Please cite as BT Technology Journal 25 (2007) p118-127 DOI: 10.1007/s10550-007-0014-8

Security Risk Mitigation for Information Systems

Victor Page[1], Maurice Dixon[2], Islam Choudhury[2]

[1]Faculty of Computing Information Systems and Mathematics,
Kingston University,
Sopwith Building, Penryhn Road, Kingston upon Thames, Surrey, KT1 2EE, UK
v.page@kingston.ac.uk
[2]Department of Computing Communications Technology and Mathematics,
London Metropolitan University,
31 Jewry Street, London, EC3N 2EY, UK
{m.dixon, i.choudhury}@londonmet.ac.uk

Abstract **– Security risk mitigation is a salient issue in systems development research. This paper introduces a light weight approach to security risk mitigation, that can be used within an Agile Development framework, the Security Obstacle Mitigation Model (SOMM). The SOMM uses the concept of Trust Assumptions to derive obstacles and the concept of Misuse Cases to model the obstacles. A synthetic scenario, based on an on-line system, shows how the SOMM is used to anticipate malicious behaviour with respect to an operational Information System and to document a priori how this malicious behaviour should be mitigated. Since the SOMM is conceptually simple in deployment, its use is well within the capacities of the users who form part of an Agile Development team and crucially it should not take up a significant amount of development time.**

1 Introduction

Security risk mitigation is concerned with developing trustworthy Information Systems that uphold the security requirements of Confidentiality, Integrity, Availability, and Authentication as presented in [21] and given the acronym CIAA. Goal-Oriented requirements engineering is concerned with the precise specification of software behaviour. Within goal-oriented requirements engineering Potts [19] introduced the term obstacle to describe something that would obstruct a goal from being achieved. In this spirit we describe security risk mitigation for the development of trustworthy Information Systems that are tolerant to obstacles that may obstruct the CIAA security requirements.

In [8] Fickas and Feather suggested that development teams should record assumptions made about requirements, along with amendments or extensions that would mitigate the problems caused by the assumptions that become invalid. In [14] Lamsweerde, Letier and Ponsard suggest that requirements assumptions that become invalid can cause obstacles that would obstruct a goal from being achieved. Trust Assumptions are concerned with developing trustworthy Information Systems. They are assumptions made by the development team that a Use Case, when realised in an operational Information System, will have certain stated properties and/or behaviour, in order to satisfy one or more of the CIAA security requirements [18, 23]. We can therefore state that obstacles to the CIAA security requirements can be caused when Trust Assumptions become invalid i.e. they are obstructed by an obstacle.

In [4] Dewar introduces Assumption Based Planning as a post-planning technique that is used to contain the risk to a plan caused by assumptions about that plan that become invalid. In Assumption Based Planning assumptions that would have severe consequences should they become invalid are called *load-bearing assumptions* while

assumptions that are most likely to become invalid are called *vulnerable assumptions*. Assumptions that are both load-bearing and vulnerable must be mitigated in some way. We adopt this terminology and suggest that load-bearing vulnerable Trust Assumptions must be protected from the obstacles that obstruct them; specifically the obstacles must be mitigated in some way.
In [20] Sindre and Opdhal state that significant requirements can be missed during Use Case analysis due to analysts making unjustified or naive assumptions about the problem domain. They explain that one area where this could cause problems is in the elicitation of security requirements. In order to counter this problem they introduce the concept of a Misuse Case. A Misuse Case captures unwanted behaviour and allows the development team to reason about security threats. We use the concept of Misuse Cases to model obstacles that will obstruct the load-bearing vulnerable Trust Assumptions.
This present work uses the above concepts to form the Security Obstacle Mitigation Model (SOMM). Specifically the SOMM provides a light weight approach for developing trustworthy Information Systems that preserve the CIAA security requirements. This is achieved by mitigating obstacles that will obstruct load-bearing vulnerable Trust Assumptions. The underlying principles used to guide the development of the SOMM are borrowed from our previous work on the Trust Obstacle Mitigation Model [17, 18].
The rest of this paper is structured as follows. Section 2 provides an overview of the SOMM. Section 3 demonstrates the application of the SOMM for a synthetic scenario based on an on-line student grades system. Section 4 overviews related work. Finally conclusions are presented in Section 5 along with proposals for future work

2 Overview of the SOMM

Section 2.1 presents the definitions and terminology that are used within the SOMM. This is followed in section 2.2 by an Activity Diagram that shows how the SOMM is used. Section 2.3 presents the abstract syntax of the SOMM in the form of a meta-model. The meta-model is used in section 2.4 to define extensions to the UML, via the use of Stereotypes, in order for it to incorporate the vocabulary of the SOMM.

2.1 Concepts and Terminology

Listed alphabetically in the following are the definitions of the concepts and terminology that are used within the SOMM:

- **Mitigation**: A mitigation is a procedure that will counter the effect of an obstacle.
- **Mitigation Case**: A Use Case that shows what should be done to counter the effect of an Obstacle Case.
- **Obstacle**: An obstacle is something that, should it occur, will obstruct a trust assumption and affect the CIAA security requirements. Obstacles are caused by malicious or inadvertent use of the operational Information System.
- **Obstacle Case**: A Use Case that shows misuse (malicious or inadvertent) of the operational Information System that would obstruct a Trust Assumption.
- **RAG Code:** RAG Codes form an intuitive 'traffic light' approach to ranking the ability for load-bearing and the vulnerability of a Trust Assumption. RAG is an acronym for Red, Amber, and Green:
    - **R:** signifies stop and mitigate; issues associated with this rank will have associated obstacles that can cause significant problems with the CIAA security requirements.

- **A:** signifies proceed with caution; issues associated with this rank will have associated obstacles that will cause undesirable problems with the CIAA security requirements.
- **G:** signifies continue with trust; issues associated with this rank are insignificant and can be ignored.

