2011

Modelling misuse cases as a means of capturing security requirements

Michael N. Johnstone
Edith Cowan University

DOI: 10.4225/75/57b536ddcd8c1
Originally published in the Proceedings of the 9th Australian Information Security Management Conference, Edith Cowan University, Perth Western Australia, 5th -7th December, 2011

MODELLING MISUSE CASES AS A MEANS OF CAPTURING SECURITY REQUIREMENTS

Michael N. Johnstone
School of Computer and Security Science
Edith Cowan University, Perth, Western Australia
m.johnstone@ecu.edu.au

Abstract

Use cases as part of requirements engineering are often seen as an essential part of systems development in many methodologies. Given that modern, security-oriented software development methods such as SDL , SQUARE and CLASP place security at the forefront of product initiation, design and implementation, the focus of requirements elicitation must now move to capturing security requirements so as not to replicate past errors. Misuse cases can be an effective tool to model security requirements. This paper uses a case study to investigate the generation of successful misuse cases by employing the STRIDE framework as used in the SDL.

Keywords
Security, Vulnerability, Software Engineering, Information Systems Security, Conceptual Modelling, UML

INTRODUCTION

Bishop (2005, p.4) describes a threat as "a potential violation of security.", whereas a vulnerability, as defined by the Internet Engineering Task Force is "A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy." (RFC 2828, 2000, p. 190). Consequently an exploit is an attack that takes advantage of a vulnerability by realising a threat. One of the aims of secure development (apart from building software systems) is to identify and mitigate threats before they become exploitable vulnerabilities in production systems.

Software engineering as a discipline is still maturing, so it is not unreasonable that secure development is still in its infancy. Most software is insecure, according to Shostack and Stewart (2008). This could be because, as Wysopal et al. (2007) note, security requirements are often omitted from requirements specifications altogether. Security-oriented software development methodologies are an obvious attempt to address this issue. The most widely-reported of these methodologies are CLASP (OWASP, 2006), SDL (Howard and Lipner, 2006; Microsoft, 2010) and SQUARE (Mead et al., 2005). As with any new methodology, none have been extensively field-tested yet, but the fact that these methods place security requirements at the vanguard of all stages of their respective development lifecycles must result in better (more secure, therefore more useable) systems, provided that the methodologies have a sound theoretical underpinning and that they are applied correctly by experienced practitioners).

Misuse cases (Sindre and Opdahl, 2001, 2005, 2008) are one technique that can be used to capture security requirements. This paper starts by describes the difference between UML use cases and misuse cases, explains the syntax and semantics of misuse cases by example, uses a case study to illustrate the effectiveness of misuse cases and concludes by considering some weaknesses of the technique.

UML USE CASES

Use cases were first reported by Jacobson et al. (1992) and have been part of the UML standard since its inception (perhaps unsurprising, as Jacobson is one of the "three amigos" of the UML). A use case is "*a description of a set of sequences of actions, including variants, that a system performs that yields an observable result of value to an actor*" (Booch et al., 1999, p468). Thus, a use case is a high-level, user-focussed description of what a system will do which serves to define the scope of the system, therefore use cases are closely bound to requirements. Use cases are often expressed in a diagram (see figure 1) but are, in fact, textual representations of processes. A use case diagram consists of use cases (represented by the ellipses) surrounded by a system boundary with actors (shown as stick figures) outside the system acting on it to initiate processes (use cases). A use case diagram provides information about the major functionality of a system, who is enabled to perform that functionality and relationships between the functions. The relationships are of four types, viz:

association (actor/actor), inclusion (use case/use case), extension (use case/use case) and generalisation (actor/actor or use case/use case).

