Technical Report No: 2003/10

Analysing Security Threats and Vulnerabilities Using Abuse Frames

L.Lin
B.A.Nuseibeh
D.C.Ince
M.Jackson
J.D.Moffett

October 2003

Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom

http://computing.open.ac.uk

Analysing Security Threats and Vulnerabilities Using Abuse Frames

L. Lin, B. Nuseibeh, D. C. Ince, M. Jackson, J. D. Moffett[1]

Department of Computing, The Open University, Walton Hall, Milton Keynes MK7 6AA
{L.C.Lin, B.A.Nuseibeh, D.C.Ince, M.Jackson}@open.ac.uk
[1]Department of Computer Science, University of York, Heslington, York Y010 5DD
jdm@cs.york.ac.uk

Abstract. In this paper, we present an approach using *problem frames* to analyse security problems in order to determine security threats and vulnerabilities. We use problem frames to capture and bound the base system that is to be protected. We consider threats to this base problem frame from the point of view of the attacker. For each class of threats, their successful realisation is regarded as the *anti-requirement* in an *abuse frame*. Anti-requirements are quantified existentially: that is, the attacker succeeds by realising the threat in any one instance. For a threat to be realised, its abuse frame must be composed with the base problem frame in the sense that the asset attacked in the abuse frame must overlap, or be identified with, a domain of the base problem frame. We explain the process of composition and some of its variations. We illustrate and assess our approach using a case study of a medical information system, and suggest how abuse frames can provide a means for bounding the scope of and reasoning about security problems in order to analyse security threats and identify vulnerabilities. We conclude with an agenda for future work.

1 Introduction

The security engineering community has developed a variety of techniques for managing and protecting computer-based information. These techniques focus primarily on design and implementation issues, such as security mechanisms for detecting attacks and countermeasures for reacting to security breaches. They also focus on the notion of threats, driven by risk-analyses carried out at the later stages of the development process life-cycle, but the results are often unsatisfactory [7, 9, 28]. This has limited the support for effective reasoning about security objectives during the early stages of the development process.

To address these problems, the requirements engineering community [29] has started to investigate systematic approaches to analysing security threats [3, 12, 13, 17, 21, 27, 34] and security requirements [4, 5, 13, 15, 24, 28]. However, current techniques for analysing and reasoning about security requirements are still preliminary [28, 30]. They often lack the explicit notion of a security threat posed by malicious attackers. Without this, many security threats cannot be expressed explicitly and security measures cannot be determined effectively. A well defined system boundary enables systems engineers to focus on the characteristics of problem domains and their interactions. Non-trivial security vulnerabilities can be uncovered more easily and security threats reduced by selecting appropriate security measures.

In this paper we present an approach using problem frames [19] to analyse security threats and identify security vulnerabilities. Our approach introduces two conceptual tools – *anti-requirements* and *abuse frames* – and deploys these systematically to explore security problems in a medical information system. The paper is structured as follows. The next section briefly introduces problem frames and the relevant issues of security engineering and security requirements. Section 3 describes our approach, including patterns for different classes of security threats. Section 4 describes a case study of a medical information system to illustrate and assess our approach. Section 5 discusses related work, and section 6 concludes the paper.

2. Background

2.1 Problem Frames

A problem frame defines an identifiable problem class in terms of its context and the characteristics of its domains, interfaces, and requirements. Domain and interface characteristics are based on an identification of *phenomena*—element of what can be observed in the world. Phenomena may be individuals or relations. Individuals are *entities, events,* or *values*. Relations are *roles, states,* or *truth*s.

Domains can be one of three types – *causal, lexical,* or *biddable* [18, 19]. A causal domain is one whose properties include predictable causal relationships among its phenomena. A causal domain may control some, all, or none of the shared phenomena at an interface with another domain. A biddable domain consists of entities that lack positive predictable internal causality (e.g., people). A lexical domain is a physical representation of data that is represented as *symbolic* phenomena (e.g., data or information).

To illustrate the structure (the *principal parts*) of problem frames, consider a simple patient record display system:

A patient record display system is required as part of the medical information system for a hospital. Patient records are stored in a database, and the information of a particular patient record is displayed on a screen upon the request of a clinician.

Based on this, a requirement (*Req*) may be identified: *Information from a patient record should be displayed correctly upon the request of a clinician.*

Figure 1 overleaf is an *information display* problem frame diagram describing the problem. In the figure, the plain rectangles represent problem domains, and the connecting lines represent interfaces between them. A rectangle with two double vertical stripes represents the machine to be developed. This machine is built in the form of software and deployed by running the software on a general-purpose computer. The dashed oval represents the *requirement*; *Req* in this example. The dashed lines connecting the oval to a problem domain represent a *requirement reference;* that is, the requirement refers to certain phenomena of the problem domain.

A dashed arrow denotes the requirement reference as a *constraining reference* – meaning that the requirement stipulates some desired relationships or behaviour involving the domain phenomena. A causal domain is indicated by a C in the bottom right corner. The Clinician is biddable and is indicated by a B in the bottom right corner, whereas the PR domain is a lexical domain and is indicated by an X.