Trust Assumptions that have a (R,R) load-bearing/vulnerability pair ranking are classified as load-bearing vulnerable Trust Assumptions whose obstacles must be mitigated. Some projects may require that additional Trust Assumptions should also be addressed. To facilitate this, the SOMM uses a load-bearing/vulnerability matrix (Figure 1) to combine the RAG Codes. This approach is borrowed from risk analysis and is generally known as a probability impact grid, or summary risk profile [11].

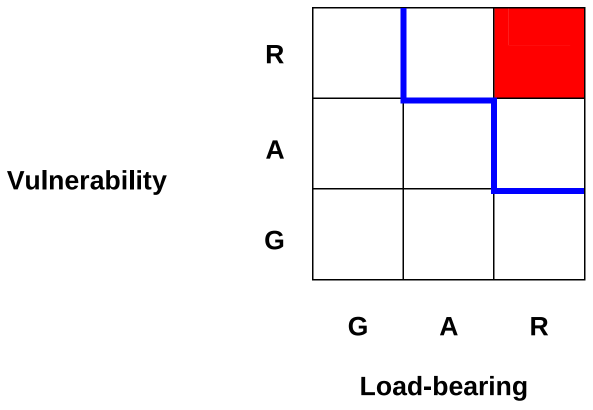

Figure 1 Load-bearing/Vulnerability Matrix

The filled square shows load-bearing vulnerable Trust Assumptions whose obstacles must be mitigated. The thick line is a tolerance line that is decided on a per project basis. For a specific project Trust Assumptions that fall above and to the right of the tolerance line are classed as significant and may also be required to have their obstacles mitigated.

- **Trust Assumption**: Documents the way in which a Use Case, when realised in the operational Information System, can be trusted to have certain stated properties and/or behaviour.
- **Use Case**: A representation by diagram and text of a sub-set of the Information System functionality.

2.2 Activity Diagram

Figure 2 presents an Activity Diagram for the SOMM. The phases and activities of the SOMM allow the development team to anticipate malicious behaviour with respect to the operational Information System and to document a priori how this malicious behaviour should be mitigated.

The swim lanes show the three phases of the SOMM: Elicit Use Cases, Elicit Obstacle Cases, and Elicit Mitigation Cases. Each phase could be carried out in either a separate Facilitated Workshop, or in a particular stage of an Agile Development framework. Each phase of the SOMM has its own unique set of activities. They do not form a strictly sequential process and there is an implied iteration in some of the activities. Specifically the development team may wish to re-prioritise the Use Case list either at the end of the Elicit Obstacle Cases phase, or at the end of the Elicit Mitigation Cases phase.

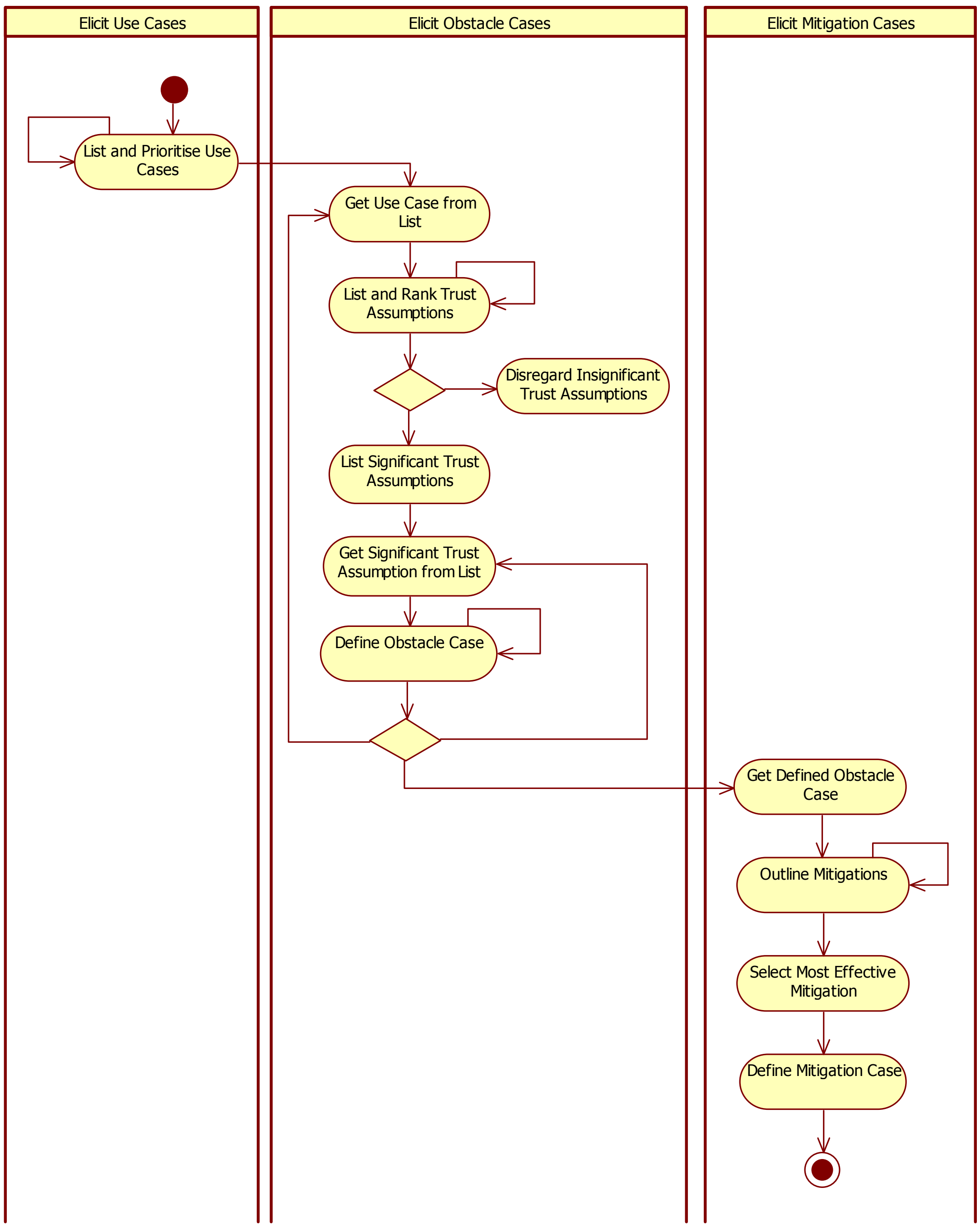
Elicit Use Cases
Elicit Obstacle Cases
Elicit Mitigation Cases
List and Prioritise Use Cases
Get Use Case from List
List and Rank Trust Assumptions
Disregard Insignificant Trust Assumptions
List Significant Trust Assumptions
Get Significant Trust Assumption from List
Define Obstacle Case
Get Defined Obstacle Case
Outline Mitigations
Select Most Effective Mitigation
Define Mitigation Case

