification rules and to create a program to automate the derivative classification process. A program was created to generate rules that would follow the XML requirements of the UITags project and this was confirmed by running unit tests on the classification rules generated by the Classification Rule Builder software. The second objective was also met by creating the Derivative Classification Automator Software and incorporating that into the UITags code base, this was confirmed to function correctly by verifying the XML tags within the newly created word document and by verifying the upgrades and errors in the respective error and upgrade logs that were produced during testing.

6.1 Problem Areas

The following subsections describe a few potential problem areas that will need to be addressed when the software is released into a production environment.

6.1.1 Classification Rule Builder Potential Problems

The Classification Rule Builder software will need a login mechanism to prevent unauthorized access to rule sets. The software will also need a rule checker built in to help facilitate the rule checking process. Currently the rules have to be manually edited which would also require editing the XML in the Word document.

6.1.2 Derivative Classification Automator Potential Problems

The Derivative Classification Automator software algorithm may need to be modified for very large documents. Currently there are two nested for loops which did not appear to have

degraded performance during the code tests but this may prove to be problematic if documents of an exceptionally large size are derivatively classified.

6.2 Future Work

The two software programs created by this thesis meet the requirements as outlined in the objectives in Chapter 1, but there are a few things that could be done to increase the functionality to improve the derivative classification process.

6.2.1 Classification Rule Builder Future Work

The Classification Rule Builder software could benefit from a few modifications that would help improve the derivative classification process, the future modifications could add an extra layer of checking. One of the things that would improve the Classification Rule Builder software is generating new rules based on synonyms this would help address the problem of paraphrasing discussed in Chapter 2 where information is reworded. Another improvement that could be made is adding spell checking to catch if the original classifier misspelled a word when entering it into the software, new rules could also be generated based on commonly misspelled words so that those are caught in the new documents. Another improvement would be to incorporate an editor into the Classification Rule Builder software which would provide a means for the original classifiers to view the rule set without having to open the rules file and manually edit that. A layer of security should also be added so that only original classifiers can run the software and an encryption mechanism could be used so that only the Classification Rule Builder software could read the XML file.

6.2.2 Derivative Classification Automator Future Work

The Derivative Classification Automator software could be improved upon to provide additional functionality to the derivative classification process. First there needs to be a solution created to solve the “Classification by Compilation” problem where two or more items are combined to create a new classification level which then elevate the security of the document.

Another issue with “Classification by Compilation” is if two items exist in separate paragraphs that should be elevated, for example exercise name and dates as shown in Table 1, matching for this rule only exists at the paragraph level not the document level so the classification elevation will be missed. The problem of “revealed by” discussed in chapter 2 is also important to address and may involve modification to both the Derivative Classification Automator and the Classification Rule Builder software. Error reporting should be incorporated in the data store utilized by the UITags project so the error logs could also be generated for users and their classification level. The time check for expiration may want to be upgraded to use an outside trusted time source for expiration checks so a malicious actor does not change the clock on their computer and fool the system. Security considerations would also add protection to the log files created by the Derivative Classification Automator so information spillage does not occur from an unauthorized user viewing those files. The tagging should be implemented at the sentence level rather than the paragraph level, this would provide a higher level of granularity and may even reduce the number of multiple “needs review” tags from occurring.

6.2.3 Other Future Work

A program should be created that would provide an interface for original classifiers to view error logs and make corrections to classification errors without having to manually edit the rules files. This would eliminate the need to search through the log files, to find why the error occurred, and prevent mistakes in editing of the files.

Bibliography

- [1] S. Vidyaraman, M. Chandrasekaran, and S. Upadhyaya, "Position: The user is the enemy," in *Proceedings of the 2007 Workshop on New Security Paradigms*, 2008, pp. 75–80.
- [2] M. Ahmed, L. Sharif, M. Kabir, and M. Al-Maimani, "Human errors in Information Security," *Int. J. Adv. Trends Comput. Sci.*, Vol. 1, No. 3, pp. 82–87, 2012.
- [3] M. Whitman, "Enemy at the gate: threats to information security," *Commun. ACM*, Vol. 46, No. 8, pp. 91–95, 2003.
- [4] M. E. Whitman and H. J. Mattord, "The enemy is still at the gates: threats to information security revisited," *2010 Information Security Curriculum Development Conference*. ACM, pp. 95–96, 2010.
- [5] C. Herley, "So long, and no thanks for the externalities: the rational rejection of security advice by users," *Security*, pp. 133–144, 2009.
- [6] A. Adams and M. Sasse, "Users are not the enemy," *Commun. ACM*, Vol. 42, No. 12, 1999.
- [7] D. Ashenden and D. Lawrence, "Can we sell security like soap?: a new approach to behaviour change," *Work. New Secur. Paradig.*, pp. 87–94, 2013.
- [8] E. Order, "13526," *Classif. Natl. Secur. Inf.*, Vol. 29, 2009.
- [9] K. R. Kosar, *Classified information policy and executive order 13526*. DIANE Publishing, 2011.
- [10] L. Kerr, *Polyinstantiation in Multilevel Secure XML Databases*, University of Idaho, 2012.
- [11] B. B. Gellman and G. Miller, "U.S. spy network's successes, failures and objectives detailed in 'black budget' summary," *The Washington Post*, 2013.
- [12] D. V Gioe, "Tinker, Tailor, Leaker, Spy.," *Natl. Interes.*, No. 129, pp. 51–59, 2014.
- [13] D. M. D'Agostino, A. Borseth, M. Fenton, A. Hatton, B. Hills, D. Keefer, D. Mayfield, J. Reid, T. Richardson, and M. Schwartz, *Managing Sensitive Information: DOD Can More Effectively Reduce the Risk of Classification Errors*, 2006.

- [14] “Atomic Energy Act of 1946,” 1946. [Online]. Available: http://science.energy.gov/~media/bes/pdf/atomic_energy_act_of_1946.pdf. [Accessed: 28-Apr-2014].
- [15] E. Order, “12356,” *Natl. Secur. Inf.*, Vol. 6, 1982.
- [16] J. K. Elsea, “The protection of classified information: The legal framework,” 2006.
- [17] “Derivative Classification Course Glossery,” 2012. [Online]. Available: http://cdsetrain.dtic.mil/derivative/common/cw/data/Derivative_Classification_Glossary.pdf. [Accessed: 06-Jan-2014].
- [18] “Derivative Classification Student Course Lesson 1 Introduction,” 2012. [Online]. Available: http://cdsetrain.dtic.mil/derivative/common/cw/data/DC_SG_Lesson_1_Course_Introduction.pdf. [Accessed: 06-Jan-2014].
- [19] “Derivative Classification Student Course Derivative Classification Lesson 2 Basics,” 2012. [Online]. Available: http://cdsetrain.dtic.mil/derivative/common/cw/data/DC_SG_Lesson_2_Derivative_Classification_Basics.pdf. [Accessed: 06-Jan-2014].
- [20] “Derivative Classification Student Course Lesson 3 Classification Concepts,” 2012. [Online]. Available: http://cdsetrain.dtic.mil/derivative/common/cw/data/DC_SG_Lesson_3_Classification_Concepts.pdf. [Accessed: 06-Jan-2014].
- [21] D. E. Bell and L. J. La Padula, *Secure computer system: Unified exposition and multics interpretation*, 1976.
- [22] M. Crocker, “Cross-Domain Information Sharing in a tactical Environment,” *J. Def. Softw. Eng. March*, vol. 20076, pp. 26-30, 2007.
- [23] P. Hoekstra, *Secrets and Leaks: The Costs and Consequences for National Security*. 2005.
- [24] R. L. Marchant, “Common Access Control Terminology Used in Multilevel Security Systems,” in *Proceedings of the Information Systems Educators Conference ISSN*, 2012, Vol. 2167, p. 1435.
- [25] D. E. Denning, “A lattice model of secure information flow,” *Commun. ACM*, vol. 19, no. 5, pp. 236–243, 1976.
- [26] “The Supreme Court of the United States,” 2011. [Online]. Available: <http://www.supremecourt.gov/opinions/10pdf/10-290.pdf>. [Accessed: 14-Jan-2014].

Appendix A: Table of Definitions

Table 2 Glossary of terms (reproduced with permission from [12])

Term/Acronym	Definition
Access	The ability and opportunity to gain knowledge of classified information.
Classification	The act or process by which information is determined to be classified information.
Classifier	An individual who makes a classification determination and applies a security classification to information or material. A classifier may be an original classification authority or a person who derivatively assigns a security classification based on a properly marked classification source or a classification guide.
Classified National Security Information	Information that has been determined, pursuant to Executive Order 13526 or any predecessor order, or pursuant to the Atomic Energy Act of 1954, to require protection against unauthorized disclosure and is marked to indicate its classified status when in documentary form.
Classification Guidance	Is an authorized source of classification guidance. Within DoD there are three authorized sources for classification guidance: The Security Classification Guide (SCG), a properly marked source document, and the DD Form 254.
Classification Guide	Also referred to as the Security Classification Guide (SCG). A document issued by an authorized original classifier that identifies the elements of information regarding a specific subject that must be classified and establishes the level and duration of classification for each such element. A SCG is a collection of precise, comprehensive guidance about a specific program, system, operation, or weapon system telling what elements of information are classified. For each element of information, the SCG includes its classification level, the reasons for that classification, and the downgrading/duration of classification.
Cognizant Security Agencies (CSAs)	Agencies of the Executive Branch that have been authorized by Executive Order 12829 to establish an industrial security program to safeguard classified information under the jurisdiction of these agencies when disclosed or released to U.S. Industry. These agencies are: The Department of Defense, Department of Energy, Central Intelligence Agency, and Nuclear Regulatory Commission.