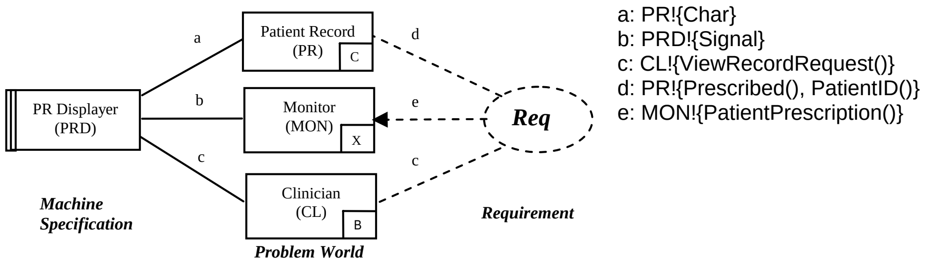


Figure 1: The problem frame diagram for a patient record display system

Interface phenomena are annotated to show the sets of shared phenomena between two domains. For example, the shared phenomenon *PR!{Char}* represents the set of *characters* that represents the symbolic states (*Prescribed(), PatientID()*) of the PR and is accessible by the PR Displayer. The prefix *PR!* also shows that *Char* is controlled by the PR; that is, the PR Displayer cannot cause *Char* to change.

To show that a problem can be solved, the systems engineer needs to derive a specification of the machine and produce a *correctness argument* that fits requirements, specification, and the problem world. The correctness argument should show that, given the machine specification (*S*) and *all* possible behaviours in the world (*W*), the requirement (*R*) is always satisfied. That is: $S, W \rightarrow R$. This relies on the fact that the *W* is strictly bounded by the problem frame diagram. For Figure 1, a correctness argument demonstrates that a Patient Record will be displayed correctly on the Display as requested by the Clinician.

2.2 Security Engineering and Security Requirements

In traditional security engineering, a *threat* is the potential for abuse of assets, and is characterised in terms of an attacker, a presumed *attack* method, any *vulnerabilities* that are exploited by the attack, and the *asset* under attack. An *attack* is a sequence of events resulting in a threatening phenomenon. An *attacker* is a malicious user, not necessarily a human, causing an attack. A vulnerability is the condition of a system exploited by an attacker to cause a security violation [11].

The security engineering community often partitions security for information systems into three categories: *confidentiality* is the protection of information assets against unauthorised access; *integrity* is the protection of assets from unauthorised alteration and corruption, while *availability* is the prevention of unauthorised degradation of the accessibility of assets.

Information security is achieved by introducing security measures that satisfy a system's *security requirements*. A security requirement—often a constraint on functional requirement—ensures that a particular security violation cannot happen [28]. In Figure 2, we represent a security requirement (SR) as a requirement in a problem frame.

E3 identifies the potential shared phenomena between the Security Machine and an Attacker. It can be regarded as the phenomena that describe the attacks from the Attacker and the reactions from the Security Machine.

The Security Machine is the machine to be built. The specification of the Security Machine—E2 and E3—represents the security measure to counteract attacks from the Attacker. *E1* identifies the *desirable* phenomena of the Asset that is under attack.

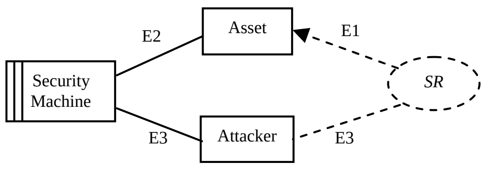


Figure 2: A security problem frame diagram

The correctness argument of the security problem frame must show that, in the presence of an attacker, the specification of the Security Machine satisfies the security requirement. However, an immediate difficulty arises in that the frame diagram in Figure 2 provides no explicit indication of the scope of the problem domain, nor does it describe the underlying vulnerabilities to the threats imposed by the attacker. Without these, a detailed specification of the Security Machine cannot be determined. To address these issues, we need to describe the behaviours of attackers and analyse security vulnerabilities in such a bounded scope.

3. Introducing Abuse Frames

In [12], we introduced the notion of *anti-requirement*s (AR) to represent the requirements of malicious attackers. Anti-requirements are expressed in terms of the problem domain phenomena and are quantified existentially: an anti-requirement is satisfied when the security threats imposed by the attacker are realised in any one instance of the problem. In this paper, we incorporate anti-requirements into abuse frames (Figure 3). The purpose of abuse frames is to represents security threats and to facilitate the analysis of the conditions in the system in which a security violation occurs. They allow the examination of a system's vulnerabilities to different kinds of security threats in a bounded context. Abuse frames share the same notation as the normal problem frames, but each domain is now associated with a different meaning.

The Asset (AS) is the domain under attack. The Malicious Machine (M/M) domain acts as the interface between the attacker and the Asset domain. Its behaviour allows the attacker to achieve the anti-requirement. The Attacker domain represents the domain that is imposing the threat.