Figure 2 Activity Diagram of the SOMM

The activities of the SOMM are defined as follows:

- **List and Prioritise Use Cases**: It is assumed that the initial Use Cases for the Information System will be elicited and prioritised in a Facilitated Workshop.
- **Get Use Case from List:** Get the first/next Use Case from the elicited list.
- **List and Rank Trust Assumptions**: The purpose of this activity is to list the Trust Assumptions we hold about a particular Use Case. The Trust Assumptions are given a load-bearing rank and a vulnerability rank. RAG Codes are used for the ranking process.

- **Disregard Insignificant Trust Assumptions**: Based on the load-bearing/vulnerability RAG Code pair in the load-bearing/vulnerability matrix, those Trust Assumptions that lie below the tolerance line are disregarded.
- **List Significant Trust Assumptions**: Based on the load-bearing/vulnerability RAG Code pair in the load-bearing/vulnerability matrix, those Trust Assumptions that lie above and to the right of the tolerance line are listed.
- **Get Significant Trust Assumption from List**: Get the first/next significant Trust Assumption from the list.
- **Define Obstacle Case:** An Obstacle Case is defined for the significant Trust Assumption. In some cases more than one Obstacle Case may be associated with a particular Trust Assumption.
- **Get Defined Obstacle Case**: Get the first/next Obstacle Case to mitigate.
- **Outline Mitigations:** For the Obstacle Case a possible list of mitigations is derived. The options at this point are to completely mitigate the obstacle, partially mitigate the obstacle, or monitor the obstacle. The estimated cost and duration of each of the mitigations are also recorded at this stage along with the particular benefits to the operational Information System.
- **Select Most Effective Mitigation:** Each of the mitigations for a particular obstacle is considered and the most effective mitigation, or possibly the most effective hybrid mitigation, is selected. Here the development team could prototype and evaluate the mitigation approaches in order to help in selecting the most effective mitigation and strike the right balance between cost and value.
- **Define Mitigation Case**: A Mitigation Case is defined for the most effective mitigation.

2.3 Meta-Model

Figure 3 presents the meta-model for the SOMM.

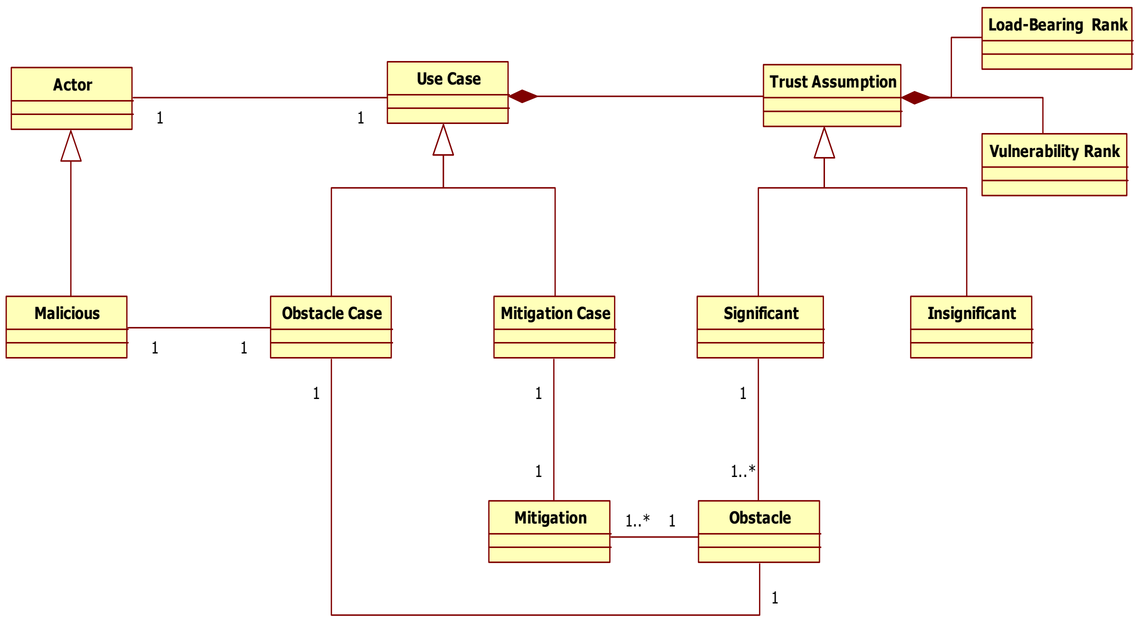

Figure 3 Meta-Model of the SOMM

The SOMM is based on Use Cases and Trust Assumptions, which we extend in several directions. Use Case is extended to incorporate the concepts of obstacle (from misuse) and mitigation. Also Actor is extended to include the concept of a malicious actor. Trust Assumptions are given a load-bearing rank and a vulnerability rank which are used to classify them as significant or insignificant. Significant Trust Assumptions are related explicitly to obstacles and obstacles are related to mitigations. Each obstacle is related to an Obstacle Case and each mitigation is related to a Mitigation Case.