Cognizant Security Office (CSO)	The organizational entity delegated by the head of a CSA to administer industrial security on behalf of the CSA.
Compilation	The concept also known as aggregation, which involves combining or associating individually, unclassified information which reveals an additional association or relationship that warrants protection as classified information. This concept also applies to elements of information classified as a lower level which become classified at a higher level when combined.
Compromise	An unauthorized disclosure of information.
Confidential	The classification level applied to information, the unauthorized disclosure of which reasonably could be expected to cause damage to the national security that the original classification authority is able to identify or describe.
Contained in	The concept that refers to the process of extracting classified information as it is stated in an authorized source of classification guidance without the need for additional interpretation or analysis, and incorporating this information in a new document.
DD Form 254, (Department of Defense Security Classification Specification	This form provides classification guidance to contractors performing on classified contracts. It informs them of the level of information they will need to access, the required level of security clearance for access, and the performance requirements to include safeguarding, special security requirements, etc. This form is an authorized source of classification used by derivative classifiers.
Damage to National Security	Harm to the national defense or foreign relations of the United States from the unauthorized disclosure of information.
Declassification	The authorized change in the status of information from classified information to unclassified information.
Derivative Classification	The process of determining whether information has already been originally classified and, if it has, ensuring that it continues to be identified as classified by marking or similar means when included in newly created material.
Derivative Classifier	The individual responsible for ensuring that they apply the highest possible level of security classification when derivatively classifying information. These individuals bear the principal responsibility for the accuracy of the derivative classification.

DoD 5200.1-R Department of Defense Industrial Security Program	The Regulation that implements Executive Order 12958, as amended “Classified National Security Information,” and associated OMB directives within the DoD. It applies to all Components of the DoD. It establishes the DoD Information Security Program to promote proper and effective classification, protection, and downgrading of official information requiring protection in the interest of the national security. It also promotes the declassification of information no longer requiring such protection.
DoD 5220.22-M National Information Security Program Operating Manual (NISPOM)	The manual issued in accordance with the National Industrial Security Program that prescribes the requirements, restrictions, and other safeguards to prevent unauthorized disclosure of classified information.
Downgrading:	A determination that information classified at a specific level shall be classified at a lower level.
Duration of Classification:	A determination made by an original classifier, at the time of original classification, on the length of time information will require protection of security classification.
Extract	Taking information directly from an authorized source of classification guidance and stating it verbatim in a new or different document.
Facility Security Officer (FSO)	A U.S. citizen employee, appointed by a contractor who will supervise and direct security measures necessary for implementing the NISPOM and other Federal requirements for classified information.
Generate	Taking information from an authorized source of classification guidance and using it in another form or media.
Government Contracting Agency (GCA)	An element of an agency designated by the agency head and delegated broad authority regarding acquisition functions.
Information	Any knowledge that can be communicated or documentary material that is owned by, produced by or for, or is under the control of the United States Government. “Control” means the authority of the Agency that originates information, or its successor in function, to regulate access to the information.
Information Security	The result of any system of administrative policies and procedures for identifying, controlling, and protecting from unauthorized disclosure, information the protection of which is authorized by executive order.
Marking	The principal means to inform holders of classified information about specific protection requirements for

	that information. Marking and designation of classified information are the specific responsibility of original and derivative classifiers.
Multiple Sources	Two or more source documents, classification guides, or a combination of both.
Need-to-Know (NTK)	A determination made by an authorized holder of classified information that a prospective recipient requires access to specific classified information in order to perform or assist in a lawful and authorized governmental function.
National Security	The national defense or foreign relations of the United States
Original Classification	An initial determination that information requires, in the interest of national security, protection against unauthorized disclosure.
Original Classification Authority	An individual authorized in writing, either by the President, or by Agency Heads or other officials designated by the President, to originally classify information.
Paraphrase/Restate	Taking information from an authorized source of classification guidance and re-wording it in a new or different document.
Regrade	To raise or lower the classification assigned to an item of information.
Revealed by	The concept applied when derivative classifiers incorporate classified information from an authorized source of classification guidance into a new document, which is not clearly or explicitly stated in the source document.
Safeguarding	Measures and controls that are prescribed to protect classified information.
Secret	The classification level applied to information, the unauthorized disclosure of which reasonably could be expected to cause serious damage to national security that the original classification is able to identify or describe.
Security Classification Guide (SCG)	Also referred to as a Classification Guide. A document issued by an authorized original classifier that identifies the elements of information regarding a specific subject that must be classified and establishes the level and duration of classification for each such element. A SCG is a collection of precise, comprehensive guidance about a specific program, system, operation, or weapon system telling what elements of information are classified. For each element of information, the SCG includes its

	classification level, the reasons for that classification, and the downgrading/duration of classification.
Source Document	An authorized source of classification used by a derivative classifier, from which information is extracted, paraphrased, restated, and/or generated in a new form for inclusion in another document.
Top Secret	The classification level applied to information, the unauthorized disclosure of which reasonable could be expected to cause exceptionally grave damage to the national security that the original classification authority is able to identify or describe.
Unauthorized Disclosure	A communication or physical transfer of classified information to an unauthorized recipient.

Appendix B: Classification Rule Builder Source

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Xml.Serialization;

namespace XMLRules
{
    /// <summary>
    /// This class contains the List of Rules as well as the methods to serialize
    (open) and deserialize (save) the xml file containing the rule data
    /// </summary>
    public class Rules
    {
        /// <summary>
        /// RuleSetList contains all of the rules from the XML file
        /// </summary>
        public List<ClassificationRules> RuleSetList { get; private set; }

        /// <summary>
        /// Initialize the List of rules
        /// </summary>
        public Rules()
        {
            RuleSetList = new List<ClassificationRules>();
        }

        /// <summary>
        /// The OpenXml method opens the xml file and reads it into the List if it
        is a valid XML file type
        /// This method can be modified to open the XML from any source including
        an XML database
        /// </summary>
        /// <param name="filename">This is the name of the file to open</param>
        public Boolean OpenXml(String filename)
        {
            var serializer = new XmlSerializer(typeof(List<ClassificationRules>));
            using (var stream = File.OpenRead(filename))
            {
                try
                {
                    var other =
(List<ClassificationRules>)(serializer.Deserialize(stream));
                    RuleSetList.Clear();
                    RuleSetList.AddRange(other);
                    return true;
                }
                catch (InvalidOperationException)
                {
                    return false;
                }
            }
        }

        //need to modify this to save the list to the existdb
        /// <summary>
        /// The SaveXml method saves the list contents to an XML file
    }
}

```

```

    /// This method can be modified to save the List to any XML source
    including an XML database if desired
    /// It is important to note that this file is saved in plaintext and
    because of the sensitivity of the data it should be protected
    /// </summary>
    /// <param name="filename">This is the name of the file to save</param>
    public void SaveXml(String filename)
    {
        var serializer = new XmlSerializer(typeof(List<ClassificationRules>));
        using (TextWriter writer = new StreamWriter(filename))
        {
            serializer.Serialize(writer, RuleSetList);
        }
    }
    /// <summary>
    /// This method adds a new rule to the Rule List
    /// </summary>
    /// <param name="classificationText">This is the text that is being
    classified for example "meeting at one"</param>
    /// <param name="classificationLevel">"This is the level of classification
    "Top Secret","Secret","Classified" etc</param>
    /// <param name="classificationCompartment">This is the classification
    compartment in our example code it is "Red", "Green","Blue" or a combination of
    the three</param>
    /// <param name="expiration">This is the expiration date of the rule. The
    DateTime type is just the Date without the time stamp </param>
    /// <param name="referenceFile">This is the file name of the original
    classified document.</param>
    public void AddNewRule(String classificationText, String
    classificationLevel, String classificationCompartment, DateTime expiration, String
    referenceFile)
    {
        ClassificationRules newRules = new ClassificationRules();
        newRules.ClassificationString = classificationText;
        newRules.ClassificationLevel = classificationLevel;
        newRules.ClassificationCompartment = classificationCompartment;
        newRules.ExpirationDateTime = expiration;
        newRules.ReferenceFile = referenceFile;
        newRules.UserName = Environment.UserDomainName;
        newRules.ComputerName = Environment.MachineName;
        newRules.CreationDate = DateTime.Now;
        RuleSetList.Add(newRules);
    }
    /// <summary>
    /// This checks the existing list to see if there is a duplicate entry.
    Since we do not want to have duplicate classification rules
    /// </summary>
    /// <param name="classificationText">This is the text that is being
    classified for example "meeting at one"</param>
    /// <param name="classificationLevel">This is the level of classification
    "Top Secret","Secret","Classified" etc</param>
    /// <param name="classificationCompartment">This is the classification
    compartment in our example code it is "Red", "Green","Blue" or a combination of
    the three</param>
    /// <param name="expiration">This is the expiration date of the rule. The
    DateTime type is just the Date without the time stamp </param>
    /// <param name="referenceFile">This is the file name of the original
    classified document.</param>

```