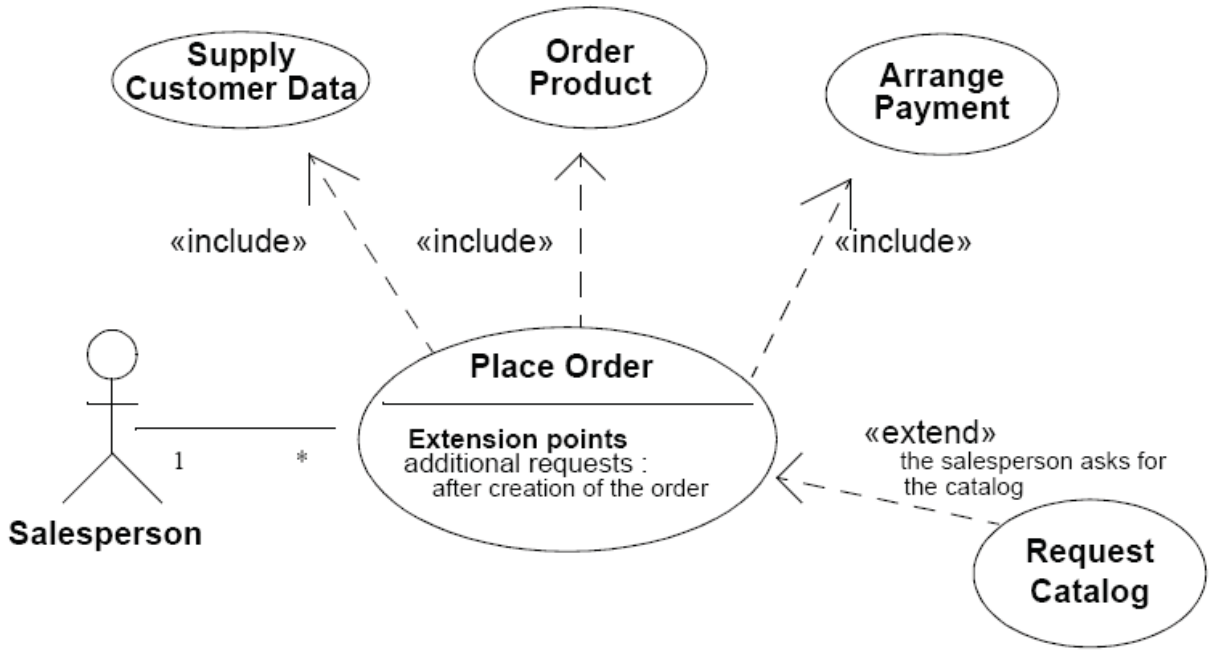

Figure 1: A Use Case Diagram (OMG, 2003, p. 3-99).

Association connects actors to use cases. In this example, a salesperson can place many orders. The «include» stereotype facilitates reuse of common functionality, similar to the way a function call operates in programming, so as part of the "Place Order" use case, the "Supply Customer Data" use case will be invoked. The «extend» stereotype allows optional behaviour, similar to selection or conditionals in programming, thus as part of the "Place Order" use case, the "Requesr Catalog" use case may be invoked. Generalisation (not shown on figure 1) allows IS-A relationships to be described e.g. a Salesperson is an employee or "Pay by credit card" is a way to "Arrange Payment".

Actors can be roles played by humans, other systems or hardware devices. UML version 2 (OMG, 2007) allows for other actor icons for example, other icons that convey the kind of actor may also be used to denote an actor, such as using a separate icon for nonhuman actors, but this usage appears to be fairly uncommon in practice and actors tend to represented as stick figures no matter what kind they are.

Having provided the basic details of the use case diagram technique, the next section introduces misuse cases as a security-oriented analogue of use cases.

MISUSE CASES

Whilst use cases are helpful in eliciting functional requirements, they do little for non-functional requirements (of which security requirements are a sub-category). Sindre and Opdahl (2001, p1) describe a misuse case as *"…the inverse of a use case, i.e., a function that the system should not allow…one could define a misuse case as a completed sequence of actions which results in loss for the organization or some specific stakeholder*.". Similarly, "*A mis-actor is the inverse of an actor, i.e., an actor that one does not want the system to support, an actor who initiates misuse cases*."

In conjunction with the concepts of misuse case and mis-actor (sometimes called a misuser), Sindre and Opdahl (2001) also propose extra stereotypes, viz: «prevents» and «detects», to aid in modelling security relationships. For example, an "Encrypt message" use case «prevents» a "Tap communications" misuse case and a "Monitor system" use case «detects» an "Obtain password" misuse case (see figure 2). In a misuse case diagram, the misuse cases and mis-actors are depicted in inverse colours to distinguish them from conventional use cases and actors.