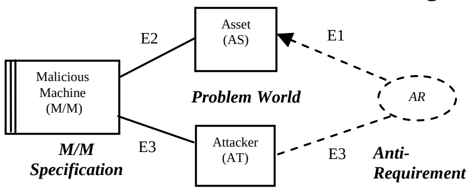


Figure 3: A generic abuse frame diagram.

The shared phenomenon AS!E1 represents the undesirable phenomenon in the victim domain as the result of an attack. It also shows that the phenomenon is controlled by the Asset domain (prefix AS!). The M/M is an abstract machine that is to be specified during abuse frame analysis. The M/M can be assigned to an existing domain, or a domain that is introduced into the problem world in order to reflect that an attacker can utilise the existing domains or other tools (e.g., a virus) that are not originally in the problem world *W*.

A security vulnerability is identified as the conditions in the problem world that combined with the Malicious Machine specification, will satisfy the anti-requirements. Because anti-requirements are existentially quantified, this means that a vulnerability is an instance of the problem world that gives rise to an opportunity for attacks. To show that a threat can be realised by the vulnerability, an *abuse frame argument* (ABA) must be constructed to show that the malicious machine specification (*MS*), with some behaviour in the problem world that satisfies the vulnerability conditions (*v(W)*), will achieve the anti-requirement: $MS,\ v(W) \rightarrow AR$. The argument relies on the fact that *W* is strictly bounded by the problem frame diagram; that is, any phenomena that are not explicitly shown on the diagram are not within the scope of *W* and, therefore, are not included in *W*. The vulnerability conditions *v* is expressed as the relations between the domain phenomena in *W*.

3.2 Abuse Frame Classes

In problem frames each frame describes a particular problem class (e.g., Information Display, Workpiece, and Required Behaviour frames). Similarly, abuse frames describe classes of security violation and include: *interception*, *modification*, and *denial of access*. Each represents a threat that can violate a particular security goal.

Interception, also known as *information disclosure*, arises whenever there is some information asset in the physical world that an attacker wishes to obtain, thus violating confidentiality. The abuse problem is to find a malicious machine that allows the attacker to achieve this. Figure 4 shows a standard interception frame.

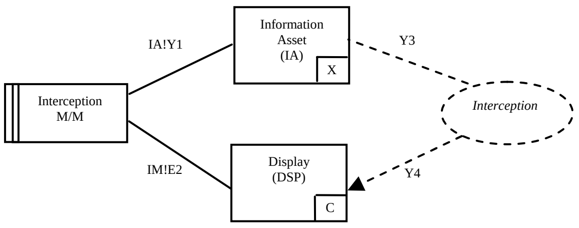


Figure 4: A standard interception abuse frame.

The Attacker domain is not shown in the frame diagram because we assume the attacker has access to the Display domain.

Modification arises whenever there is an information asset in the physical world that an attacker wishes to change. The problem is to find a modification machine that allows an attacker to achieve it. Modification violates *integrity*. Figure 5 shows a standard modification frame.

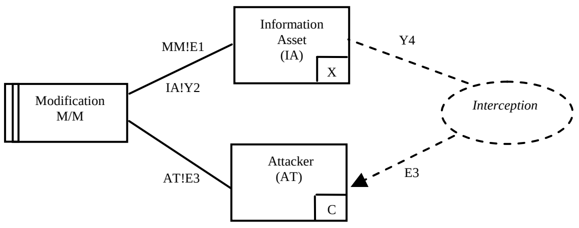


Figure 5: A standard modification abuse frame.

Denial of access arises whenever there is some information asset in the physical world that an attacker wishes to make unavailable or unusable. Denial of access violates *availability*. Figure 6 shows a standard denial of access frame.

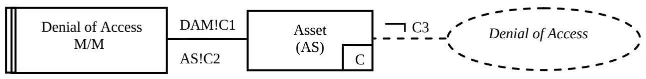


Figure 6: A standard denial of service abuse frame.

The classification of security problems in this way allows the exploration of the correspondences between standard problem frames and abuse frames (e.g., Table 1).

Problem Frames	Abuse Frames
Information Display	*Interception*
Workpiece	*Modification*
Required behaviour	*Denial of Access*

Table 1: Some correspondences between problem frames and abuse frames.

Each row in the table suggests a correspondence between a problem class and an abuse frame class, and each correspondence indicates that there is a direct mapping between the principal parts of the corresponding frame diagrams.

3.3 Abuse Frame Analysis

The purpose of abuse frame analysis is to analyse the threats to a *base problem*, which is the system to be protected and bounded by a problem frame, and to identify security vulnerabilities. Threats imposed by an attacker are represented by an abuse frame that captures the anti-requirement of an attacker. The abuse frame is then composed with the base problem.

To show that a threat is realisable, an abuse frame argument is constructed demonstrating the anti-requirement is satisfied by a sequence of interactions of domain phenomena. The security vulnerability is then identified as the conditions on the base problem that allow this sequence of interactions to take place.