2.4 UML Stereotypes for Security Obstacle Mitigation

In order for the meta-model of the SOMM to be used effectively with the UML, we need to extend the UML to incorporate the vocabulary of the SOMM within Use Cases and Class Diagrams. The extensions to the UML are via the use of Stereotypes and are given in Table 1.

Table 1 UML Stereotypes for Security Obstacle Mitigation

Stereotype	Base Class	Description
<<ObstacleCase>>	Use Case	An action that would obstruct one or more of the CIAA security requirements.
<<MitigationCase>>	Use Case	A mitigation that would stop or ameliorate one or more of the CIAA security requirements from being obstructed by an Obstacle Case.
<<MaliciousActor>>	Use Case (Actor)	An actor that would instigate an Obstacle Case.
<<Mitigates>>	Use Case (Relation)	Shows that a mitigation case mitigates a particular Obstacle Case.
<<ObstacleTo>>	Use Case (Relation)	Shows that an Obstacle Case is an obstacle to a particular Use Case.
<<MitigationClass>>	Class	A class that mitigates a particular obstacle.
<<MitigationAttribute>>	Attribute	An attribute in a class that is used to mitigate a particular obstacle.
<<MitigationMethod>>	Method	A method in a class that is used to mitigate a particular obstacle.

In the next section typical use of the Stereotypes associated with Use Cases can be found in Figures 5 and 6. Figure 8 shows typical use of the Stereotypes associated with Class Diagrams.

3 Example of Applying the SOMM

Section 3.1 provides an overview and Use Case Diagram for a Student Grades scenario. For clarity in illustrating the principle deployment of the SOMM, the scenario does not address all aspects of security that are related to such a system. For example there are no Trust Assumptions relating to the way in which user identities or passwords should be managed. Section 3.2 shows how the SOMM is applied to the scenario.

3.1 Scenario

The SOMM is used to design an Information System that is secure while tolerant to security obstacles. The primary aim of the system to be developed is to provide a Web-based interface to an existing Information System. At present lecturers pass paper copies of student grades to operatives who enter the student grades on the system. The grades are then printed out by the operatives and placed on course notice boards. The Web-based interface should provide the following functionality:

- lecturers will be able to either enter, or update or view student grades;
- lecturers will be able to view grades for a particular student or group of students;
- students will be able to view their own grades for all modules on the course they are taking.

The following assumptions were made with respect to the web system being developed:

- passwords will be secure;
- student records can only be maintained by a lecturer;
- the system will be available during normal working hours.

No specific design decisions were made with respect to the assumptions above. However a load-bearing/vulnerability matrix was produced for this project that has a tolerance line like that in Figure 1. The Use Case Diagram for the new system is shown in Figure 4.

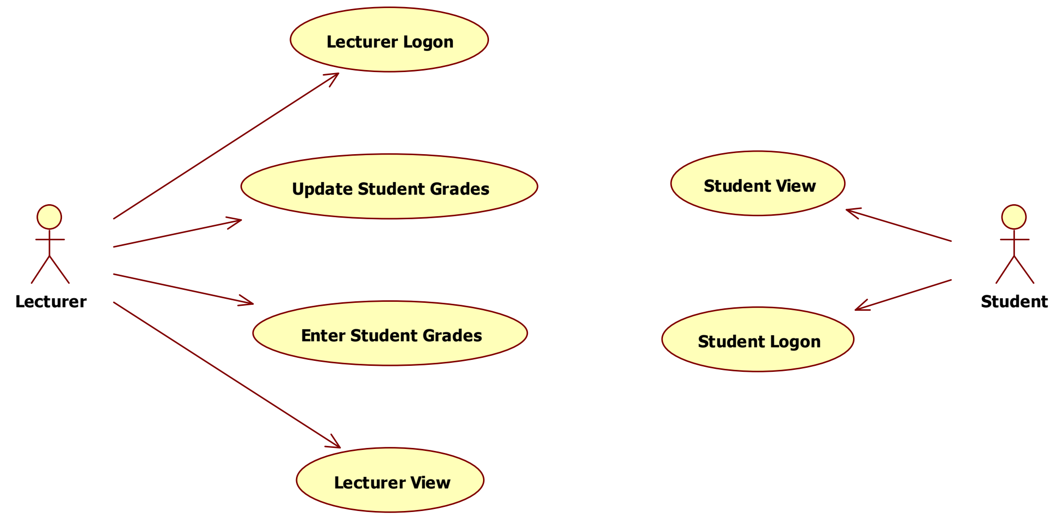

Figure 4 Student Grade System Use Case Diagram

Lecturer Logon and Student Logon are shown separately in Figure 4 as the access paths for each type of user will be different.

3.2 Applying the SOMM to the Scenario

The Trust Assumptions are an expansion of the assumptions made in the scenario. Use Cases and their associated Trust Assumptions are shown in Table 2

Table 2 Use Cases and Trust Assumptions

Use Case	Trust Assumption
Lecturer Logon	Lecturer password and user identity will not be revealed
Enter Student Grades	Student records will only be entered by a lecturer who is an authorised user
Update Student Grades	Student records will only be updated by a lecturer who is an authorised user Student records will only be deleted by a lecturer who is an authorised user
Lecturer View	Student records will be revealed to a lecturer only if he/she is an authorised user
Student Logon	Student password and user identity will not be revealed
Student view	A Student's records will be revealed to that student alone and then only if he/she is an authorised user

There is one general Trust Assumption - system use will not be blocked during normal operational times. Trust Assumptions along with their load-bearing and vulnerability RAG Codes are shown in Table 3.

