**PROGRAMMING SECURE SOFTWARE SYSTEMS – ASSIGNMENT 2**

Sam Reid – 10284311

Alex Till – 10364452

20/05/2016

For Michael N. Johnstone

Table of contents

[Introduction 1](#_Toc451713534)

[Decision Tree’s 2](#_Toc451713535)

[Brute Force Pruning Algorithm 3](#_Toc451713536)

[Author Notes 3](#_Toc451713537)

[Minimax Decision Theory Concept With Alpha-Beta Pruning 4](#_Toc451713538)

[Author Notes 6](#_Toc451713539)

[Calculating Misuse Case Weighting 6](#_Toc451713540)

[Conclusion 7](#_Toc451713541)

[Program Requirements 9](#_Toc451713542)

[Design Artifact 10](#_Toc451713543)

[Test Cases 11](#_Toc451713544)

[References 14](#_Toc451713545)

# Introduction

An integral part of planning a software system is defining the functional requirements. Before development begins documentation is required that clearly represents what the system is meant to do and how a user interacts with it. Use cases encompass the requirements of a system and map the order of events that enable the system to achieve its goal (Bittner, 2002). The Unified Modelling Language (UML) is used to create models that visually display aspects of a system. UML is considered a standard modelling language used in systems development to identify and demonstrate the use cases of a software system (Quatrani & Evangelist, 2003). UML use case diagrams provide an effective method of modelling functional requirements of software systems while providing a simple representation of the systems functionalities for both the prospective user and developer (Shen & Liu, 2003).

While UML use cases are helpful for understanding the interactions between application users and the system, a need exists for a similar diagram that examines the threats and threat actors against the system. As the name suggests, misuse case diagrams are use case diagrams created from a malicious actor’s point of view; a misuser who create misuses. Misuse case diagrams interact with the use case diagram to demonstrate the security threats at each functional requirement (Alexander, 2003). They can be used as an effective means of modelling security requirements (Johnstone, 2011).

There exist many methodologies for creating misuses and identifying threats for a given actor and category of vulnerability. The “STRIDE” methodology consists of a matrix which binds six areas of security; Spoofing identity, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege (Johnstone, 2010). Using the STRIDE matrix it is possible to enhance a UML with threat modelling capability by creating misuse cases for a given threat actor, threat type and use case (Johnstone, 2010). However, not all the misuses created will make practical sense and intelligence is needed to examine the candidate misuse cases and determine their validity.

Accepting a use case diagram in XML format we propose a system that will generate a list of candidate misuse cases and then apply a pruning algorithm to produce a reduced list of valid misuse cases. By building a decision tree that defines what properties valid misuse cases often have we can prune our candidate misuse case dataset and ultimately automate the validation task of creating misuse cases. This report examines popular pruning algorithms, explaining how they operate and their feasibility in relation to pruning candidate misuse cases.

# Decision Tree’s

The program must have knowledge of what attributes make up a valid or invalid misuse case before it can effectively apply a pruning strategy. The use of decision trees to dictate what attributes are required for a valid entity and influence system decisions and is a popular machine learning technique (Quinlan, 1986). In visual representation, the tree diagrams begin at a root node and based on a series of decisions and their consequences span out to many leaf nodes. The size (number of nodes) is important to the accuracy of the decision tree and the rule of thumb in order to achieve an appropriate generalisation is that the tree should be the smallest form that will fit the data. Determining the smallest form of a decision tree is often a complex task as it usually isn’t obvious what the best size really is. The potential that adding another node to the tree will greatly skew the results for better or worse is difficult to overcome (Reed, 1993). Too few nodes and the tree misclassifies the production data it’s using against; however, too many nodes and the tree begins to contain unwanted or meaningless nodes, called “overfitting” (Patil, Wadhai, & Gokhale, 2010). The way to overcome overfitting is by algorithmically pruning nodes from the tree, reducing the set to the most efficient and accurate.