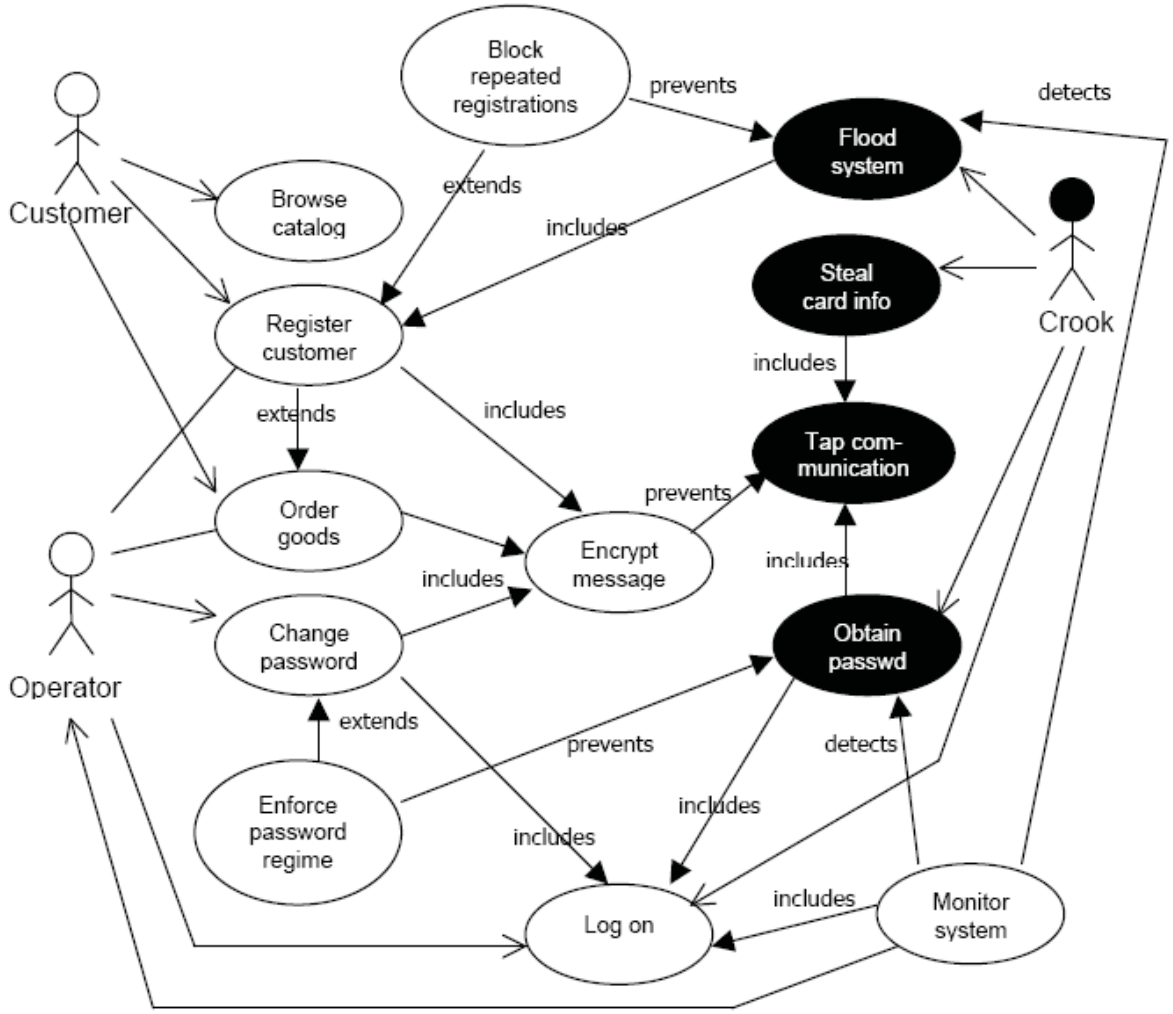

Figure 2: A Misuse Case Diagram (Sindre and Opdahl, 2000).

Later work by Sindre and Opdahl (2005) appears to have discarded those stereotypes in favour of «threatens» and «mitigates». In the same way that a use case can «include» another, a misuse case may also «include» the functionality of another misuse case. It may not, however, be obvious that the relationships between use cases and misuse cases are a) a use case can «mitigate» a misuse case; and b) a misuse case can «threaten» a use case.