Table 3 Trust Assumptions and RAG Codes

Trust Assumption	Load-Bearing RAG Code	Vulnerability RAG Code
1. Lecturer password and user identity will not be revealed	R	R
2. Student records will only be entered by a lecturer who is an authorised user	R	R
3. Student records will only be updated by a lecturer who is an authorised user	R	R
4. Student records will only be deleted by a lecturer who is an authorised user	R	R
5. Student records will be revealed to a lecturer only if he/she is an authorised user	G	A
6. Student password and user identity will not be revealed	G	A
7. A Student's records will be revealed to that student alone and then only if he/she is an authorised user	G	A
8. System use will not be blocked during normal operational times	G	A

Trust Assumptions 5 and 7 were given a load-bearing RAG Code of G, because we considered the revelation of a grade as non-critical, since grades are commonly displayed on course notice boards. Trust Assumption 6 was given a load-bearing RAG Code of G because a student can only look up grades, but can not change or delete them.

Trust Assumption 8 was given a load-bearing RAG Code of G, because we judged that blocking system use would only be a problem when departments were preparing for assessment boards; work could continue with a minimum of disruption by using the underlying system as previously intended.

The vulnerability RAG Codes for Trust Assumptions 5-8 are set to A, because although the Trust Assumptions are not load bearing they are still vulnerable to being obstructed. Trust Assumptions 1-4 have a load-bearing/vulnerability RAG Code pair of (R, R) due to the seriousness of student grades being changed.

After considering the tolerance line on the load-bearing/vulnerability matrix for this project it is clear that Trust Assumptions 1-4 require Obstacle Cases to be developed for them because they have a RAG Code pair of (R,R). Trust Assumptions 5-8 can be classed as insignificant and disregarded. Obviously any Trust Assumptions, should they exist, with a RAG Code pair of (A,R) or (R,A) would also require Obstacle Cases to be developed for them. Table 4 shows Trust Assumptions 1-4 along with their potential obstacles.

Table 4 Trust Assumptions and Obstacles

Trust Assumption	Obstacle
Lecturer password and user identity will not be revealed	Revealed by a brute-force password attack Revealed by a traffic analysis attack
Student records will only be entered by a lecturer who is an authorised user	Subverted by a masquerade attack
Student records will only be updated by a lecturer who is an authorised user	Subverted by a masquerade attack
Student records will only be deleted by a lecturer who is an authorised user	Subverted by a masquerade attack

From Table 4 we can define the following Obstacle Cases:

- The obstacle Masquerade Attack is an obstacle to Update Student Grades and Enter Student Grades.
- The obstacle Traffic Analysis Attack is an obstacle to Lecturer Logon.
- The obstacle Brute-force Password Attack is an obstacle to Lecturer Logon.

We assume that an Attacker, an external agent with malicious intentions, instigates all three of the attacks. Figure 5 shows a subset of the Use Case Diagram for the Student Grades system for those Use Cases that have Obstacle Cases associated with them.

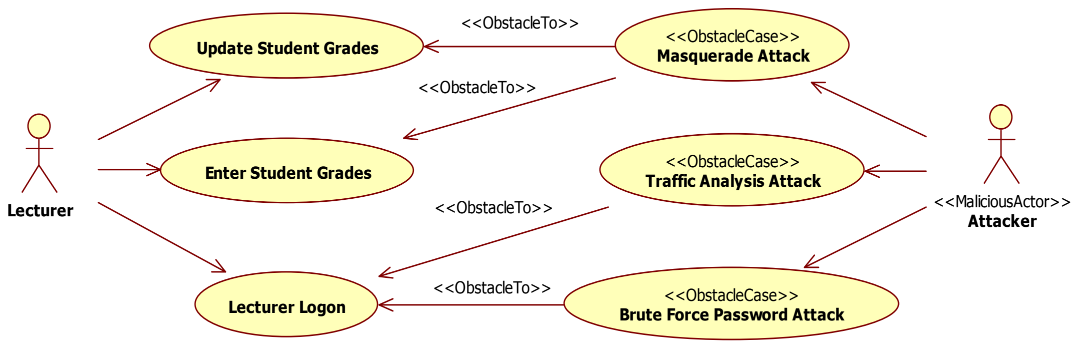
Update Student Grades
<<ObstacleTo>>
<<ObstacleCase>> Masquerade Attack
<<ObstacleTo>>
Lecturer
Enter Student Grades
<<ObstacleCase>> Traffic Analysis Attack
<<MaliciousActor>> Attacker
<<ObstacleTo>>
<<ObstacleTo>>
Lecturer Logon
<<ObstacleCase>> Brute Force Password Attack

Figure 5 Student Grades System with Associated Obstacle Cases

The mitigations for the Obstacle Cases are shown in Table 5.

Table 5 Obstacle Cases and Mitigations

Obstacle Case	Mitigation
Brute-force Password Attack	Monitor access attempts
Masquerade Attack	Use public key encryption Identify location
Traffic Analysis Attack	Use a firewall

The Traffic Analysis Attack and the Brute-Force Password Attack both have a single mitigation associated with them. However the Masquerade Attack has two candidate mitigations associated with it. We decided to use the identify location mitigation because public-key encryption would be too costly to implement and also identifying location would be complimented by monitor access attempts. The Mitigation Cases can be defined as follows:

- Identify Location
- Use a Firewall
- Monitor Access Attempts

Figure 6 shows a subset of the Use Case Diagram for the Student Grades system with the Obstacle Cases and Mitigation Cases included.

This section has shown that the SOMM aids Agile Developers by helping them focus on mitigating malicious use of an operational Information System.