```

    /// <returns></returns>
    public Boolean InList(String classificationText, String
classificationLevel, String classificationCompartment,
    DateTime expiration, String referenceFile)
    {
        Boolean inList = false; //default to not in the list
        ClassificationRules newRules = new ClassificationRules();
        newRules.ClassificationString = classificationText;
        newRules.ClassificationLevel = classificationLevel;
        newRules.ClassificationCompartment = classificationCompartment;
        newRules.ExpirationDateTime = expiration;
        newRules.ReferenceFile = referenceFile;
        if (RuleSetList.Any(newRules.Equals))
        {
            return true;
        }
        return inList;
    }
}
/// <summary>
///
/// </summary>
public class ClassificationRules
{
    [XmlElement("ComputerName")]
    public string ComputerName { get; set; }
    [XmlElement("UserName")]
    public string UserName { get; set; }
    [XmlElement("String")]
    public string ClassificationString { get; set; }
    [XmlElement("Level")]
    public string ClassificationLevel { get; set; }
    [XmlElement("Compartment")]
    public string ClassificationCompartment { get; set; }
    [XmlElement("Expiration")]
    public DateTime ExpirationDateTime { get; set; }
    [XmlElement("CreationDate")]
    public DateTime CreationDate { get; set; }
    [XmlElement("FileReference")]
    public String ReferenceFile { get; set; }
    /// <summary>
    ///
    /// </summary>

    /// <summary>
    /// This is a standard CompareTo method for helping to find duplicate
    entries in the list. We are only comparing on Classification String and Reference
    File Matches because even if the
    /// dates do not match there should not be two entries with the same
    information
    /// </summary>
    /// <param name="that">The classificationrules that we are
    comparing</param>
    /// <returns>returns an int of the distance between what we are comparing
    a 0 value returned means equal</returns>
    public int CompareTo(ClassificationRules that)

```

```

        {
            if (System.String.Compare(this.ClassificationString,
that.ClassificationString, System.StringComparison.Ordinal) == 0)
            {
                return this.ReferenceFile.CompareTo(that.ReferenceFile);
            }
            return System.String.Compare(ClassificationString,
that.ClassificationString, System.StringComparison.Ordinal);
        }
        /// <summary>
        /// Checking for equality between two ClassificationRules
        /// </summary>
        /// <param name="that">the incoming rule to compare</param>
        /// <returns>Returns true if equal otherwise it returns false</returns>
        public Boolean Equals(ClassificationRules that)
        {
            return CompareTo(that) == 0;
        }
    }
}

```

```

using System;
using System.Linq;
using System.Windows.Forms;

```

```

namespace XMLRules
{
    public partial class RuleBuilderForm : Form
    {
        //indicates if the file is saved or not which is important if people close
        early without saving
        //Ruleset contains the list of all of our rules
        private readonly Rules Ruleset;
        //these are the classification options presented to the end user
        private readonly string[] classificationStrings =
        {
            "Unclassified", "For Official Use Only", "Secret",
            "Top Secret"
        };

        //these are the classification compartments presented to the end user
        private readonly string[] compartmentString = {"Red", "Green", "Blue"};
        private String ReferenceFile = String.Empty;
        public Boolean Saved;
        //this is the name of our XML file containing the rules
        private string fileName = String.Empty;

        public RuleBuilderForm()
        {
            //makes the form not resizable
            FormBorderStyle = FormBorderStyle.FixedSingle;
            //create a new List of rules
            Ruleset = new Rules();
            //set saved default value to true
            Saved = true;
            InitializeComponent();
        }
    }
}

```



```

        classificationLevelComboBox.DataSource = classificationStrings;
        classificationCompartmentCheckedListBox.DataSource =
compartmentString;
        ResetValues();
        //turn off all boxes on the form so user opens xml file or starts a
new one
        DisableAllBoxes();
        //Forces the user to select a date from when the program is run up
until 50 years from program run date
        dateTimePicker1.MinDate = DateTime.Now;
        dateTimePicker1.MaxDate = DateTime.Now.AddYears(50);
    }

    /// <summary>
    ///     Opens an xml file of classification rules and if it is the correct
file type then allowing the user to continue
    /// </summary>
    private void openToolStripMenuItem_Click(object sender, EventArgs e)
    {
        var openXML = new OpenFileDialog();
        openXML.Filter = "xml files (*.xml)|*.xml";
        openXML.RestoreDirectory = true;
        if (openXML.ShowDialog() == DialogResult.OK)
        {
            fileName = openXML.FileName;
            if (Ruleset.OpenXml(fileName))
            {
                Saved = false;
                EnableAllBoxes();
            }
            else
            {
                MessageBox.Show("You must open the correct XML file type");
            }
        }
    }

    /// <summary>
    ///     This prompts the user with a file save dialog if we dont have a
filename otherwise it just saves the file
    /// </summary>
    private void saveToolStripMenuItem_Click(object sender, EventArgs e)
    {
        var saveXML = new SaveFileDialog();
        saveXML.Filter = "xml files (*.xml)|*.xml";
        openXML.RestoreDirectory = true;

        if (fileName.Equals(String.Empty))
        {
            if (saveXML.ShowDialog() == DialogResult.OK)
            {
                fileName = saveXML.FileName;
                Ruleset.SaveXml(fileName);
                Saved = true;
            }
            else
            {
                MessageBox.Show("You must pick a filename");
            }
        }
    }

```

```

        Saved = false;
    }
}
else
{
    Ruleset.SaveXml(fileName);
    Saved = true;
}
}

/// <summary>
///     This causes the form closing event to file which calls the
RuleBuilderForm_FormClosing method
/// </summary>
private void closeToolStripMenuItem_Click(object sender, EventArgs e)
{
    Close(); //causes the form closing event to fire
}

/// <summary>
///     This button allows you to add the value entered into the rules
set. We also double check to make sure the rule is
///     not duplicated
/// </summary>
private void addValueButton_Click(object sender, EventArgs e)
{
    Boolean textCheck = true;
    String classificationText = classificationTextBox.Text;
    String classificationLevel = classificationLevelComboBox.Text;
    String classificationCompartment =
classificationCompartmentCheckedListBox.CheckedItems.Cast<object>()
        .Aggregate("", (current, item) => current + (item.ToString() +
" "));
    if (ReferenceFile == String.Empty)
    {
        MessageBox.Show("You must select a reference file");
        textCheck = false;
    }
    //need to check classification level and text to make sure they are
not empty
    if (classificationText == String.Empty && classificationLevel ==
String.Empty)
    {
        MessageBox.Show("You must fill in values for Classification text
and level.");
        textCheck = false;
    }
    else
    {
        if (classificationText == String.Empty)
        {
            MessageBox.Show("Classification Text Cannot Be Empty");
            textCheck = false;
        }
        else if (classificationLevel == String.Empty)
        {
            MessageBox.Show("You must select a classification level");

```

```

        textCheck = false;
    }
}
//all the boxes are good here but now we need to make sure we don't
have the info in the list already

if (textCheck)
{
    //check for duplicate here
    if (Ruleset.InList(classificationText, classificationLevel,
classificationCompartment,
    dateTimePicker1.Value, ReferenceFile))
    {
        MessageBox.Show(
            "That value is already in the classification rules, please
double check your entries.");
    }
    else
    {
        Ruleset.AddNewRule(classificationText, classificationLevel,
classificationCompartment,
            dateTimePicker1.Value, ReferenceFile);
        Saved = false;
        ResetValues();
    }
}
}

/// <summary>
///     This button presents a file browse dialog for the original
classifier to select the file they are creating rules
///     for
/// </summary>
private void referenceFileButton_Click(object sender, EventArgs e)
{
    if (openWordDoc.ShowDialog() == DialogResult.OK)
    {
        ReferenceFile = openWordDoc.SafeFileName;
        fileNameTextBox.Text = openWordDoc.SafeFileName;
    }
}

/// <summary>
///     This resets all of the form values
/// </summary>
private void ResetValues()
{
    classificationTextBox.Text = String.Empty;
    fileNameTextBox.Text = String.Empty;
    classificationLevelComboBox.SelectedIndex = -1;
    for (int i = 0; i <
classificationCompartmentCheckedListBox.Items.Count; i++)
    {
        if (classificationCompartmentCheckedListBox.GetItemChecked(i))
        {
            classificationCompartmentCheckedListBox.SetItemCheckState(i,
CheckState.Unchecked);
        }
    }
}

```

```

    }
    dateTimePicker1.Value = DateTime.Now;
    ReferenceFile = String.Empty;
}

/// <summary>
///     This disables all of the form boxes
/// </summary>
private void DisableAllBoxes()
{
    saveToolStripMenuItem.Enabled = false;
    classificationTextBox.Enabled = false;
    classificationLevelComboBox.Enabled = false;
    dateTimePicker1.Enabled = false;
    referenceFileButton.Enabled = false;
    fileNameTextBox.Enabled = false;
    classificationCompartmentCheckedListBox.Enabled = false;
    addValueButton.Enabled = false;
}