Other work on misuse cases includes Firesmith (2003), Alexander (2003), Hope et al. (2004) and De Win et al. (2008). Firesmith (2003) elaborates on the subtle distinction between misuse cases and security use cases. The former are, of course, about specifying what happens when a mis-actor succeeds, whilst the latter are concerned with ensuring that the system succeeds (or rather the mis-actor does not). Firesmith would say that "Perpetrate fraud" is a misuse case (because it is performed by a mis-actor, with the aim of exploiting a system) whilst "Ensure non-repudiation" is a security use because it «mitigates» the "Perpetrate fraud" use case. The idea of security use cases is also supported by Damodaran (2006).

Firesmith (2003) also distinguishes between the differing purposes of misuse cases and security use cases in that whilst misuse cases can be an effective technique for analysing threats, they should not be used for specifying (security) requirements because security requirements are about providing protection from threats. This separation between threat and mitigation is seen in figure 3, where misuse cases drive security use cases.

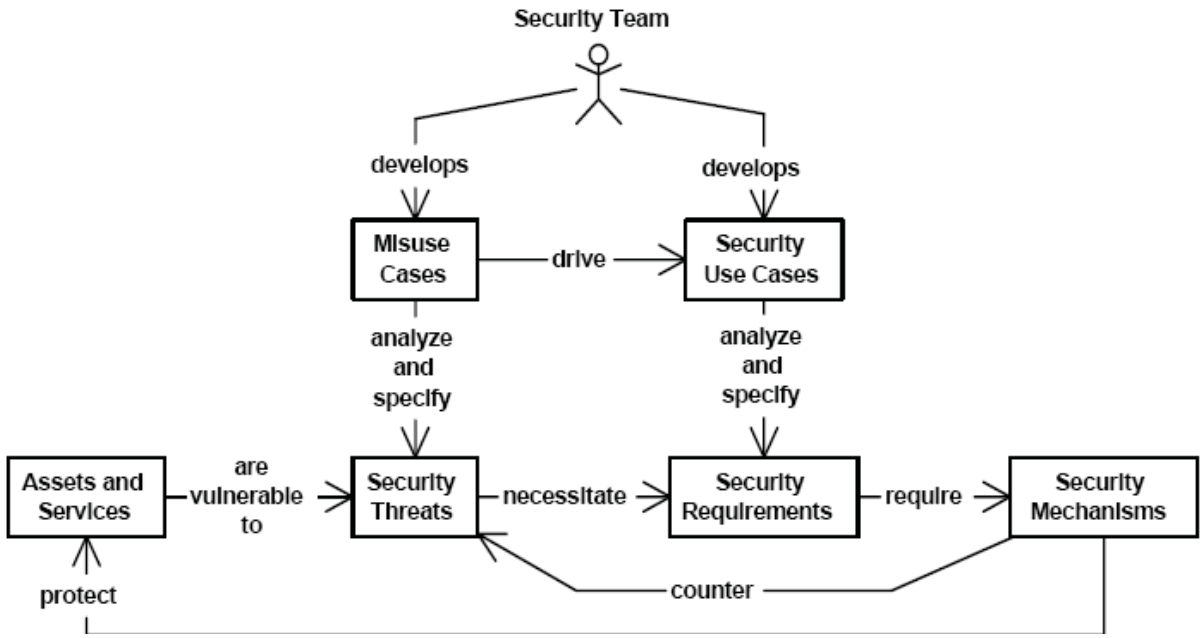

Figure 3: Misuse Cases and Security Use Cases (Firesmith, 2003, p54).

Alexander (2003) points out that excessive security controls can hamper access by legitimate users of a system and suggests «aggravates» and «conflicts with» as extensions to model relationships with the level of control required.

Hope et al. (2004) support the employment of misuse cases but state that security is an emergent property not a software feature and thus can't be added into the system as an afterthought. They argue that security must be considered at project inception. Whilst this is a laudable aim, many software engineers know that the addition of security functions change the scope, time and cost of a project and such functions may be left unimplemented as a product shipping date looms, especially as security requirements are not as visible as functional requirements. This type of behaviour is concerned with deferring risk as noted by Johnstone (2009).

In comparing SDL, CLASP and TouchPoints, De Win et al. (2008) note that CLASP takes both a black hat and a white hat perspective, it utilises misuse cases and identifies defence mechanisms for those misuse cases, thus generating security requirements as discussed by Firesmith (2003). De Win et al. mention that SDL does not utilise misuse cases, but there is no reason that they can't be integrated into SDL as will be seen in a later section.