3.3.1 Abuse Frame Composition

Abuse frames are composed with a base problem by *mapping* domains in an abuse frame to those in the base problem. Composition may be of two forms. The first applies when a base problem contains machine domains that can be abused directly by an attacker, without the introduction of additional domains, i.e. the attacker exploits the vulnerability that *already* exists in the base problem. This form of composition requires a direct correspondence between the principal parts of the base problem and those of the corresponding abuse frames. Table 1 suggests that the principal parts of an Interception abuse frame and an information display problem frame can be mapped directly. So the composition can proceed as follows:

- The Information Asset (IA) domain in the interception frame is mapped to the monitored domain, whose phenomena are monitored and displayed in the information display base problem.
- The Display (DSP) domain in the interception frame is mapped to the Display domain that displays the phenomena of the monitored domain in the information display base problem.
- The Malicious Machine in the Interception frame is mapped to the Information Machine, which is the machine domain in the information display base problem.

Any remaining unmapped domains in the base problem also remain in the composed abuse frame. Figure 7 shows an example of the patient record display base problem (Figure 1) composed with the interception frame (Figure 4). In this example, the IA in the interception frame is mapped to the Patient Record (PR), which is the monitored domain in the patient record display base problem. The DSP in the interception frame is mapped to the Monitor (MON) domain, which is responsible for displaying the information of the PR. The interception M/M maps to the machine domain, namely PRD, in the base problem. The CL domain is unmapped in the base problem, and is preserved in the composed abuse frame.

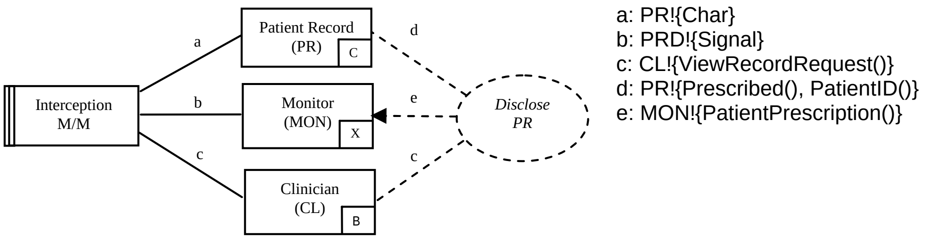


Figure 7: Composed interception abuse frame for a patient record display problem.

In the second form of composition, additional domains are introduced to *perturb* the domain(s) in the base problem, i.e. the vulnerability in the base problem is identified by extending the its scope to include new domains. A perturbation occurs when a domain outside the scope of the base problem can establish some kind of interface phenomena with a domain in the base problem. These interface phenomena are shared between these two domains, and each phenomenon is controlled either by the new domain or by the perturbed domain but not both. Each phenomenon may also be related by casual properties, i.e. one phenomenon triggers the happening of another.

A *perturbation mapping* needs to be identified before the composition can proceed. It indicates the mapping from the Asset domain in an abuse frame to the perturbed domain in the base problem. There are no fixed rules for deciding the perturbation mapping for a composition—different perturbations must be explored. However, a perturbed domain must be in the base problem and is mapped to the Asset domain in an abuse frame. Also, only *domains* in the base problem can be perturbed; interface phenomena (indicated by a line between domains) cannot be perturbed.

An example of perturbation is shown in Figure 8. Here, the PR is the domain being perturbed, and it is mapped to the Information Asset in the interception frame.

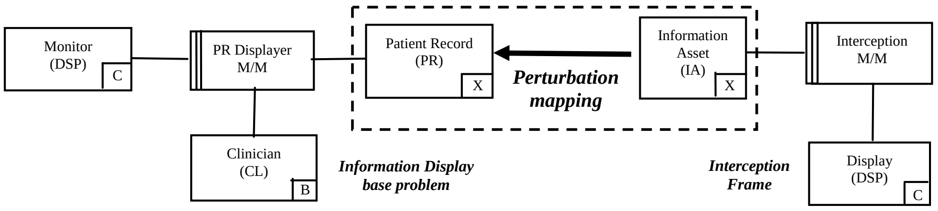


Figure 8: Perturbing a patient record display problem frame with an interception frame.

Figure 9 shows the resulting composed abuse frame. The newly introduced domains are the Interception M/M (IMM) and Display.

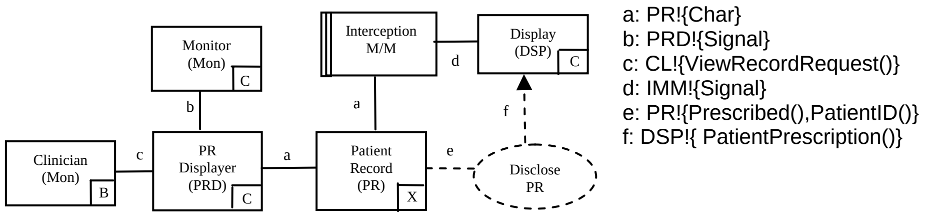


Figure 9: Composition of an interception abuse frame with the patient record display problem frame.