# Brute Force Pruning Algorithm

A potential solution to overfitting and hence invalid misuse cases is with the use of brute force pruning, also known as reduced error pruning. This method requires a “training set” which is data containing a series of “known good” correctly classified data and the corresponding attributes that led to that classification. This dataset is presented to the decision tree and the output is compared so that the tree can be checked whether it is correctly classifying data and what percentage of error occurred (Quinlan, 1986). The reduced error pruning algorithm generalises the tree by deleting rules in the decision tree and reassessing the error against the training set until further deletion results in significant decreases in classification accuracy (Fürnkranz & Widmer, 1994).

## Author Notes

Reduced error pruning has been proven to work well on some datasets. It does not come without limitations though, one being efficiency. Assuming that each decision rule is deleted or modified at least once before the tree is generalised the pruning loop must execute numerous times depending on the number of decision rules. With every rule needed to be assessed to determine its effect on the error, reduced error pruning becomes unfeasible on large decision trees (Fürnkranz & Widmer, 1994).

Additionally, and more importantly, to solve the problem posed by this assignment with reduced error pruning would involve constructing a training set. Investing the time to generate training data to prune a decision tree would involve solving the assignment task manually which defeats the purpose of the program.

# Minimax Decision Theory Concept With Alpha-Beta Pruning

The minimax decision theory concept is one of many designed for two player games where the artificial intelligence (AI) player must determine its best moves without having knowledge of what moves the other player will make. The minimax theory is to choose the path that grants minimal loss for the AI player given a worst case scenario. It plays to maximise the worst case payoff (Goodrich, 2006). In layman’s terminology it’s the path that might not guarantee the highest score but can guarantee the highest score if the other player makes perfect moves (Jones, n.d.). In the context of the assignment, minimax would algorithmically determine a misuse case with decent probability after taking into consideration that choosing a more probable misuse case could lead to a very improbable misuse case being included in the results.

Alpha-Beta pruning is an enhancement of the minimax theorem (Bundy & Wallen, 1984). It efficiently determines the best value for the decision trees root node by traversing the tree left to right and pruning any nodes that can no longer influence the value of the root node. It chooses to only explore a branch if there is a possibility of it producing a more superior result (Pearl, 1982).

The “Alpha” value is the best already explored option along the path to the root for the maximiser and “Beta” being the best already explored option along the path to the root for the minimiser. If faced with a (Beta < Alpha) situation then the algorithm will prune any further leaf nodes as choosing leafs from this node will result is poor results being classified as valid paths (Rich, n.d.).

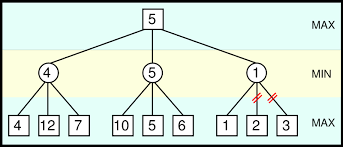


Figure 1: Alpha-Beta Pruning on minimax decision tree (Rich, n.d.)

Taking the example tree in figure 1, Alpha-Beta will traverse the tree beginning at the left node with 4, 12 and 7. The minimiser node (denoted by the circles) will examine the minimum value it can obtain from the leaf nodes, in this case 4, and pass it back up to the parent (the root node).

At this point we will have an Alpha value of 4 (current value of the root node) and a Beta value of “negative infinity” (there is no minimiser node between the root node and the path to the root node).

The centre node is then explored starting with the furthest left leaf containing the number 10. At this point the current maximiser (Alpha) is 4 and since 10 > 4 Alpha-Beta cannot prune because the result is superior (meaning this path is currently looking like a good choice). The minimiser inherits the 10 as it is the current smallest assessed value, so Beta becomes 10.

Continuing, it checks 5 > 4 (Alpha) and although it still cannot prune (because Beta (10) > (4) Alpha) the minimiser has found a better result (10 > 5) so the minimiser is set at 5. Lastly, it checks 6 > 4 (Alpha) and cannot prune. The minimiser concludes that 5 is the smallest value obtainable from this path. The result 5 is passed up to the parent node, a maximiser, which determines that 5 > 4 and hence the root node and new Alpha value becomes 5.

Alpha-Beta traverses the last node which calculates 1 < 5 (Alpha). The minimiser node sets 1 as Beta and Alpha-Beta concludes that this path is no longer worth traversing because the minimum (Beta) is 1 which is a worse result than the current maximiser (Alpha) 5 and hence Beta (1) < (5) Alpha – it prunes the rest of the leaf nodes attached to this branch.