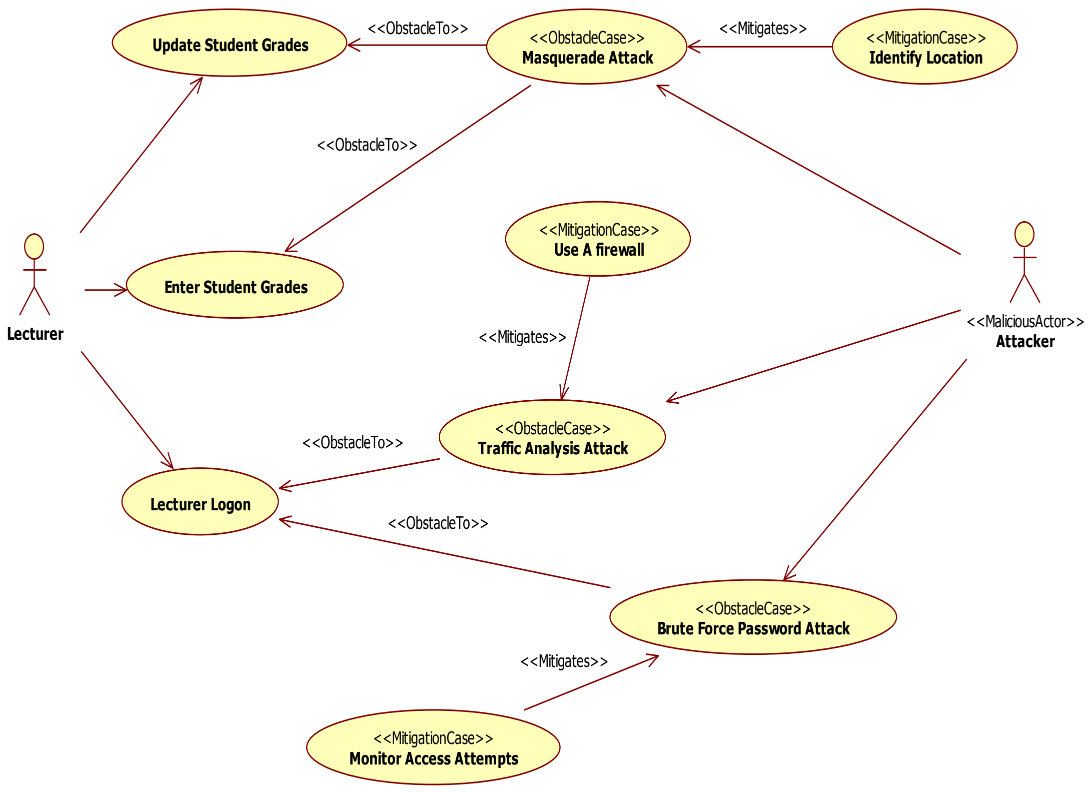

Figure 6 Student Grades System with Associated Obstacle Cases and Mitigation Cases

In order for the Agile Development team to further understand how each of the mitigations should be implemented we introduce an approach that has been used successfully by a significant number of our postgraduate students. It involves iteratively developing a simple paper based Sequence Diagram for each {Use Case, Obstacle Case, Mitigation Case} triple. This helps the development team to incorporate the mitigations in a baselined Class Diagram ready for further enhancements via prototyping. Figure 7 shows the output from using this approach on the {Update Student Grades, Masquerade Attack, Identify Location} triple.

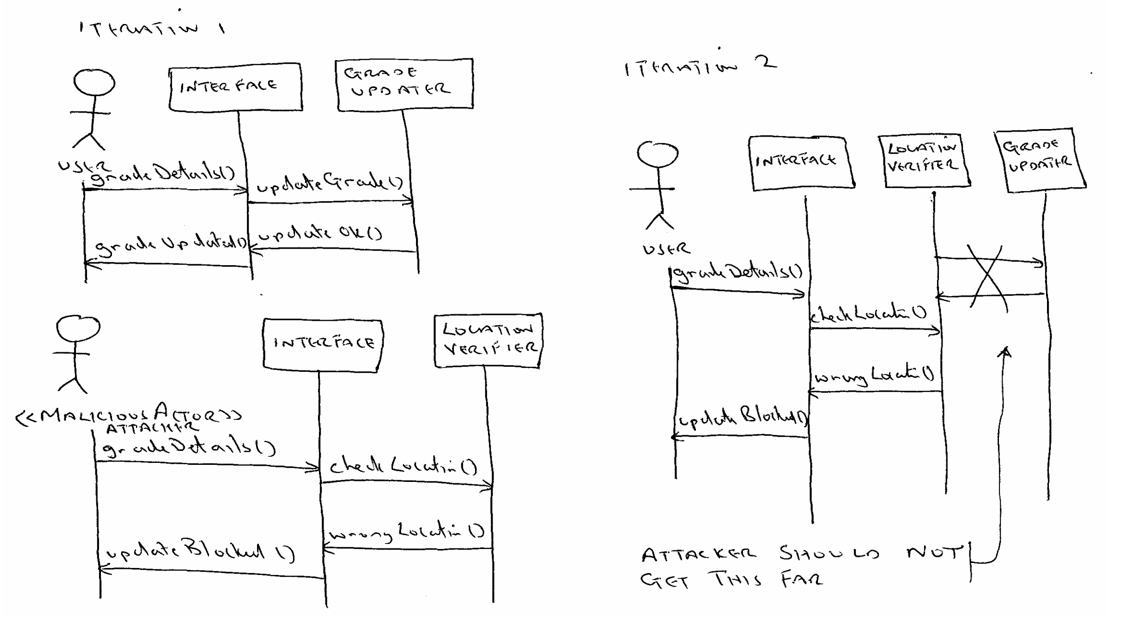

Figure 7 Iterative Paper Based Sequence Diagram

The first iteration in Figure 7 shows two Sequence Diagrams:

1. In the first Sequence Diagram a normal user is attempting to update a student grade. Grade change details are sent to the Grade Updater via an interface. The grade is updated and a grade updated message is sent to the user via an interface.
2. In the second Sequence Diagram an attacker is attempting to update a student grade. Location details of the attacker (in the form of an IP address) are sent to the Location Verifier via an interface. The location is checked and found to be a wrong location. The interface blocks the student grade update from taking place.