This form of composition extends the scope of the base problem by introducing new domains that will participate in achieving the anti-requirement. The extended scope is bounded by the problem context of the composed abuse frame, which is analysed to reveal vulnerabilities to the threats imposed by the composed abuse frame.

The problem world *W* of the composed abuse frame is the union of the domains in the base problem and the new non-machine domains introduced.

3.3.2 The Abuse Frame Argument

A composed abuse frame diagram needs to be realised in order to show that the anti-requirement is satisfied. This is achieved by constructing an abuse frame argument (ABA). The central task of constructing such an argument is to investigate and describe the properties of the domains in the problem world and to derive the Malicious Machine specification. The ABA must fit these descriptions together and provide a sequence of interaction of domain phenomena as the evidence that the anti-requirement can be satisfied.

For different classes of abuse frames, different descriptions of the properties of the domains are needed. Using the composed interception frame in Figure 7 as an example, the relevant descriptions are:

- Derive descriptions of the domains in the problem world. Because there is no non-machine domain introduced, only the descriptions of the domains in the base problem are needed.
  - For the Information Asset, describe the relationship between *Char* and *Prescribed()*, and the relationship between *Char* and *PatientID()*. They are the properties of the PR.
  - For the Display domain, describe the relationship between *Signal* and *PatientPrescription()*. It is the property of the MON.
  - For the Clinician domain, describe the event phenomena generated by the domain. It is the property of the CL.
- Derive the Malicious Machine specification. In Figure 7, the specification of the *IMM* can be derived through the following steps:
  - Use the property of the PR to determine the set of characters (*Char*) that resemble the information *Prescribed(), PatientID()* as stated in the anti-requirement.
  - Use the property of the MON to determine the *Signal* that is required to cause the information *PatientPrescription()* , as stated in the anti-requirement, to be displayed.
  - Derive the relationship between *ViewRecordRequest()*, *Char*, and *Signal*. This is the specification of IMM.

- The descriptions of *IMM* and *W* must be shown to satisfy the anti-requirement, by constructing the ABA: *IMM, CL, PR, MON* $\rightarrow$ *AR.*

Thus, the ABA for Figure 9 will show that *PatientPrescription()*, which is derived from *Prescribed() and PatientID()* of the PR, will be disclosed if the CL issues the request *ViewRecordRequest()*, because the Attacker has access to the *PatientPrescription()* displayed on the MON. A more detailed account of an ABA is provided in the case study in the next section.

If no ABA can be constructed successfully to demonstrate the satisfaction of an anti-requirement, this indicates that the threats imposed by the abuse frame cannot be realised within the bounded context.

3.3.3 Identification of Security Vulnerabilities

A security vulnerability is identified as the conditions that describe the participating domains in the ABA, and their behaviour, that will satisfy the anti-requirement. We use u_n to denote the condition on domain *n* for *n* to exhibit the behaviours described in the ABA. u_n is expressed in terms of the phenomena of *n*. The vulnerability *v* in an ABA is defined as the union of the conditions *u* on the participating domains. New domains that are introduced to perturb the base problem during composition are regarded as outside the scope of the base problem and are not included in *v*.

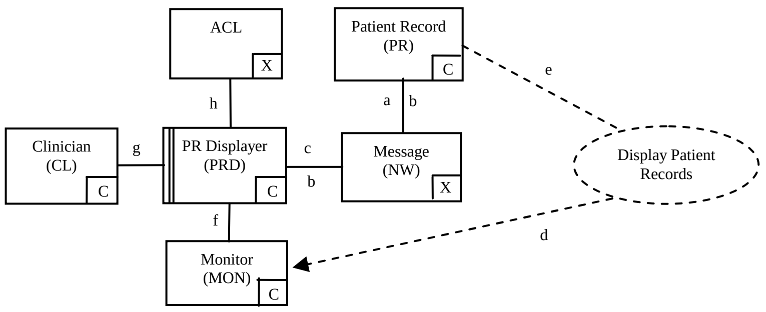
It is often helpful if the set of participating domains that are used to identify the vulnerability is *minimal*. A set of domains is said to be minimal when the removal of any domains from the set will result in the failure to construct the ABA (i.e. failing to satisfy the anti-requirement). For example, the minimal set of participating domains for the ABA for Figure 7 is the set {CL, PRD, PR, MON}. The condition on each domain is denoted by u_{CL} , $u_{PRD,}$ u_{PR}, and u_{MON}, respectively. u_{CL} represents the condition that the CL issues *ViewRecordRequest()*. u_{PR} represents the condition that the PR is non-empty, i.e. the PR contains some patient information. U_{MON} represents the condition that the attacker has access to the domain. No conditions are required on the PRD (assuming perfect reliability of the PRD, MON, and PR), and therefore u_{PRD} always holds. The vulnerability condition *v* is therefore the set $\{u_{CL}, u_{PR,,} u_{PRD}, u_{MON}\}$.

The reason for representing vulnerability as a set of conditions on each participating domain is to allow each participating domain to be treated individually when the vulnerability is being addressed. Addressing vulnerabilities is done through the addition of new security requirements (e.g., only allowing non-sensitive patient information to be displayed on the Monitor) or the revision of existing requirements (e.g., substituting the Monitor with an output device to which the attacker has no access). In both cases, systems engineers need to show that the anti-requirement cannot be satisfied in the context of the new system.