APPLYING MISUSE CASES: A CASE STUDY

The case study used is that described by Howard and Lipner (2006) which is an e-commerce system for a pet shop that allows customers to purchase goods over the Internet. Customers may be members or anonymous users. Orders may be placed and paid-for immediately or deferred. If the goods ordered are not in stock then a back order is raised but the customer is not charged until the goods are shipped. Howard and Lipner modified the original case study slightly to include an audit log. For simplicity, the specific area of the case that will be focused on is the order processing system. Other actors, such as the Anonymous User and Administrator will be ignored as will extra use cases involved in processing deferred orders.

Using this pet shop e-commerce case, key actors and processes were identified, thus generating a conventional use case diagram (figure 4). This was then extended to cover misuse cases by considering likely security threats (figure 5).

Even though figure 5 is effectively only a partial misuse case diagram (as it contains only a single misuse case tied to processing a payment), it is nonetheless instructive. What is interesting about figure 5 is that it was actually created in a two-stage process. First, the misuser (Hacker) and misuse case were introduced into figure 4 (the use case diagram). Use cases are about successful completion and misuse cases are effectively the inverse of use cases (that is, they act to disrupt the system in some way), so in the second stage the security use case

"Encrypt Transaction" was introduced to mitigate the effect of the "Tamper with Payment Details" misuse case on the system.

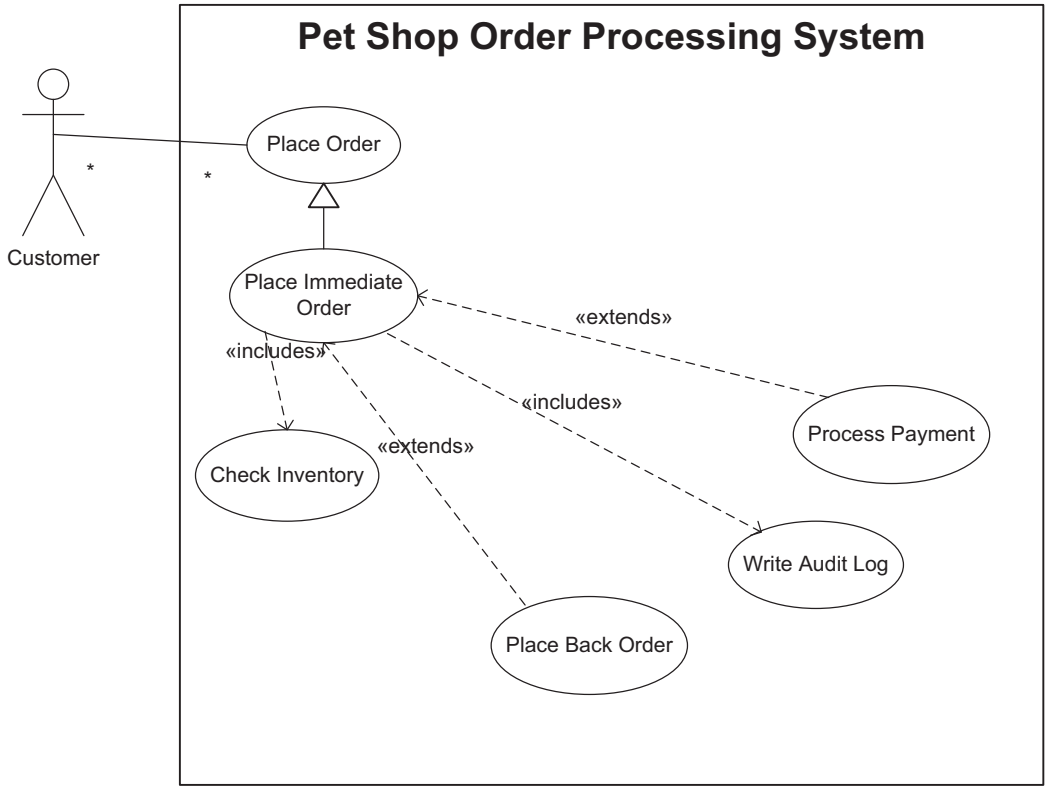

Figure 4: Partial Use Case Diagram derived from the Pet Shop Case Study.