Finally, minimax returns the root node value 5 as the best possible score obtainable (Pearl, 1982; Rich, n.d.).

## Author Notes

Minimax and Alpha-Beta define a game playing methodology which isn’t the most suited algorithm for pruning misuse cases as there is no second ‘player’ in our scenario. It is also designed to return only one “best path” for a given game situation, whereas for our program we wish to collect all non-pruned results. Having said that, with minor adaptations we anticipate that minimax and Alpha-Beta will effectively prune poorly weighted misuse cases from the resultant dataset in an intelligent way and provide meaningful results that align with the overall goal of the assignment.

# Calculating Misuse Case Weighting

In order to use our STRIDE generated misuse cases as leaf nodes in a decision tree we need to convert them into integer figures that can be parsed and analysed by minimax and Alpha-Beta. By examining industry published threat probability reports and comparing the results we can, with greater accuracy, define probability estimates of STRIDE derived misuse cases. The following reports were analysed in constructing the probability results table:

* Internal Vs. External Penetrations: A Computer Security Dilemma (Diaz-Gomez, ValleCarcamo, & Jones, 2010)
* Web Application Security Consortium Statistics ("Web Application Security Statistics ", 2009)
* WhiteHat Website Security Statistics Report (*Website Security Statistics Report*, 2015)
* SANS DDoS Survey (Pescatore, 2014)

Cyber threats are a rapidly growing and evolving field, it was important to compare findings from a range of publications that collect information from many different sources to provide the best overview of the current threat landscape. A “probability range” has been established to better represent the likelihood of attack and have been amalgamated with the STRIDE matrix (1 being rare and 100 being extremely likely):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| MUC Construct | S | T | R | I | D | E |
| Actor |  |  |  | 16-55 |  |  |
| Mis-Actor(External) | 26-47 | 48-99 |  | 17-64 | 38-99 |  |
| Mis-Actor(Internal) |  | 3-46 | 11-70 | 3-46 |  | 3-46 |

Table 1: STRIDE Probabilities

# Conclusion

The program will use the Alpha-Beta pruning algorithm with the minimax decision theory methodology. After the generation of candidate misuse cases tied to use cases the program will generate a probability value for each misuse case. These values will be derived from the probability ranges reported in the STRIDE probability table and subsequently will make up the nine leaf nodes of the decision tree. Admittedly, minimax is not the most appropriate decision theory given the assignment problem. A better decision theory would be one that could determine risks for a given use case and produce valid misuse cases through risk evaluation. However, a versatile decision theory that comprehends use cases would be very difficult to develop and given the timeframe for this assignment and knowledge of machine learning techniques, minimax enhanced with the Alpha-Beta pruning produces appropriate results. One minor adaptation to the minimax decision theory made for this assignment is after pruning instead of reporting the most viable use case (the root node) the program will instead return all non-pruned misuses which will become our set of valid misuses cases for the given use case.