The second iteration in Figure 7 merges the normal use Sequence Diagram and the attack Sequence Diagram in order for the location of a user to be checked before they are allowed to update a student grade. The attacker has been removed, but the mitigation remains. This is because malicious actors and their associated Obstacle Cases need only be present in the system documentation up to the point where the development team have a grasp of how the mitigation should be implemented. In Agile Development 'just enough' work is completed in order for the development team to move on to the next stage of development. The Sequence Diagrams in Figure 7 provide a stepping stone from Use Case Diagrams to Class Diagrams. The Class Diagrams do not need to be complete to allow mitigations to be explored. One of the purposes of prototyping is to uncover missing classes, methods, attributes and relations. Another is to address issues like how the data is stored and how to protect the data at rest.

Figure 8 shows how the Stereotypes introduced in Table 1 should be used to model a mitigation at class level. The sole purpose of the Location Verifier class is to mitigate a Masquerade Attack. So the <<MitigationClass>> Stereotype is used to model this. Therefore there is no need to assert either the <<MitigationMethod>> or the <<MitigationAttribute>> in this case since the methods and attributes are implicitly defaulted to be mitigation terms.

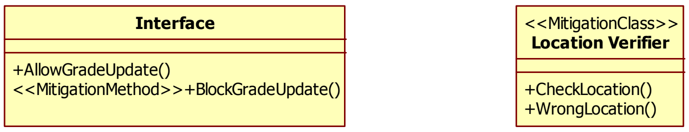

Figure 8 Using the SOMM Class Stereotypes

In contrast the Interface class is not solely in place to mitigate an obstacle. Only the method BlockGradeUpdate plays a part in mitigating the Masquerade Attack. So the Stereotype <<MitigationMethod>> is used to model this. A similar argument would apply for the use of the <<MitigationAttribute>> Stereotype.

These Stereotypes will show the development team and more importantly the maintenance team where and how mitigations are being realised.

5 Related Work

Our work takes existing approaches to security requirements based on Trust Assumptions and Obstacle Analysis and adapts them for use in an Agile Development environment. In [9] Trust Assumptions are used to analyse the domains of a system from a security perspective. An approach to structuring security arguments, including the use of rebuttals and mitigations is added in [10]. The idea is to carefully document assumptions about the extent to which domains can be trusted. We build on this work, but use a more lightweight approach, in order to facilitate the use of Trust Assumptions in an Agile Development environment. We note that an approach such as that in [10], that includes the use of formal reasoning and domain analysis, is likely to be better suited to large, safety critical systems.

In [13], a goal-based approach to requirements engineering is extended to deal with obstacles to a system meeting its requirements. In [12] the work is extended and applied to security, by modelling threats to security (malicious obstacles) as anti-goals. Security requirements are added to address such threats. Whilst that work seems amenable to automation through the use of a library of security patterns, goal based approaches have yet to be adapted to Agile Development.

Several approaches to addressing security at the requirements stage extend the Use Case approach. These include Misuse Cases [1, 20] and Abuse Cases [15]. The basic idea in both cases is to model an attacker's intentions as anti-requirements. This approach is extended in [6] by modelling misuse in Collaboration Diagrams. Coupled with this is the concept that UML diagrams can be represented as attributed typed graphs that can have positive or negative graphical constraints. Negative graphical constraints are similar to obstacles.

The CORAS methodology [22] uses an extended version of the UML for a modelling language and provides a heavyweight approach to risk analysis for security critical systems. The methodology integrates and further develops several significant techniques from the risk engineering domain. We take a similar but more light weight approach and focus on Obstacle Analysis as opposed to Risk Analysis.

The Model Driven Security approach [2] is grounded in formal logic. It defines a security language based on Role Based Access Control (RBAC) [7]. A meta-model for the language is used to extend the UML via the use of Stereotypes. Together they are used to provide a model driven architecture for developing secure systems. We take a similar approach in as much as we have developed a meta-model for the SOMM and used it to extend the UML via the use of Stereotypes.

6 Conclusions and Future Work

This work has shown that the SOMM aids Agile Development teams to focus on the detection and mitigation of security obstacles which would block the CIAA security requirements. Sections 2 and 3 showed that the detection and mitigation of security obstacles need not be complex tasks with the SOMM nor require complex technical skills and also crucially would not take up a lot of development time. This is because Use Cases are used to focus the development team on what and, at a high level of abstraction, how an obstacle should be mitigated. Also the iterative development of simple hand drawn Sequence Diagrams gives the development team an understanding of the underlying classes associated with the mitigation and a grasp of how the mitigation should be implemented. Both techniques provide a simple yet powerful diagrammatic representation of security obstacles and their mitigations at a level of granularity appropriate for reasoning about them in Facilitated Workshops and/or moving to the prototyping stage in an Agile Development project.

RAG Codes are used to rank Trust Assumptions with respect to two factors: that of being load-bearing and that of being vulnerable. As RAG Codes are qualitative, the two factors can not be multiplied. We have introduced the concept of a load-bearing/vulnerability matrix to express the combination of the two factors. This is summarised by the inclusion of a tolerance line which partitions the matrix. The top right RAG Code pairs above the tolerance line are classed as significant and RAG Code pairs below the tolerance line are classed as insignificant. The strength of this approach is that the development team can decide where the tolerance line sits in the matrix. Once this decision has been made the development team need only focus on ranking the load-bearing and vulnerability factors for each Trust Assumption. The combination of the RAG Codes is ‘automatic’ and fast.

Future work will focus on an Action Research project. The underpinning epistemology of Action Research is interpretive [16]. It merges research and practice to produce research findings that are relevant to academia and industry [3]. This is important for research in Information Systems as the end result of the research is aimed at being of use to industry while satisfying the rigour required by academia. The Action Research project will be used for two purposes:

1. to evaluate the use of the SOMM within the Dynamic Systems Development Method (DSDM) [5];
2. to extend the concepts within the SOMM to cover the complete software development lifecycle.

In the Action Research project we will be working closely with the development team and the Workshop Facilitator to evaluate the use of existing security techniques to elicit Trust Assumptions within the SOMM. This will include but will not be limited to security reviews, self assessment questionnaires, and the documentation of Trust Assumptions associated with any underlying systems. The aim will be to use and/or adapt the techniques to provide the right mix of speed of development and adequate protection. The Action Research project will also provide us with an opportunity to evaluate the SOMM against other approaches such as threat trees.