4 Case Study

This section illustrates and assesses our approach by using it to analyse part of a medical information system (MIS). The case study is from the British Medical Association policy model proposed in [2], and is an example of a real practical system with important security requirements.

One primary task of an MIS is to provide medical information to clinicians. The medical records of each patient are stored as electronic clinical records in an information system, and are only accessible to clinicians responsible for that patient. Each patient is associated with an access control list (ACL), and the MIS must prevent anyone not on the ACL from accessing the patient's records in anyway. Only a designated clinician can modify the ACL. Part of the case study concerned with the display of patient records is shown in Figure 10 as an information display problem frame.



a: PR!{MsgWrite()}	b: NW!{header, content}	c: PRD!{MsgWrite()}
d: PR!{PatientPrescription()}	e: PR!{Prescribed(), PatientID()}	f: PRD!{Signal}
g: CL!{ViewRecordRequest()}	h: ACL!{AssignedTo()}	

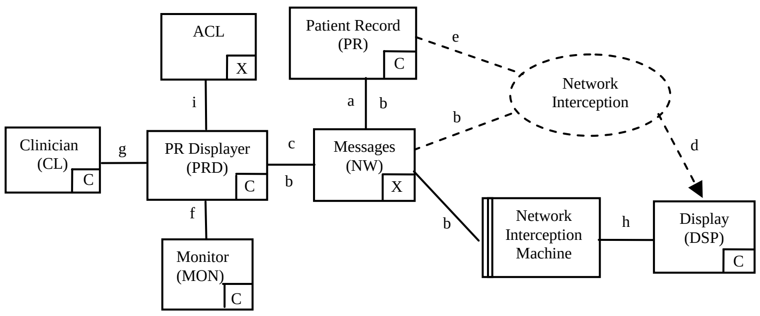
Figure 10: A display frame for displaying patient records in the MIS.

The PR is connected to the PRD through an internal network. The PR and PRD communicate by sharing a network Message domain. To view the information of a particular patient *Pname* (we assume each *Pname* is unique), the CL issues *ViewRecordRequest()* command to the PRD, which then accesses the information *AssignedTo()* of the ACL to verify whether the clinician is responsible for that patient. The PRD also has direct access to the PR. The PRD causes the event phenomena *Signal* to the Monitor to produce the required phenomena *PatientPrescription()*.

The Message domain consists of two major parts: the *header* and *content*. The header contains the identifications of the sender and receiver, and the content contains the information to be transmitted. We use *Msg(h, c)* to denote a state that holds if and only if *h* is the header, and *c* is the content of the message, and we use the notation *h.sender* to identify the sender (e.g., *h.sender* = PRD), and *h.receiver* to identify the receiver.

4.1 Abuse Frame Analysis

Suppose an attacker wishes to intercept the network messages to disclose information of the patient record that is transmitted through the network. We call this anti-requirement "Network Interception", which stipulates that the information *PatientPrescription()* shall be obtained from the network messages between the PR and the PRD and present them on an output device. This threat is represented in an interception abuse frame (Figure 4). The composed abuse frame diagram of Figure 4 and 10 is shown in Figure 11.



a: PR!{MsgWrite()}
b: NW!{header, content}
c: PRD!{MsgWrite()}
d: PR!{PatientPrescription()}
e: PR!{Prescribed(), PatientID()}
f: PRD!{Signal}
g: CL!{ViewRecordRequest()}
h: NIM!{Signal}
i: ACL!{AssignedTo()}

Figure 11: Abuse frame for network interception

The PR is represented as a causal domain, as it generates network messages to be transmitted to the PRD. The newly introduced domains in the composed abuse frame diagram are the Network Interception Machine (NIM) and the Display. The perturbed domain is the Message domain, with which the NIM shares the symbolic states *header* and *content*, which are controlled by the Message domain.

4.2 The Abuse Frame Argument

To construct the ABA, we need to describe the properties of the domains in the base problem (NW, PR, ACL, PRD, CL), the newly introduced domain (DSP), and the malicious machine (NIM). We then develop the ABA from which vulnerabilities are identified. Due to page constraints, domain descriptions are presented in the appendix.

4.2.1 Specification of NIM

We assume that the NIM has a simple built-in local memory to store temporary symbolic phenomenon. (i.e. temporary data). The specification of the NIM is developed by iterating through the following four steps.

1. Recognise the header *h* and content *c* for each message *Msg(h, c)*. Examine *h* to identify the *sender* and *receiver*.
2. If *h.sender* = PR and *h.receiver* = PRD, the NIM examines the *content c* to retrieve the information, which is either *Prescribed(P_ID, med)* or *PatientID(P_ID, Pname)*.
3. If *c = Prescribed(P_ID, med),* the NIM accesses *PatientID(P_ID, Pname)* in its local memory, and derive *PatientPrescription(Pname, med)*, using the property of the PR.
   If *c = PatientID(P_ID, Pname)*, the NIM stores *PatientID(P_ID, Pname)* in local memory and return to 1.
4. The NIM generates *Signal* to display the derived information *PatientPrescription(Pname, med)* on the DSP.