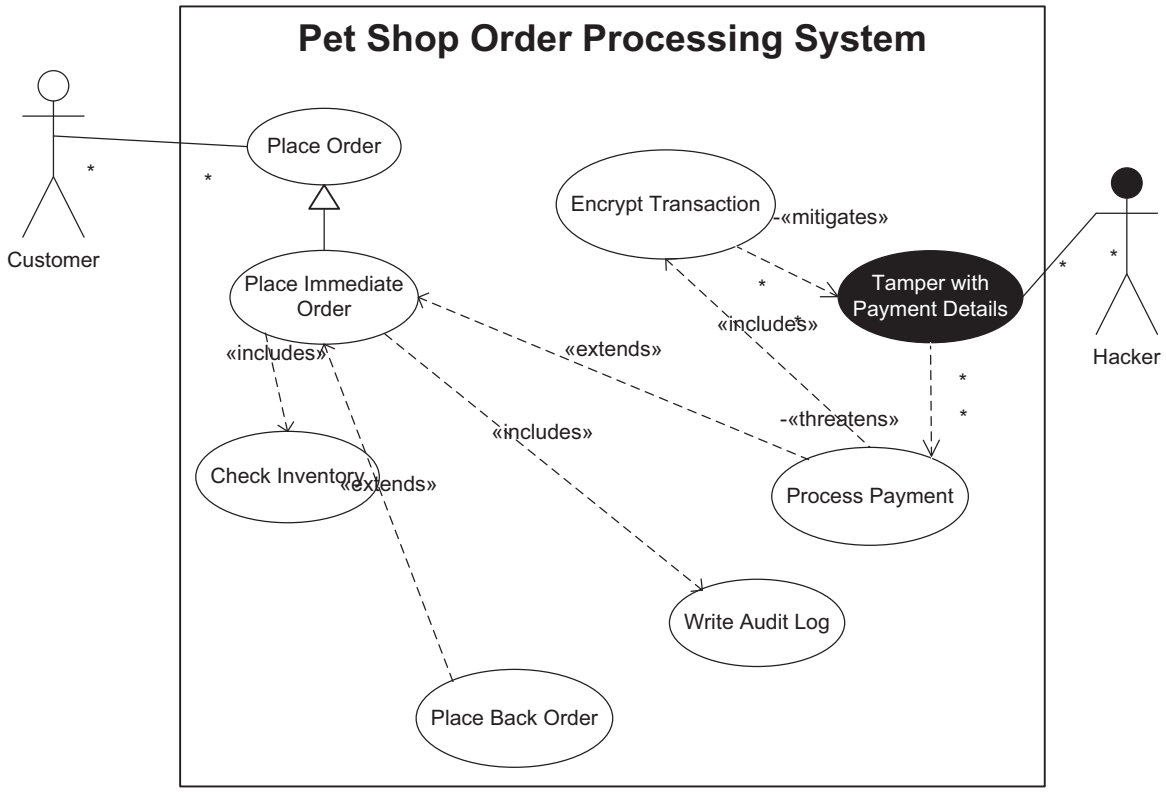

Figure 5: Partial Misuse Case Diagram derived from Figure 4.

ANALYSIS AND DISCUSSION

Mead et al. (2005) evaluated a range of requirements elicitation techniques, with misuse cases comparing favourably against the majority of the other techniques, especially in the areas of adaptability and ease of learning (see table 1). The evidence from the case study and table 1 suggests that generating misuse cases (and their associated security use cases) is a relatively straightforward task, however this is not so.

Table 1: Comparison of Requirements Elicitation Techniques (Mead et al., 2005, p38).

	Misuse Cases	SSM	QFD	CORE	IBIS	JAD	FODA	CDA	ARM
Adaptability	3	1	3	2	2	3	2	1	2
CASE Tool	1	2	1	1	3	2	1	1	1
Client Acceptance	2	2	2	2	3	2	1	3	3
Complexity	2	2	1	2	3	2	1	1	2
Graphical Output	2	2	1	1	2	1	2	2	3
Implementation Duration	2	2	1	1	2	1	2	2	3
Learning Curve	3	1	2	1	3	2	1	1	1
Maturity	2	3	3	3	2	3	2	2	1
Scalability	1	3	3	3	2	3	2	1	2

Scale: 3 = very good, 2 = fair, 1 = poor.