Acknowledgements

The authors wish to thank Robin Laney and Charles Haley, from the Open University, for the insights that they provided about Trust Assumptions. We are grateful to Stephen Cook from Reading University for introducing us to Assumption-Based Planning. Peter Bielkowicz of London Metropolitan University is thanked for useful discussions.

References

1. Alexander, I. (2003): Misuse Cases: Use Cases with Hostile Intent, IEEE Software, Vol. 20, No. 1. pp. 58-66.
2. Basin, D., Doser, J., and Lodderstedt, T. (2006): Model Driven Security: From UML models to access control infrastructures, ACM Transactions on Software Engineering Methodolgy, Vol. 15, No. 1, pp. 39-91.
3. Baskerville, R., and Wood-Harper, T. (1996): A critical perspective on action research as a method for information systems research, Journal of Information Technology, Vol. 11, pp 235-246.
4. Dewar, J. (2002): Assumption-Based Planning: A Tool for Reducing Avoidable Surprises. Cambridge University Press, ISBN 0 521 001269.
5. DSDM Version 4.2, (2005): (www.dsdm.org) Available: http://www.dsdm.org/ (Accessed: 200, December 2).
6. Dwaikat, Z., and Parisi-Presicce, F. (2004): From Misuse Cases to Collaboration Diagrams, in UML, Proceedings of the 3rd International Workshop on Critical System Development with UML, pp.130-138.
7. Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D., and Chandramouli, R.(2001): Proposed NIST standard for role-based access control, ACM Transactions on Information and System Security, Vol. 4, No. 3, pp. 224–274.
8. Fickas, S., and Feather, M. (1995): Requirements Monitoring in Dynamic Environments, Proceedings of the 2nd IEEE International Symposium on Requirements Engineering, pp. 140-147.
9. Haley, C., Laney, R., Moffett, J., Nuseibeh, B.: (2004), The Effect of Trust Assumptions on the Elaboration of Security Requirements, Proceedings of the 12th International Requirements Engineering Conference, pp. 102-111.

10. Haley, C., Moffett, J., Laney, R.., Nuseibeh, B. (2005): Arguing Security: Validating Security Requirements Using Structured Argumentation, Proceedings of the 3rd Symposium on Requirements Engineering for Information Security held in conjunction with the 13th International Requirements Engineering Conference.
11. Hughes, B., and Cotterell, M. (2006): Software Project Management (4th Edition), McGraw Hill, ISBN 0 07 710989 9.
12. Lamsweerde, A. (2004): Elaborating Security Requirements by Construction of Intentional Anti-Models, Proceedings of the 26th International Conference on Software Engineering, pp. 148-157.
13. Lamsweerde, A., and Letier, E. (2000): Handling Obstacles in Goal-Oriented Requirements Engineering, IEEE Transactions on Software Engineering, Vol. 26, No. 10, pp. 978-1005.
14. Lamsweerde, A., Letier, E and Ponsard, C. (1997): Leaving Inconsistency, Position paper for the ICSE'97 workshop on Living with Inconsistency.
15. McDermott, J. (2001): Abuse-Case-Based Assurance Arguments, Proceedings of the 17th Computer Security Applications Conference, pp. 366-374.
16. Olesen, K., and Myers, M. (1999): Trying to improve communication and collaboration with information technology: an action research project which failed, Information Technology and People, Vol. 12, pp. 12-27.
17. Page, V., Dixon, M., and Choudhury, I. (2006): Mitigating Data Gathering Obstacles within an Agile Information Systems Development Environment, Proceedings of the 10th International Conference on Intelligent Engineering Systems, pp. 11-16.
18. Page, V., Laney, R., Dixon, M., and Haley, C. (2006): Trust Obstacle Mitigation for Database Systems, Proceedings of the 23rd British National Conference on Databases, pp. 254-257.
19. Potts, C. (1995): Using Schematic Scenarios to Understand User Needs, Proceedings of the ACM Symposium on - Designing Interactive Systems: Processes, Practices, and techniques, pp. 247-256.
20. Sindre, G., and Opdahl, A (2000).: Eliciting Security Requirements by Misuse Cases, Proceedings of the 37th International Conference on Technology Object-Oriented Languages and Systems, pp. 120-131.
21. Stallings, W. (2005): Business Data Communications (5th Edition), Pearson Prentice Hall, ISBN 0 13 127633 6.
22. Stolen, K. (2002): Model-based risk assessment – the CORAS approach, Presented at the 1st iTrust Workshop.
23. Viega, J., Kohno, T., and Potter, B. (2001): Trust (and mistrust) in Secure Applications, Communications of the ACM, Vol. 44, No. 2, pp. 31-36.

Curriculum Vitae of Authors

Victor Page

Victor Page is Senior Lecturer in Information Systems at Kingston University. He worked for BT from 1979-1997. His last ten years in BT were spent in various software development roles. After resigning from BT he also worked as a lecturer at London Guildhall University, London Metropolitan University and the University of Westminster. He holds an MSc in Information Systems Design from the University of Westminster.

Maurice Dixon

Maurice Dixon is Reader in Computational Modelling at London Metropolitan University. He is a long term Visitor to the CCLRC’s BIT (now e-Science) Department at the Rutherford Appleton Laboratory. He spent 8 years in software development for a SME. Prior to that he was a computational modelling researcher at Reading and then Oxford University and attached to AERE Harwell. He holds both a BSc and a PhD degree from the University of Leeds.

Islam Choudhury

Islam Choudhury has been a lecturer in Information Systems for the last 12 years. He has worked at South Bank University, London Guildhall University and London Metropolitan University. He holds a BSc from the University of Wales College of Cardiff and a PhD, funded by BT, from South Bank University. His research interests include generic business modelling, workflow modelling and software process improvement.