/// <summary>
///     This enables all of the form boxes
/// </summary>
private void EnableAllBoxes()
{
    saveToolStripMenuItem.Enabled = true;
    classificationTextBox.Enabled = true;
    classificationLevelComboBox.Enabled = true;
    dateTimePicker1.Enabled = true;
    referenceFileButton.Enabled = true;
    fileNameTextBox.Enabled = true;
    classificationCompartmentCheckedListBox.Enabled = true;
    addValueButton.Enabled = true;
}

/// <summary>
///     Watches if the form is closing and checks to make sure it is saved
/// </summary>
private void RuleBuilderForm_FormClosing(object sender,
FormClosingEventArgs e)
{
    if (!Saved)
    {
        DialogResult dialogResult = MessageBox.Show("Would you like to
save your work?", "WARNING",
        MessageBoxButtons.YesNo);
        if (dialogResult == DialogResult.Yes)
        {
            saveToolStripMenuItem_Click(sender, e);
            //Close();
            this.Dispose();
        }
        else if (dialogResult == DialogResult.No)
        {
            //Close();
            this.Dispose();
        }
    }
}

```

```

    }

    /// <summary>
    ///     If the user selects new we enable all of the boxes so they can
save later
    /// </summary>
    private void newToolStripMenuItem_Click(object sender, EventArgs e)
    {
        EnableAllBoxes();
    }

    /// <summary>
    ///     This presents a save as file dialog so the user can create another
copy of their xml file
    /// </summary>
    private void saveAsToolStripMenuItem_Click(object sender, EventArgs e)
    {
        var saveXML = new SaveFileDialog();
        saveXML.Filter = "xml files (*.xml)|*.xml";
        openXML.RestoreDirectory = true;
        if (saveXML.ShowDialog() == DialogResult.OK)
        {
            fileName = saveXML.FileName;
            Ruleset.SaveXml(fileName);
            Saved = true;
        }
    }

    /// <summary>
    ///     This displays an about box for the program.
    /// </summary>
    private void aboutToolStripMenuItem1_Click(object sender, EventArgs e)
    {
        var aboutBox1 = new AboutBox1();
        aboutBox1.Show();
    }
}
}
}

```

Appendix C: Classification Rule Builder Manual

To start the software you will need to double click on the Classification Rule Builder icon on your desktop. You will be presented with the screen shown in Figure 17.

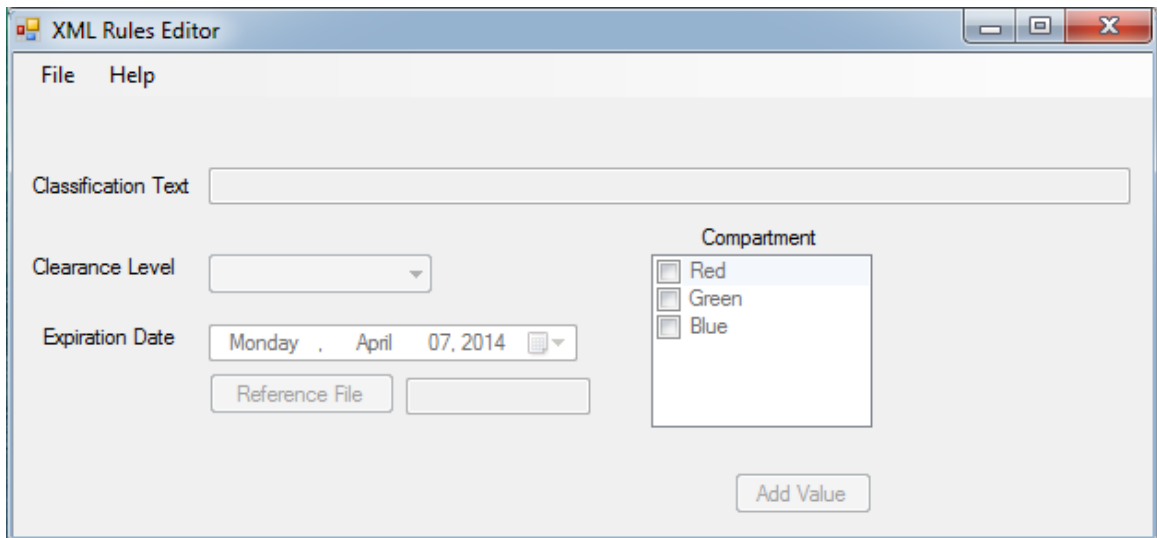


Figure 17 Program Default Screen

The screen will not have any input boxes active because you will first need to start a new file or open an existing xml file. These options are presented under the file menu as shown in Figure 18.

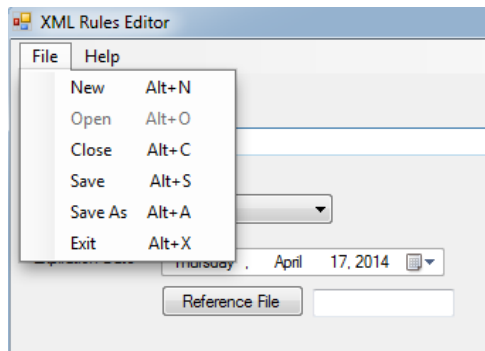
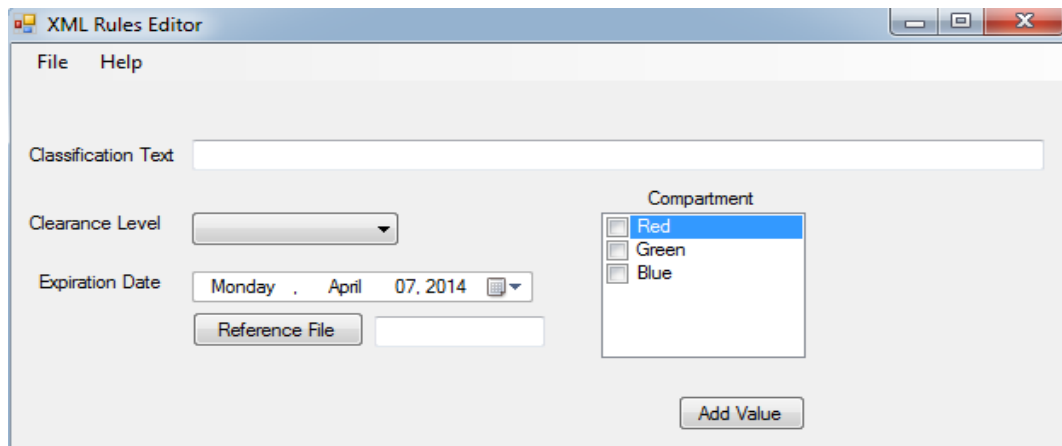


Figure 18 File Menu

The file menu options are:

1. New – Start a new xml file. Select this when you want to create a new ruleset from scratch.
2. Open – Open an existing rule set xml file. Select this when you want to continue working on an existing rule set.
3. Close – Close the current xml rule file. Select this option when you want to close the current rule set you are working on. If the file has not been saved it will be saved and if a file name was not previously selected you will be prompted to name the file.
4. Save – Save the current file you are working on. This will cause one of two things to occur. First if you are working on a new file this will give you the option of naming the file and then saving it or secondly it will save the current rule set you are working on. Warning if you have not added the current rule that is on the screen into the rule set it will not be saved in the xml file.
5. Save As – Save the current file you are working on with a new name. This works exactly as the save button but gives you the opportunity to select a new file name for the rule set you are working on.
6. Exit – Exit the current program. This will terminate the program. If you have unsaved data it will prompt you to save the data file first. Warning if you do not save the data file you will lose all of your changes.



The screenshot shows the 'XML Rules Editor' window. It has a menu bar with 'File' and 'Help'. Below the menu bar, there is a 'Classification Text' text box. Underneath that is a 'Clearance Level' dropdown menu. To the right of the dropdown is a 'Compartment' list with three items: 'Red', 'Green', and 'Blue', each with a checkbox. Below the dropdown is an 'Expiration Date' field showing 'Monday . April 07, 2014' with a calendar icon. Below the date field is a 'Reference File' button and an empty text box. At the bottom right of the window is an 'Add Value' button.

Figure 19 All Options Enabled

After selecting the new or open option you will be presented a screen that has all of the input boxes enabled as shown in Figure 19. To enter a new classification rule you will need to fill out all of the required boxes or you will be presented with error messages as shown in Figure 20 a missing reference file error and Figure 21 the missing classification test and level error.

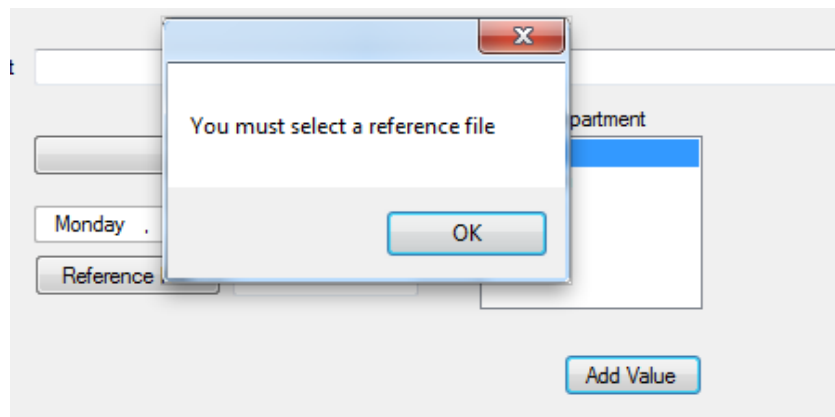


Figure 20 Reference File Not Selected Error

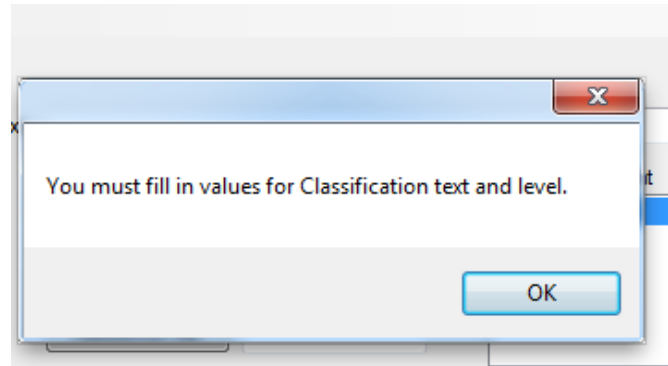


Figure 21 Classification Text Missing Error

To create a successful rule entry you must have all entries completed with compartment being the only optional entry. Each entry has the following requirements.

1. Classification Text – This field can have an entry up to a maximum of 2000 characters. It is important that you enter the classification text correctly as this is what is used when running the derivative classification program.
2. Clearance Level – This will present a drop down box, as shown in Figure 22 that contains all of the valid clearance levels available to you. You should select the correct level that corresponds with the classification text you have entered.

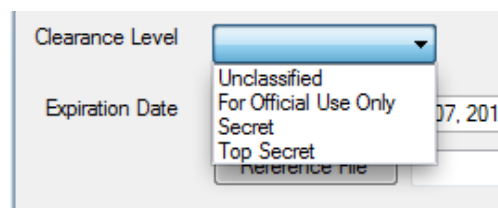
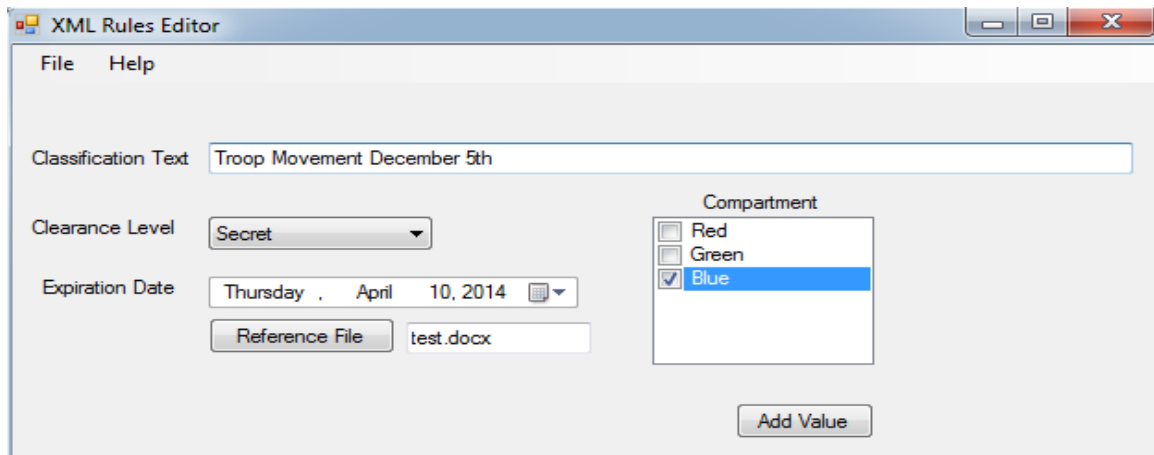


Figure 22 Clearance Level Drop Down

3. Expiration Date – this will present a date picker box that will allow you to select a maximum expiration date of the current day plus 50 years.

4. Reference File – this will present a file system browser dialog that will allow you to select the reference file that is the document that is the reference for this rule. This is the file that the security classification guide would have been written for.
5. Compartment – this is the only optional box and will allow you to select zero to all options for the classification compartment.

A properly filled out form will look like the form in Figure 23. Notice that all boxes have an entry and we will be able to now add this rule to the rule set using the Add Value button. After all entries are added the form will reset and allow you to continue entering rules. It is important that while rules are being entered you carefully check spelling and make sure to add all possible combinations of text that will need to be classified in the new document. It is also very important that you carefully double check all entries before adding them to the rule set, while the xml file is manually editable it will make things a lot easier if you double check all of you work first.



The screenshot shows the 'XML Rules Editor' window. The form contains the following fields and options:

- Classification Text:** Troop Movement December 5th
- Clearance Level:** Secret (dropdown menu)
- Expiration Date:** Thursday, April 10, 2014 (calendar icon)
- Reference File:** test.docx
- Compartment:** A list box with three options: Red (unchecked), Green (unchecked), and Blue (checked).
- Add Value:** A button at the bottom right of the form.

Figure 23 Program Filled Out Correctly

Appendix D: Derivative Classifier Source Code

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Text.RegularExpressions;
using System.Xml;

namespace SecTagMain
{
    class DerivativeClassifier : ITaggingModule
    {
        readonly Rules _classificationRules;
        private String _classificationErrors;
        SecurityTagManager _stm;

        /// <summary>
        /// Default constructor which create a new Rules set and fills it from our
        xml file.. at some point this should be a value passed in
        /// </summary>
        public DerivativeClassifier()
        {
            _classificationRules = new Rules();
            _classificationRules.OpenXml(Cnst.RulesFile);
            _stm = SecurityTagManager.GetInstance();
        }

        /// <summary>
        /// This walks through the xml document and changes the values of the
        paragraphs in the word document to their correct classification value from the xml
        label
        /// </summary>
        /// <param name="xml">Xml of the document we are working on</param>
        /// <returns>The modified document with the correct labels in the
        paragraph</returns>
        public XmlDocument ChangeParagraphLabel(XmlDocument xml)
        {
            NameTable nt = new NameTable();
            XmlNamespaceManager ns = new XmlNamespaceManager(nt);
            ns.AddNamespace("w",
                "http://schemas.openxmlformats.org/wordprocessingml/2006/main");
            ns.AddNamespace("stox", "http://securetagged.com");

            XmlNodeList nodes = xml.SelectNodes("//w:p", ns);

            if (nodes != null)
            {
                foreach (XmlNode node in nodes)
                {
                    XmlNode dnode = node.SelectSingleNode("./w:t[1]", ns);

                    if (dnode != null)
                    {
                        if (!dnode.InnerText.StartsWith("[") //does not have a
tag
                            {
                                {
                                    if (node.Attributes != null)
                                    {