# Program Requirements

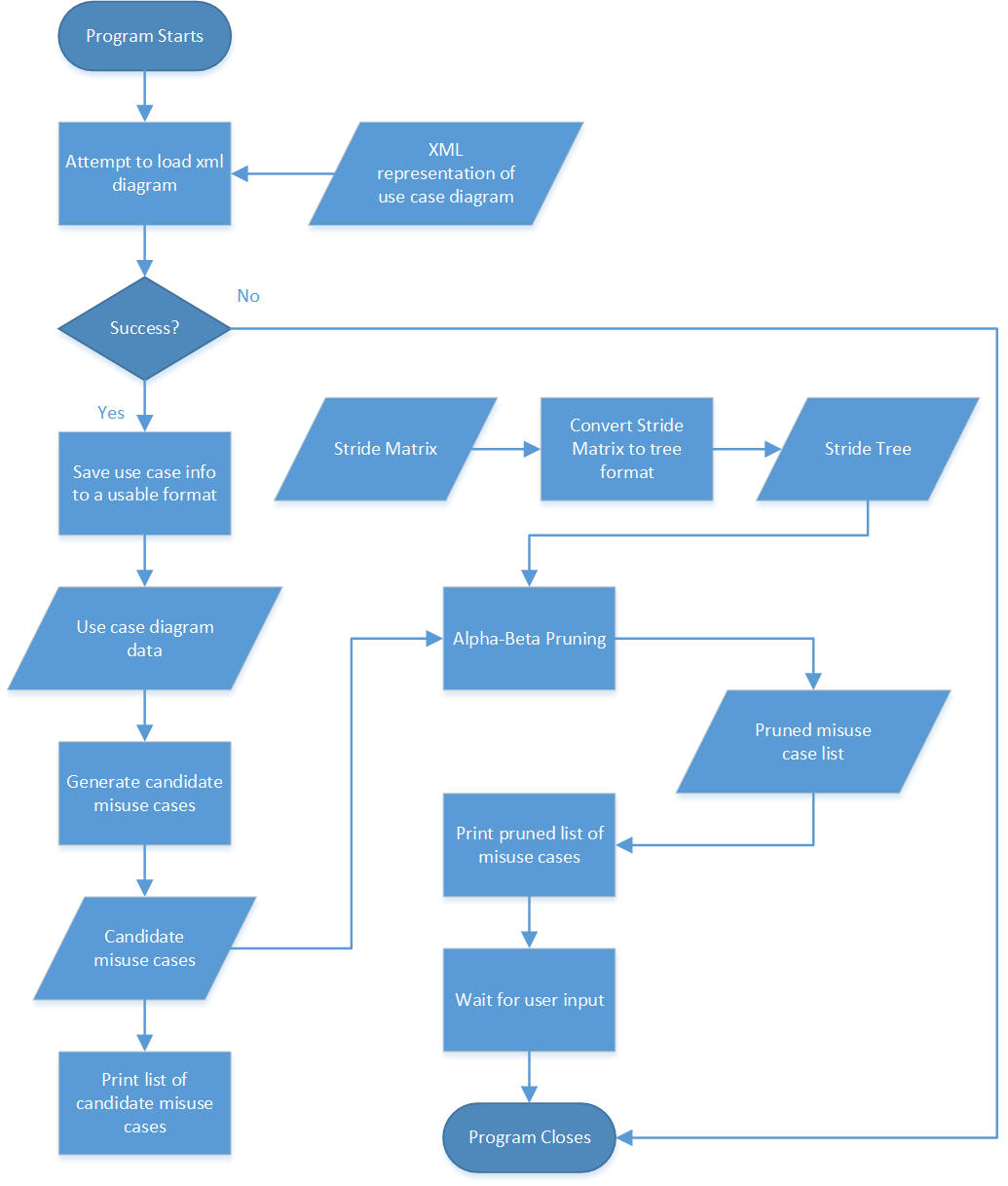
As a proof of concept, the program needs only a few key requirements:

* Read representation of case diagram in xml format
* Store STRIDE matrix in code
* Generate candidate misuse cases using STRIDE matrix
* List the candidate misuse cases
* Prune candidate misuse cases to remove invalid cases
* List misuse case and threatened node in console

**Non-functional:**

* The program will be written in C++

# Design Artifact



# Test Cases

**Application System:** Automatic Misuse Case Generation Tool

**Date:** 19/05/2016

**Tests Conducted By:** Alex Till

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test Description** | **Expected Behaviours/ Evaluation Criteria** | **Outcome** |
| Test-01 | Check that application correctly reads an XML representation of a use case diagram. | The application will be able to correctly read a representation of case diagram in xml format. | The XML file was read correctly and properly inserted into memory. |

**Test Passed?:** YES.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test Description** | **Expected Behaviours/ Evaluation Criteria** | **Outcome** |
| Test-02 | Check that the STRIDE matrix is stored in the program code. | The application will have a STRIDE matrix stored in the program code. | A STRIDE matrix is stored in the program code. |

**Test Passed?:** YES.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test Description** | **Expected Behaviours/ Evaluation Criteria** | **Outcome** |
| Test-03 | Check that the program generates candidate misuse cases for each use case. | Each use case will have 9 candidate misuse cases generated based on the STRIDE matrix. | 9 misuse cases were generated for each use case in the diagram. |