The ability to generate misuse cases and then the corresponding mitigating security use cases requires specialised knowledge and is, in fact, a non-trivial task. There are several key areas where decisions must be made that affect the security of the system, viz. the identity of the misusers, the scope of the misuse cases and the corresponding mitigations.

The misuser in the case study was an external hacker. Whilst this is perhaps an obvious choice, no consideration was given to internal mis-actors (grey-hats) such as frustrated employees who circumvent security controls because they slow down the system or disgruntled employees who actively subvert or decide to destroy the system. Røstad (2006) proposed that internal mis-actors should be represented in grey to differentiate them from external mis-actors. Internal threats can assist in identifying misuse cases such as "Increase Privileges" or "Reject Payment" (not shown on figure 5) as only legitimate system users can execute them (therefore not considered as an action of the “Hacker” actor).

How to create effective misuse cases is, then, an interesting question. Conventional requirements engineering elicitation techniques are not appropriate as there would be no existing documentation about mis-uses of the system, nor would interviewing a client help (unless the client knows the answer and the requirements engineer knows the right question ask in the first place-unlikely). A common suggestion is to brainstorm with a black hat on. This is possibly valid as placing oneself in the other persons position can be effective. This method does have one significant drawback–the requirements engineer is not the hacker and can never be, so this technique is not entirely successful, leading to vulnerabilities. Further, the person responsible for the system development may be too close to the code to notice vulnerabilities (an all too common failing). How to address this issue is problematic, to say the least. One solution might be to employ a ‘tiger team’ in the testing phase-at least this splits the roles dealing with the functional requirements from the non-functional (security) requirements. As a ‘tiger team’ is contracted to exploit vulnerabilities, it is effectively mimicking potential actions of a hacker.

Another way to generate misuse cases could be to use a set of existing pre-determined patterns, much like the Gang-of-Four design patterns in software engineering. Just as design patterns are experience-based solutions to

existing problems, a catalogue of misuse case patterns could serve the same purpose in a security context. There are two difficulties with this approach. First, it works for known problems (vulnerabilities), so how well it might work for unknown vulnerabilities is open to question and second, it presupposes that a requirements engineer has the skill set and experience to recognise that the pattern will solve the problem (an issue with software design patterns).

A further way to produce misuse cases is by using a security framework. This idea has some merit as it is abstract like design patterns (so it can be applied across a wide range of problem domains) and has the advantage of structuring the generation phase so that it is less likely that a misuse case will be missed (a problem with brainstorming). It was no accident that the misuse case in the previous section was named "Tamper with Payment Details". It could have been called "Perpetrate Fraud" or "Tap into Communications" without any loss of meaning. A framework such as STRIDE (an acronym for Spoofing identity, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege) as used in Microsoft's SDL could be the answer. Notice that the name of the misuse case matches one of the STRIDE elements (tampering, in this case). Therefore, to use this approach, the STRIDE elements become the template for the misuse cases. This could be done by mapping the STRIDE elements to misuse case constructs and then eliciting the misuse cases from the intersection (see table 2 for one such mapping generated for the Pet Shop case study). It then remains to generate the security use cases by considering mitigation strategies for the threats posed by the misuse cases (for example, "Encrypt Transaction" is the security use case that mitigates the "Tamper with Payment Details" misuse case in figure 5). It is possible that other threat modelling taxonomies may be effective at generating misuse cases, but this has not been tested.

Table 2: Mapping STRIDE to Misuse Case Elements.

MUC Construct	S	T	R	I	D	E
Actor				X*		
Mis-Actor (external)	X	X		X	X	
Mis-Actor (internal)		X	X	X		X

* Actors may accidentally disclose information.

In summary, the difficulties in discovering misuse cases were discussed, despite the deceptive simplicity of the case study and several methods for generating misuse cases put forward, with threat modelling within a security framework appearing to be the most likely candidate in terms of coverage for the generation of all required misuse cases. Ultimately, it may be that experience in secure development is the key determinant of the success of any threat modelling technique.

CONCLUSIONS AND FURTHER WORK

This study explored the utility of misuse cases in threat modelling. The complex nature of the generation of effective misuse cases was revealed and several techniques for creating misuse cases were articulated and discussed.