```

```

        dnode.InnerText = "[" +
node.Attributes["stox:label"].Value.ToUpper();
    }
}
}
else //has a tag
{
    //get the tag value
    string currentTagValue = dnode.InnerText.Split(new[]
    {
        ',', ']'
    })[1];
    //replace the tag value with correct value
    if (node.Attributes != null)
        dnode.InnerText =
dnode.InnerText.Replace(currentTagValue,
node.Attributes["stox:label"].Value.ToUpper());
}
}
}
}
return xml;
}

/// <summary>
/// Method to searches though the rule list and checks for matches within
a node, if a match is found we check the total value from the node with the rule
/// if the rule has a greater classification value then we need to modify
the xml with the new rule
/// if the rule has an equal value we need to mark it for review so an
original classifier can investigate further
/// </summary>
/// <param name="node">The node we are working on and looking for a
match</param>
public void LookForMatch(XmlNode node)
{
    //iterate through the rules looking for a string match in the
paragraph
    foreach (ClassificationRules rules in
_classificationRules.RuleSetList)
    {
        SecurityTag paragraph = new SecurityTag(node);
        SecurityTag rule = new SecurityTag(rules.ClassificationLevel, new
List<string>(rules.ClassificationCompartment.Split(' ')));
        if (Regex.IsMatch(node.InnerText, rules.ClassificationString,
RegexOptions.IgnoreCase)) //looking to see if we have a match in the
classification string
        {
            if (DateTime.Now < rules.ExpirationDateTime) //make sure the
rule has not expired
            {
                if
(node.Attributes["stox:label"].Value.ToUpper().Equals("Needs Review".ToUpper()))
                {
                    GenerateClassificationReviewFile(node, rules,
"Paragraph has already been marked for review");
                    return;
                }
            }
        }
    }
}