4.3 The Abuse Frame Argument

In the following ABA, we use *PR_PRD* to represent the header of the message that is transmitted from the PR to the PRD, and we use *PRD_PR* to represent the header of the message that is transmitted from the PRD to the PR. The underlined descriptions describe the behaviours of the newly introduced domains (NIM, DSP).

1. Once the CL imitates the event *ViewRecordRequest(Pname, C_ID),* the PRD produces a network message *Msg(PRD_PR, c1),* where *c1 = QueryID(Pname).*
2. Once received the query from the PPD, the PR produces a network message *Msg(PR_PRD, c2),* where *c2 = PatientID(P_ID, Pname).*
3. The NIM intercepts the network message and interprets *c2* to obtain *PatientID(P_ID, Pname).*
4. The PRD verifies whether the state *AssignedTo(P_ID, C_ID)* holds in the ACL.
5. If the state *AssignedTo(P_ID, C_ID)* holds in the ACL, the PRD produces a network message *Msg(PRD_PR, c3),* where *c3 = QueryP(P_ID).*
6. Once received the query from the PPD, the PR produces a network message *Msg(PR_PRD, c4), where c4 = PatientP(P_ID, med).*
7. The NIM intercepts the network message and interprets *c4* to obtain *PatientP(P_ID, med).*
8. The NIM derives *PatientPrescription(Pname, med)* from *PatientP(P_ID, med)* and *PatientID(P_ID, Pname).*
9. The NIM sends *Signal* events to the DSP according to the information derived, and the DSP shows the information of the patient record that is transmitted through the network.

Thus, we have shown that the anti-requirement Network Interception is satisfied, and a security vulnerability exists.

4.4 Identification of Security Vulnerability

For the above ABA, the minimal set of participating domains after removing the new domains introduced (NIM and DSP) is {PR, NW, PRD, CL, ACL}. The condition on each domain is listed as the follows:

u_{PR} : the PR contains patient information.

u_{NW} : the symbolic state *Msg(header, content)* of NW is accessible to and interpretable by the NIM.

u_{CL} : the CL causes the event *ViewRecordRequest(Pname, C_ID).*

$u_{ACL:}$ the state *AssingedTo(P_ID, C_ID)* holds, i.e. the Clinician *C_ID* is responsible for the patient *P_ID*.

The vulnerability *v* for the ABA is {u_{PR} , u_{NW}, u_{CL} , u_{ACL}}. Informally, *v* states that a vulnerability exists because the NIM has access to network messages and is able to interpret the messages transmitted through network, and that the patient information is transmitted across this network.

After the security vulnerability is identified, the properties of each participating domain in *v* should be investigated to determine the appropriate treatment to address the vulnerability. This is beyond the scope of this paper.

5 Related Work

van Lamsweerde suggests that threats can be treated as obstacles to goals [23]. An obstacle defines a set of undesired behaviours that violate a positive goal. However, this implies a particular boundary of security attacks, and may cause some wider security threats to be overlooked. In [24], the notion of *anti-goals* is introduced. This is similar in spirit to our notion of anti-requirements, but anti-goals are still derived from the traditional security objectives, whereas anti-requirements exist as long as there are users of the envisaged system whose requirements conflict with a system's requirements, and may potentially abuse the system.

The *i** framework [26] takes an organisational view by modelling trustworthiness as softgoals to be satisfied . Attacks by malicious users are modelled as negative contributions that obstruct these softgoals. *i** has also been applied to assess the criticality and complexity of actors to model security vulnerabilities in a multi-agent system [8]. However, *i** focuses on analysis of security threats imposed by internal actors at the organisational level, whereas our approach complements this by analysing both internal and external threats in software systems problem domains.

The GBRAM framework comprises a set of techniques for extracting high level goals from various data sources and subsequently refining them into detailed requirements specifications based on a set heuristics and guidelines [6]. Chung [9, 10], treats security requirements as softgoals, that are identified and refined based on a knowledge catalogue of decomposition methods, security techniques for satisficing softgoals, and correlation rules. Both approaches focus on the elicitation process of high level security goals, which, again we regard as complementary to our approach.

Attack trees [31], adopt a goal-oriented approach to refining a root goal into a goal tree to derive scenarios. They are applied late in the design phase when a mature system design is available, and the construction of an attack tree usually requires extensive knowledge in security analysis. Similarly, software fault trees [17] allow systematic analysis of security attacks, but they are best suited to design.

Misuse cases [1, 33] and abuse cases [27] have been suggested as tools for threat analysis. However, we argue that these typically represent instances of attacks that are often described in the language of implementation. Abuse frames on the other hand, treat the solution domain as a black box and focus on the problem domain.