Specifically, this study used a well-known case study to show how a misuse cases could be effectively generated using the STRIDE taxonomy within the SDL's risk analysis process. It was argued that such a method provided benefits in that misuse cases (and the corresponding mitigating security use cases) were more likely to be found, as compared to simplistic, unstructured techniques such as brainstorming. It was noted that other threat modelling taxonomies may also be effective at eliciting misuse cases, but no others have been tested as yet.

A limitation of this work is that it covered only a single case study, therefore it would be unwise to conclude that STRIDE is the best or only way to generate misuse cases. Further work would field-test this idea to see how well the STRIDE mapping technique provides coverage in threat models of real-world systems. As noted, there are other threat models that could be substituted for STRIDE (the well-known confidentiality-integrity-availability or CIA structure is one possibility). Various threat models could be tested across a selection of case studies to see exactly which misuse cases are generated. This could then be confirmed (or otherwise) by using empirical studies.

REFERENCES

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

Bishop, M. (2005). *Introduction to Computer Security.* Boston, MA: Addison Wesley.

Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley.

Damodaran, M. (2006). Secure Software Development using Use Cases and Misuse Cases. Issues in Information Systems. VII(1): 150-154.

De Win, B., Scandariato, R., Buyens, K., Grégoire, J. and Joosen, W. (2009). On the secure software development process: CLASP, SDL and Touchpoints compared. *Information and Software Technology*. 51(7): 1152-1171.

Firesmith, D.G. (2003). Security Use Cases. *Journal of Object Technology*. 2(3): 53-64.

Hope, P., McGraw, G. and Antón, A.I. (2004). Misuse and Abuse Cases: Getting Past the Positive. *IEEE Security & Privacy*, May/June: 32-34.

Howard, M. and Lipner, S. (2006) *The Security Development Lifecycle. SDL: A Process for Developing Demonstrably More Secure Software*. Redmond, WA: Microsoft Press.

Jacobson, I., Christerson, M., Jonsson, P., and Övergaard. G. (1992). *Object-Oriented Software Engineering*. Reading, MA: Addison-Wesley.

Johnstone M.N. (2009). "Security Requirements Engineering-The Reluctant Oxymoron." *Proceedings of the 7th Australian Information Security Management Conference*, Edith Cowan University, Perth Western Australia, 1st-3rd December 2009.

Mead, N.R. Hough, E.D. and Stehney, T.R. (2005). Security Quality Requirements Engineering (SQUARE) Methodology. *TECHNICAL REPORT CMU/SEI-2005-TR-009*. Pittsburgh, PA: CMU.

Microsoft (2010). *Microsoft Security Development Lifecycle Version 5.0*. Redmond VA: Microsoft.

OMG (2003). *OMG Unified Modeling Language Specification version 1.5*. Needham, MA: Object Management Group, Inc.

OMG (2007). *OMG Unified Modeling Language Superstructure, V2.1.2*. Needham, MA: Object Management Group, Inc.

OWASP. (2006). *OWASP CLASP (Comprehensive, lightweight application security process) Project*. Retrieved August 20, 2009, from http://www.owasp.org.

RFC 2828 (2000). *Internet Security Glossary*. Internet Engineering Task Force. Retrieved September 22, 2009, from http://www.ietf.org/rfc/rfc2828.txt

Røstad, L. (2006). An Extended Misuse Case Notation: Including Vulnerabilities and the Insider Threat. *Proc. Twelfth Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'06),* Luxembourg, June 5-6.

Sindre, G. and Opdahl, A.L. (2001). Capturing Security Requirements through Misuse Cases. Retrieved August 22, 2010, from http://www.nik.no/2001/21-sindre.pdf

Sindre, G. and Opdahl, A.L. (2005). Eliciting Security Requirements with Misuse Cases. *Requirements Engineering*. 10: 34–44.

Sindre, G. and Opdahl, A.L. (2008). Misuse Cases for Identifying System Dependability Threats. *Journal of Information Privacy and Security*. 4(2): 3–22.

Shostack, A. and Stewart, A. (2008). *The New School of Information Security*. Upper Saddle River, NJ: Addison Wesley.

Wysopal, C., Nelson, L., Dai Zovi, D. and Dustin, E. (2007). *The Art of Software Security Testing*. Upper Saddle River, NJ: Addison Wesley.