```

```

do upgrade //if match check dominance if rule dominate paragraph then
    if (_stm.Dominates(rule, paragraph) &&
!_stm.Dominates(paragraph, rule)) //strict dominance
    {
        node.Attributes["stox:label"].Value =
rules.ClassificationLevel;
        node.Attributes["stox:compartments"].Value =
rules.ClassificationCompartment;
    }
    else if (!_stm.Dominates(rule, paragraph) &&
!_stm.Dominates(paragraph, rule)) //does not dominate
    {
        //only need to mark for review if labels are the same
but the compartments are different
        if (
            node.Attributes["stox:label"].Value.ToUpper()
                .TrimStart()
                .TrimEnd()

.Equals(rules.ClassificationLevel.ToUpper().TrimStart().TrimEnd()))
        {
            GenerateClassificationReviewFile(node, rules,
"Classification levels are the same but the compartments do not match.");
            node.Attributes["stox:label"].Value = "NEEDS
REVIEW";
        }
        else
        {
            GenerateClassificationReviewFile(node, rules,
"Rule Values Did Not Dominate Current Paragraph Values");
        }
    }
}
else
{
    GenerateClassificationReviewFile(node, rules, "Match found
but the rule has expired");
}
}
}

private void GenerateClassificationReviewFile(XmlNode node,
ClassificationRules rules, String reasonForFailure)
{
    StringBuilder sb = new StringBuilder();
    sb.AppendLine("Classification string to be checked is: " +
rules.ClassificationString);
    sb.AppendLine("Current Node classification level is: " +
node.Attributes["stox:label"].Value);
    sb.AppendLine("Rule classification level is: " +
rules.ClassificationLevel);
    sb.AppendLine("Node compartments are: " +
node.Attributes["stox:compartments"].Value);
}
}
}

```

```

        sb.AppendLine("Rule compartments are: " +
rules.ClassificationCompartment);
        sb.AppendLine("Current Date is: " + DateTime.Now.ToShortDateString());
        sb.AppendLine("Rule Expiration Date is: " +
rules.ExpirationDateTime.ToShortDateString());
        sb.AppendLine("File containing xml rules is " + Cnst.RulesFile);
        sb.AppendLine("Reference document for this rule is " +
rules.ReferenceFile);
        sb.AppendLine(reasonForFailure);
        sb.AppendLine();
        _classificationErrors += sb.ToString();
    }

    public XmlDocument Process(XmlDocument xml, SecurityTag tag)
    {
        NameTable nt = new NameTable();
        XmlNamespaceManager ns = new XmlNamespaceManager(nt);
        ns.AddNamespace("w",
"http://schemas.openxmlformats.org/wordprocessingml/2006/main");
        ns.AddNamespace("stox", "http://securetagged.com");
        XmlNodeList nodes = xml.SelectNodes("//w:p", ns);
        if (nodes != null)
        {
            foreach (XmlNode node in nodes) //walk through all of the xml
nodes
            {
                LookForMatch(node); //this will modify the node Security label
if necessary which is important since we need it for the ChangeParagraphLabel
call
            }
            xml = ChangeParagraphLabel(xml);
            WriteReviewErrorsToFile();
        }
        return xml;
    }
    /// <summary>
    /// Write error string to output file and overwrite the old error message
file, this can be changed to append if the false is changed to true
    /// </summary>
    private async void WriteReviewErrorsToFile()
    {
        using (StreamWriter outfile = new StreamWriter(Cnst.ErrorFile, false))
//if you want the file to not be overwritten change false to true
        {
            await outfile.WriteAsync(_classificationErrors);
        }
    }
    /// <summary>
    /// Helper method to debug rule checks
    /// </summary>
    /// <param name="rules"></param>
    /// <param name="node"></param>
    private void Debug(ClassificationRules rules, XmlNode node)
    {
        Console.WriteLine(node.InnerText);
        Console.WriteLine("Node label is " +
node.Attributes["stox:label"].Value);
    }

```



```
        Console.WriteLine("Rule label is " + rules.ClassificationLevel);
        Console.WriteLine("Node compartments are " +
node.Attributes["stox:compartments"].Value);
        Console.WriteLine("Rule compartments are " +
rules.ClassificationCompartment);
        Console.WriteLine();
    }
}
}
```

Appendix E: Classification Rules Test Source Code

```

using System;
using System.IO;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using XMLRules;

namespace ClassificationRulesTest
{
    [TestClass]
    public class ClassificationRulesTest
    {
        private const String FileName = @"..\..\..\unittest.xml";
        private const String BadFileName = @"..\..\..\unittestbadxml.xml";
        private const String SaveFileName = @"..\..\..\savefile.xml";

        [TestMethod]
        public void VerifyEmptyClassificationRulesListCreated()
        {
            //this gives us an empty list of rules so we should make sure we have
            an empty list
            Rules ruleSet = new Rules();
            Assert.AreEqual(0,ruleSet.RuleSetList.Count);
        }

        [TestMethod]
        public void VerifyOpenXml()
        {
            Rules ruleSet = new Rules();
            //we dont need to verify an empty list exists because we have done
            with already
            ruleSet.OpenXml(FileName); //should populate list
            Assert.AreEqual(1,ruleSet.RuleSetList.Count); //should be quantity of
            1
        }

        [TestMethod]
        public void VerifyOpenXmlFileContentsClassificationString()
        {
            Rules ruleSet = new Rules();
            ruleSet.OpenXml(FileName);
            Assert.AreEqual("unit testing",
            ruleSet.RuleSetList[0].ClassificationString);
        }

        [TestMethod]
        public void VerifyOpenXmlFileContentsClassificationLevel()
        {
            Rules ruleSet = new Rules();
            ruleSet.OpenXml(FileName);
            Assert.AreEqual("UNCLASS",
            ruleSet.RuleSetList[0].ClassificationLevel);
        }

        [TestMethod]
        public void VerifyOpenXmlFileContentsCompartment()
        {
            Rules ruleSet = new Rules();
            ruleSet.OpenXml(FileName);
        }
    }
}

```

```

        Assert.AreEqual("BLACK",
ruleSet.RuleSetList[0].ClassificationCompartment.TrimEnd());
    }

    [TestMethod]
    public void VerifyOpenXmlFileContentsExpirationDate()
    {
        Rules ruleSet = new Rules();
        DateTime testDateTime = new DateTime(2026, 04, 18, 12, 12, 03);
        ruleSet.OpenXml(FileName);
        Assert.AreEqual(testDateTime,
ruleSet.RuleSetList[0].ExpirationDateTime);
    }

    [TestMethod]
    public void VerifyOpenXmlFileContentsReferenceFile()
    {
        Rules ruleSet = new Rules();
        ruleSet.OpenXml(FileName);
        Assert.AreEqual("test.docx", ruleSet.RuleSetList[0].ReferenceFile);
    }

    [TestMethod]
    public void VerifyBadXmlException()
    {
        Rules ruleSet = new Rules();
        Boolean result;
        result = ruleSet.OpenXml(BadFileName);
        //method will return false if file was not correct
        Assert.AreEqual(false,result);
    }

    [TestMethod]
    public void VerifyAddNewRule()
    {
        Rules ruleSet = new Rules();
        int currentListSize = ruleSet.RuleSetList.Count;
        ruleSet.AddNewRule("Fake Rule", "Secret", "Blue ", DateTime.Now,
"testing.docx");
        int newListSize = ruleSet.RuleSetList.Count;
        Assert.AreEqual(++currentListSize,newListSize);
    }

    [TestMethod]
    public void VerifyInListTrue()
    {
        DateTime testTime = DateTime.Now;
        Rules ruleSet = new Rules();
        ruleSet.AddNewRule("Fake Rule", "Secret", "Blue ", testTime,
"testing.docx");
        Assert.IsTrue(ruleSet.InList("Fake Rule", "Secret", "Blue ", testTime,
"testing.docx"));
    }

    [TestMethod]
    public void VerifyInListFalse()
    {
        DateTime testTime = DateTime.Now;

```

```

        Rules ruleSet = new Rules();
        Assert.IsFalse(ruleSet.InList("Fake Rule", "Secret", "Blue ",
testTime, "testing.docx"));
    }

    [TestMethod]
    public void VerifyXmlFileSave()
    {
        File.Delete(SaveFileName);
        Rules ruleSet = new Rules();
        ruleSet.AddNewRule("Fake Rule", "Secret", "Blue
", DateTime.Now, "testing.docx");
        ruleSet.SaveXml(SaveFileName);
        Assert.IsTrue(File.Exists(SaveFileName));
    }

    [TestMethod]
    public void VerifyClassificationStringSet()
    {
        ClassificationRules rules = new ClassificationRules();
        rules.ClassificationString = "Testing";
        Assert.AreEqual("Testing", rules.ClassificationString);
    }

    [TestMethod]
    public void VerifyClassificationLevelSet()
    {
        ClassificationRules rules = new ClassificationRules();
        rules.ClassificationLevel = "Secret";
        Assert.AreEqual("Secret", rules.ClassificationLevel);
    }

    [TestMethod]
    public void VerifyClassificationCompartmentSet()
    {
        ClassificationRules rules = new ClassificationRules();
        rules.ClassificationCompartment = "Blue ";
        Assert.AreEqual("Blue ", rules.ClassificationCompartment);
    }

    [TestMethod]
    public void VerifyExpirationDateTimeSet()
    {
        DateTime testDateTime = new DateTime();
        ClassificationRules rules = new ClassificationRules();
        rules.ExpirationDateTime = testDateTime;
        Assert.AreEqual(testDateTime, rules.ExpirationDateTime);
    }

    [TestMethod]
    public void VerifyReferenceFileNameSet()
    {
        ClassificationRules rules = new ClassificationRules();
        rules.ReferenceFile = "test.docx";
        Assert.AreEqual("test.docx", rules.ReferenceFile);
    }

    [TestMethod]
    public void VerifyCompareToEqual()