**Test Passed?:** YES.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test Description** | **Expected Behaviours/ Evaluation Criteria** | **Outcome** |
| Test-04 | Check that the program lists each misuse case and threatened node in the console. | The application will list all the candidate misuse cases. | The misuse cases were listed correctly. |

**Test Passed?:** YES.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test Description** | **Expected Behaviours/ Evaluation Criteria** | **Outcome** |
| Test-05 | Check that the pruning algorithm receives the candidate misuse cases. | After the candidate misuse cases are generated they will be sent to the pruning algorithm. | The candidate misuse cases were sent to the pruning algorithm correctly. |

**Test Passed?:** YES.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test Description** | **Expected Behaviours/ Evaluation Criteria** | **Outcome** |
| Test-06 | Check that the program prunes the candidate misuse cases correctly. | The program will prune the candidate misuse cases down so that all invalid ones are removed. | The program has pruned the candidate misuse cases and output the remaining ones. |

**Test Passed?:** YES.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test Description** | **Expected Behaviours/ Evaluation Criteria** | **Outcome** |
| Test-07 | Check that the program lists each misuse case and threatened node in the console. | The application will list all the misuse cases returned from the pruning algorithm. | The misuse cases were listed correctly. |

**Test Passed?:** YES.

# References

Alexander, I. (2003). Misuse cases: Use cases with hostile intent. *Software, IEEE, 20*(1), 58-66.

Bittner, K. (2002). *Use case modeling*: Addison-Wesley Longman Publishing Co., Inc.

Bundy, A., & Wallen, L. (1984). Alpha/Beta Pruning *Catalogue of Artificial Intelligence Tools* (pp. 4-4): Springer.

Diaz-Gomez, P. A., ValleCarcamo, G., & Jones, D. (2010). *Internal Vs. External Penetrations: A Computer Security Dilemma.* Paper presented at the Proceedings of the 2010 International Conference on Security & Management.

Fürnkranz, J., & Widmer, G. (1994). *Incremental reduced error pruning.* Paper presented at the Proceedings of the 11th International Conference on Machine Learning (ML-94).

Goodrich, M. (2006). Minimax and Alpha-Beta Pruning. Retrieved from <https://students.cs.byu.edu/~cs670ta/Lectures/Minimax.html>

Johnstone, M. N. (2010). Threat modelling with STRIDE and UML.

Johnstone, M. N. (2011). Modelling misuse cases as a means of capturing security requirements.

Jones, J. (n.d.). Decision Theory. Retrieved from <https://people.richland.edu/james/summer02/m160/decision.html>

Patil, D. D., Wadhai, V., & Gokhale, J. (2010). Evaluation of decision tree pruning algorithms for complexity and classification accuracy. *International Journal of Computer Applications (0975–8887) Volume*.

Pearl, J. (1982). The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. *Communications of the ACM, 25*(8), 559-564.

Pescatore, J. (2014). *SANS 2014 DDoS Survey*. Retrieved from <https://www.sans.org/reading-room/whitepapers/analyst/ddos-attacks-advancing-enduring-survey-34700>

Quatrani, T., & Evangelist, U. (2003). Introduction to the Unified modeling language. *A technical discussion of UML, 6*(11), 03.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning, 1*(1), 81-106.

Reed, R. (1993). Pruning algorithms-a survey. *Neural Networks, IEEE Transactions on, 4*(5), 740-747.

Rich, C. (n.d.). *Minimax Search*. Retrieved from <http://web.cs.wpi.edu/~rich/courses/imgd4000-d09/lectures/E-MiniMax.pdf>

Shen, W., & Liu, S. (2003). Formalization, testing and execution of a use case diagram *Formal Methods and Software Engineering* (pp. 68-85): Springer.

Web Application Security Statistics (2009). Retrieved from <https://webcache.googleusercontent.com/search?q=cache:ZMGffq2IiEsJ:projects.webappsec.org/Web-Application-Security-Statistics>

*Website Security Statistics Report*. (2015). Retrieved from <https://info.whitehatsec.com/rs/whitehatsecurity/images/2015-Stats-Report.pdf>