```

```

    {
        ClassificationRules rules = new ClassificationRules();
        rules.ClassificationString = "Testing";
        rules.ReferenceFile = "test.docx";
        ClassificationRules rulesTwo = new ClassificationRules();
        rulesTwo.ClassificationString = "Testing";
        rulesTwo.ReferenceFile = "test.docx";
        Assert.AreEqual(0, rules.CompareTo(rulesTwo));
    }

    [TestMethod]
    public void VerifyCompareToNotEqual()
    {
        ClassificationRules rules = new ClassificationRules();
        rules.ClassificationString = "Testing";
        rules.ReferenceFile = "test.docx";
        ClassificationRules rulesTwo = new ClassificationRules();
        rulesTwo.ClassificationString = "Testing some more";
        rulesTwo.ReferenceFile = "testing.docx";
        Assert.AreNotEqual(0, rules.CompareTo(rulesTwo));
    }

    [TestMethod]
    public void VerifyEqualsMethodWhenEqual()
    {
        ClassificationRules rules = new ClassificationRules();
        rules.ClassificationString = "Testing";
        rules.ReferenceFile = "test.docx";
        ClassificationRules rulesTwo = new ClassificationRules();
        rulesTwo.ClassificationString = "Testing";
        rulesTwo.ReferenceFile = "test.docx";
        Assert.IsTrue(rules.Equals(rulesTwo));
    }

    [TestMethod]
    public void VerifyEqualsMethodWhenNotEqual()
    {
        ClassificationRules rules = new ClassificationRules();
        rules.ClassificationString = "Testing";
        rules.ReferenceFile = "test.docx";
        ClassificationRules rulesTwo = new ClassificationRules();
        rulesTwo.ClassificationString = "Testing equality";
        rulesTwo.ReferenceFile = "test.docx";
        Assert.IsFalse(rules.Equals(rulesTwo));
    }
}
}
}

```

Appendix F: Classification Rules Test Xml Files

Unittest.xml File

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfClassificationRules xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ClassificationRules Level="UNCLASS" Compartment="BLACK ">
    <ComputerName>SCOTTAMACK-PC</ComputerName>
    <UserName>ScottAmack</UserName>
    <String>unit testing</String>
    <Expiration>2026-04-18T12:12:03</Expiration>
    <CreationDate>2014-04-18T12:12:40.8261396-07:00</CreationDate>
    <FileReference>test.docx</FileReference>
  </ClassificationRules>
</ArrayOfClassificationRules>
```

Unittestbadxml.xml File

```
<?xml version="1.0" encoding="utf-8"?>
  <String>unit testing</String>
  <Level>Unclassified</Level>
  <Compartment>Blue </Compartment>
  <Expiration>2014-04-26T13:50:31</Expiration>
</ClassificationRules>
</ArrayOfClassificationRules>
```

Unittestsave.xml File:

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfClassificationRules xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <ClassificationRules Level="Secret" Compartment="Blue ">
    <ComputerName>SCOTTAMACK-PC</ComputerName>
    <UserName>ScottAmack-PC</UserName>
    <String>Fake Rule</String>
    <Expiration>2014-04-19T19:16:49.4045123-07:00</Expiration>
    <CreationDate>2014-04-19T19:16:49.4045123-07:00</CreationDate>
    <FileReference>testing.docx</FileReference>
  </ClassificationRules>
</ArrayOfClassificationRules>
```


Appendix G: Derivative Classification Automator Test

XML File

```

<?xml version="1.0" encoding="utf-8"?>
<ArrayOfClassificationRules xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ClassificationRules Level="UNCLASS" Compartment="">
    <ComputerName>SFS-WORKSTATION</ComputerName>
    <UserName>Scott Amack</UserName>
    <String>to insert tables, headers, footers</String>
    <Expiration>2020-04-01T12:05:51</Expiration>
    <CreationDate>2014-04-17T12:06:20.1124478-07:00</CreationDate>
    <FileReference>WordPress Hardening Guide.docx</FileReference>
  </ClassificationRules>
  <ClassificationRules Level="SECRET" Compartment="">
    <ComputerName>SFS-WORKSTATION</ComputerName>
    <UserName>Scott Amack</UserName>
    <String>to insert tables, headers, footers</String>
    <Expiration>2020-04-01T12:05:51</Expiration>
    <CreationDate>2014-04-17T12:06:20.1124478-07:00</CreationDate>
    <FileReference>WordPress Hardening Guide.docx</FileReference>
  </ClassificationRules>
  <ClassificationRules Level="TOPSECRET" Compartment="">
    <ComputerName>SFS-WORKSTATION</ComputerName>
    <UserName>Scott Amack</UserName>
    <String>to insert tables, headers, footers</String>
    <Expiration>2020-04-01T12:05:51</Expiration>
    <CreationDate>2014-04-17T12:06:20.1124478-07:00</CreationDate>
    <FileReference>WordPress Hardening Guide.docx</FileReference>
  </ClassificationRules>
  <ClassificationRules Level="TOPSECRET" Compartment="BLACK ">
    <ComputerName>SFS-WORKSTATION</ComputerName>
    <UserName>Scott Amack</UserName>
    <String>to insert tables, headers, footers</String>
    <Expiration>2020-04-01T12:05:51</Expiration>
    <CreationDate>2014-04-17T12:06:20.1124478-07:00</CreationDate>
    <FileReference>WordPress Hardening Guide.docx</FileReference>
  </ClassificationRules>
  <ClassificationRules Level="TOPSECRET" Compartment="BLACK RED ">
    <ComputerName>SFS-WORKSTATION</ComputerName>
    <UserName>Scott Amack</UserName>
    <String>to insert tables, headers, footers</String>
    <Expiration>2020-04-01T12:05:51</Expiration>
    <CreationDate>2014-04-17T12:06:20.1124478-07:00</CreationDate>
    <FileReference>WordPress Hardening Guide.docx</FileReference>
  </ClassificationRules>
  <ClassificationRules Level="TOPSECRET" Compartment="BLACK RED GREEN ">
    <ComputerName>SFS-WORKSTATION</ComputerName>
    <UserName>Scott Amack</UserName>
    <String>to insert tables, headers, footers</String>
    <Expiration>2020-04-01T12:05:51</Expiration>
    <CreationDate>2014-04-17T12:06:20.1124478-07:00</CreationDate>
    <FileReference>WordPress Hardening Guide.docx</FileReference>
  </ClassificationRules>
  <ClassificationRules Level="SECRET" Compartment="BLACK RED GREEN ">
    <ComputerName>SFS-WORKSTATION</ComputerName>
    <UserName>Scott Amack</UserName>
    <String>to insert tables, headers, footers</String>
    <Expiration>2020-04-01T12:05:51</Expiration>
    <CreationDate>2014-04-17T12:06:20.1124478-07:00</CreationDate>
    <FileReference>WordPress Hardening Guide.docx</FileReference>
  </ClassificationRules>

```

```

</ClassificationRules>
<ClassificationRules Level="UNCLASS" Compartment="BLACK RED GREEN ">
  <ComputerName>SFS-WORKSTATION</ComputerName>
  <UserName>Scott Amack</UserName>
  <String>to insert tables, headers, footers</String>
  <Expiration>2020-04-01T12:05:51</Expiration>
  <CreationDate>2014-04-17T12:06:20.1124478-07:00</CreationDate>
  <FileReference>WordPress Hardening Guide.docx</FileReference>
</ClassificationRules>
<ClassificationRules Level="UNCLASS" Compartment="BLACK RED ">
  <ComputerName>SFS-WORKSTATION</ComputerName>
  <UserName>Scott Amack</UserName>
  <String>to insert tables, headers, footers</String>
  <Expiration>2020-04-01T12:05:51</Expiration>
  <CreationDate>2014-04-17T12:06:20.1124478-07:00</CreationDate>
  <FileReference>WordPress Hardening Guide.docx</FileReference>
</ClassificationRules>
<ClassificationRules Level="UNCLASS" Compartment="BLACK ">
  <ComputerName>SFS-WORKSTATION</ComputerName>
  <UserName>Scott Amack</UserName>
  <String>to insert tables, headers, footers</String>
  <Expiration>2020-04-01T12:05:51</Expiration>
  <CreationDate>2014-04-17T12:06:20.1124478-07:00</CreationDate>
  <FileReference>WordPress Hardening Guide.docx</FileReference>
</ClassificationRules>
<ClassificationRules Level="UNCLASS" Compartment="">
  <ComputerName>SFS-WORKSTATION</ComputerName>
  <UserName>Scott Amack</UserName>
  <String>to insert tables, headers, footers</String>
  <Expiration>2020-04-01T12:05:51</Expiration>
  <CreationDate>2014-04-17T12:06:20.1124478-07:00</CreationDate>
  <FileReference>WordPress Hardening Guide.docx</FileReference>
</ClassificationRules>
<ClassificationRules Level="UNCLASS" Compartment="BLACK ">
  <ComputerName>SFS-WORKSTATION</ComputerName>
  <UserName>Scott Amack</UserName>
  <String>quick style gallery</String>
  <Expiration>2022-04-02T12:03:38</Expiration>
  <CreationDate>2014-04-17T12:04:18.9158349-07:00</CreationDate>
  <FileReference>WordPress Hardening Guide.docx</FileReference>
</ClassificationRules>
<ClassificationRules Level="UNCLASS" Compartment="RED ">
  <ComputerName>SFS-WORKSTATION</ComputerName>
  <UserName>Scott Amack</UserName>
  <String>quick style gallery</String>
  <Expiration>2022-04-02T12:03:38</Expiration>
  <CreationDate>2014-04-17T12:04:18.9158349-07:00</CreationDate>
  <FileReference>WordPress Hardening Guide.docx</FileReference>
</ClassificationRules>
<ClassificationRules Level="UNCLASS" Compartment="BLACK ">
  <ComputerName>SFS-WORKSTATION</ComputerName>
  <UserName>Scott Amack</UserName>
  <String>also format text directly</String>
  <Expiration>2020-04-05T12:05:13</Expiration>
  <CreationDate>2014-04-17T12:05:51.205597-07:00</CreationDate>
  <FileReference>Securing CentOS.docx</FileReference>
</ClassificationRules>
<ClassificationRules Level="SECRET" Compartment="RED ">

```

```

    <ComputerName>SFS-WORKSTATION</ComputerName>
    <UserName>Scott Amack</UserName>
    <String>also format text directly</String>
    <Expiration>2020-04-05T12:05:13</Expiration>
    <CreationDate>2014-04-17T12:05:51.205597-07:00</CreationDate>
    <FileReference>Securing CentOS.docx</FileReference>
  </ClassificationRules>
  <ClassificationRules Level="UNCLASS" Compartment="BLACK ">
    <ComputerName>SFS-WORKSTATION</ComputerName>
    <UserName>Scott Amack</UserName>
    <String>Styles gallery on the Home tab</String>
    <Expiration>2019-04-04T12:04:45</Expiration>
    <CreationDate>2014-04-17T12:05:13.0323299-07:00</CreationDate>
    <FileReference>How to set up VPN on server 2008.docx</FileReference>
  </ClassificationRules>
  <ClassificationRules Level="TOPSECRET" Compartment="GREEN ">
    <ComputerName>SFS-WORKSTATION</ComputerName>
    <UserName>Scott Amack</UserName>
    <String>Styles gallery on the Home tab</String>
    <Expiration>2019-04-04T12:04:45</Expiration>
    <CreationDate>2014-04-17T12:05:13.0323299-07:00</CreationDate>
    <FileReference>How to set up VPN on server 2008.docx</FileReference>
  </ClassificationRules>
  <ClassificationRules Level="TOPSECRET" Compartment="GREEN ">
    <ComputerName>SFS-WORKSTATION</ComputerName>
    <UserName>Scott Amack</UserName>
    <String>Styles gallery on the Home tab</String>
    <Expiration>2010-04-04T12:04:45</Expiration>
    <CreationDate>2014-04-17T12:05:13.0323299-07:00</CreationDate>
    <FileReference>How to set up VPN on server 2008.docx</FileReference>
  </ClassificationRules>
</ArrayOfClassificationRules>

```

Appendix H: Derivative Classification Automator Test

Logs

Error Log

Classification string to be checked is: to insert tables, headers, footers
Current Node classification level is: TOPSECRET
Rule classification level is: SECRET
Node compartments are: BLACK RED GREEN
Rule compartments are: BLACK RED GREEN
Current Date is: 4/19/2014
Rule Expiration Date is: 4/1/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is WordPress Hardening Guide.docx
Rule Values Did Not Dominate Current Paragraph Values

Classification string to be checked is: to insert tables, headers, footers
Current Node classification level is: TOPSECRET
Rule classification level is: UNCLASS
Node compartments are: BLACK RED GREEN
Rule compartments are: BLACK RED GREEN
Current Date is: 4/19/2014
Rule Expiration Date is: 4/1/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is WordPress Hardening Guide.docx
Rule Values Did Not Dominate Current Paragraph Values

Classification string to be checked is: to insert tables, headers, footers
Current Node classification level is: TOPSECRET
Rule classification level is: UNCLASS
Node compartments are: BLACK RED GREEN
Rule compartments are: BLACK RED
Current Date is: 4/19/2014
Rule Expiration Date is: 4/1/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is WordPress Hardening Guide.docx
Rule Values Did Not Dominate Current Paragraph Values

Classification string to be checked is: to insert tables, headers, footers
Current Node classification level is: TOPSECRET
Rule classification level is: UNCLASS
Node compartments are: BLACK RED GREEN
Rule compartments are: BLACK
Current Date is: 4/19/2014
Rule Expiration Date is: 4/1/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is WordPress Hardening Guide.docx
Rule Values Did Not Dominate Current Paragraph Values

Classification string to be checked is: to insert tables, headers, footers
Current Node classification level is: TOPSECRET
Rule classification level is: UNCLASS
Node compartments are: BLACK RED GREEN
Rule compartments are:

Current Date is: 4/19/2014
Rule Expiration Date is: 4/1/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is WordPress Hardening Guide.docx
Rule Values Did Not Dominate Current Paragraph Values

Classification string to be checked is: Styles gallery on the Home tab
Current Node classification level is: UNCLASS
Rule classification level is: TOPSECRET
Node compartments are: BLACK
Rule compartments are: GREEN
Current Date is: 4/19/2014
Rule Expiration Date is: 4/4/2019
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is How to set up VPN on server 2008.docx
Rule Values Did Not Dominate Current Paragraph Values

Classification string to be checked is: Styles gallery on the Home tab
Current Node classification level is: UNCLASS
Rule classification level is: TOPSECRET
Node compartments are: BLACK
Rule compartments are: GREEN
Current Date is: 4/19/2014
Rule Expiration Date is: 4/4/2010
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is How to set up VPN on server 2008.docx
Match found but the rule has expired

Classification string to be checked is: also format text directly
Current Node classification level is: UNCLASS
Rule classification level is: SECRET
Node compartments are: BLACK
Rule compartments are: RED
Current Date is: 4/19/2014
Rule Expiration Date is: 4/5/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is Securing CentOS.docx
Rule Values Did Not Dominate Current Paragraph Values

Classification string to be checked is: quick style gallery
Current Node classification level is: UNCLASS
Rule classification level is: UNCLASS
Node compartments are: BLACK
Rule compartments are: RED
Current Date is: 4/19/2014
Rule Expiration Date is: 4/2/2022
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is WordPress Hardening Guide.docx
Classification levels are the same but the compartments do not match.

Upgrade Log

Classification string to be checked is: to insert tables, headers, footers
Current Node classification level is: UNCLASS
Rule classification level is: UNCLASS
Node compartments are:
Rule compartments are:
Current Date is: 4/19/2014
Rule Expiration Date is: 4/1/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is WordPress Hardening Guide.docx
Node Upgraded

Classification string to be checked is: to insert tables, headers, footers
Current Node classification level is: SECRET
Rule classification level is: SECRET
Node compartments are:
Rule compartments are:
Current Date is: 4/19/2014
Rule Expiration Date is: 4/1/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is WordPress Hardening Guide.docx
Node Upgraded

Classification string to be checked is: to insert tables, headers, footers
Current Node classification level is: TOPSECRET
Rule classification level is: TOPSECRET
Node compartments are:
Rule compartments are:
Current Date is: 4/19/2014
Rule Expiration Date is: 4/1/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is WordPress Hardening Guide.docx
Node Upgraded

Classification string to be checked is: to insert tables, headers, footers
Current Node classification level is: TOPSECRET
Rule classification level is: TOPSECRET
Node compartments are: BLACK
Rule compartments are: BLACK
Current Date is: 4/19/2014
Rule Expiration Date is: 4/1/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is WordPress Hardening Guide.docx
Node Upgraded

Classification string to be checked is: to insert tables, headers, footers
Current Node classification level is: TOPSECRET
Rule classification level is: TOPSECRET
Node compartments are: BLACK RED
Rule compartments are: BLACK RED
Current Date is: 4/19/2014
Rule Expiration Date is: 4/1/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is WordPress Hardening Guide.docx
Node Upgraded

Classification string to be checked is: to insert tables, headers, footers
Current Node classification level is: TOPSECRET
Rule classification level is: TOPSECRET
Node compartments are: BLACK RED GREEN
Rule compartments are: BLACK RED GREEN
Current Date is: 4/19/2014
Rule Expiration Date is: 4/1/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is WordPress Hardening Guide.docx
Node Upgraded

Classification string to be checked is: Styles gallery on the Home tab
Current Node classification level is: UNCLASS
Rule classification level is: UNCLASS
Node compartments are: BLACK
Rule compartments are: BLACK
Current Date is: 4/19/2014
Rule Expiration Date is: 4/4/2019
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is How to set up VPN on server 2008.docx
Node Upgraded

Classification string to be checked is: also format text directly
Current Node classification level is: UNCLASS
Rule classification level is: UNCLASS
Node compartments are: BLACK
Rule compartments are: BLACK
Current Date is: 4/19/2014
Rule Expiration Date is: 4/5/2020
File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml
Reference document for this rule is Securing CentOS.docx
Node Upgraded

Classification string to be checked is: quick style gallery
Current Node classification level is: UNCLASS
Rule classification level is: UNCLASS
Node compartments are: BLACK
Rule compartments are: BLACK
Current Date is: 4/19/2014

Rule Expiration Date is: 4/2/2022

File containing xml rules is ..\..\DerivativeClassification\UnitTest.xml

Reference document for this rule is WordPress Hardening Guide.docx

Node Upgraded