Research on reusable patterns for security analysis has also received attention recently. At the enterprise level, Kis proposes using *anti-patterns* [21] as patterns of vulnerabilities for enterprise-wise security requirements analysis. This is analogous to abuse frames, but our focus is on software system issues rather than business issues. Sindre *et al.* propose using reuse cases [35] for security analysis. A reuse is a pair of generic misuse and use cases. Application-specific misuse and uses cases are derived by applying the pair to the context of an envisaged system. However, as with misuse and abuse cases, reuse cases are most suited at the design level.

Firesmith also suggests specifying security requirements from reusable templates of commonly identified security requirements [14]. Our approach focuses on analysing threats and identifying security vulnerabilities, and is complementary to this approach.

Security patterns such as [22] and [32] are variants of design patterns for specifying security measures. Each design pattern addresses a particular category of security concerns. Design patterns are generally used after security requirements analysis, whereas our work specifically focuses on identifying security threats and vulnerabilities during the requirements analysis.

SCR has been applied to develop a formal specification of a communication security device [20] and a user authentication system [16]. The specification describes normal behaviour of the device plus the actions that should be taken in response to an undesired event, such as equipment failures or physical tampering. However, our work focuses on the notion of malicious attackers, which is not addressed explicitly by SCR.

Finally, hazard analysis techniques from the safety engineering literature [25] share with security engineering the need to analyse undesirable phenomena. However, the introduction of users with malicious intent into security engineering means that scope of problems is harder to bound.

6 Discussion and Conclusions

We have presented a requirements-based approach for analysing security problems. We introduced anti-requirements and abuse frames, and shown how their integration into requirements analysis enables the explicit representation of security threats and facilitates early discovery of potential security vulnerabilities. Our approach provides a means of early structuring and bounding the scope of security problems when the requirements of the envisaged system are being elaborated and only partial information is available. We demonstrated this in the case study by structuring part of an MIS system using problems frames, and then composing these with abuse frames.

An important benefit of our approach is the discovery of *some* security vulnerabilities and requirements, though not all. Abuse frames are not a substitute for other traditional security engineering techniques. We have found them to be useful in complementing such techniques when deployed during requirements analysis. We have also noted that even though our analysis is currently informal, it is nevertheless systematic and repeatable. We are currently conducting a larger case study to explore a number of open issues. In particular, we are examining a more rigorous and systematic approach to constructing and expressing ABAs that lie at the heart of identifying vulnerabilities. We are also examining ways of deriving and revising security requirements once vulnerabilities have been identified. Preliminary investigations suggest that trust assumptions about domains in the problem context play a significant part in both identifying vulnerabilities and corresponding security requirements [15].

Acknowledgements
The financial support of Leverhulme Trust is gratefully acknowledged. We are also grateful to Charles Haley, Axel van Lamsweerde, and Nabukazu Yoshioka for their comments and insights.

References

1. I.F. Alexander, "Misuse cases: use cases with hostile intent", *IEEE Software,* 2003, 20(1): 58-66.
2. R. Anderson, "A Security Policy Model for Clinical Information Systems", *Proceeding of IEEE Symposium on Security and Privacy*, USA, 6-8 May 1996.
3. R. Anderson, "How to Cheat at the Lottery", *invited talk at the 12th Annual Computer Security Applications Conference*, USA, 9-13 Dec 1999.
4. A.I. Anton and J.B. Earp, "Strategies for Developing Policies and Requirements for Secure E-Commerce Systems", *1st ACM Workshop on Security and Privacy in E-Commerce (CCS 2000),* Greece, 1-4 Nov 2000.
5. A.I. Anton, J.B. Earp and T.A. Alspaugh, "The Role of Policy and Privacy Values in Requirements Engineering", *IEEE 5th International Symposium on Requirements Engineering (RE'01)*, Canada, 27-31 Aug 2001.

6. A.I. Anton, J.B. Earp and A. Reese, "Analyzing Web Site Privacy Requirements Using a Privacy Goal Taxonomy", *IEEE Joint Requirements Engineering Conference (RE'02)*, Germany, 9-13 Sep 2002.
7. R. Baskerville, "Information Systems Security Design methods: Implications for Information Systems Development", *ACM Computing Surveys,* 1993, 25(4): 375-414.
8. P. Bresciani, P. Giorgini and H. Mouratidis, "On Security Requirements Analysis for Multi-Agent Systems", *Proceedings of 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems SELMAS 2003 in conjunction with the 25th International Conference on Software Engineering (ICSE 2003)*, USA, May 3-4, 2003.
9. L. Chung, "Dealing with Security Requirements During the Development of Information Systems", *5th Int. Conf. Advanced Information Systems Engineering (CAiSE'93)*, 1993.
10. L. Chung, B. Nixon, E. Yu and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer, 2000.
11. *Common Criteria for Information Technology Security Evaluation, Version 2.1, August*, 1999. http://www.commoncriteria.org
12. R. Crook, D. Ince, L. Lin and B. Nuseibeh, "Security Requirements Engineering: When Anti-requirements Hit the Fan", *Proceeding of the 10th Requirements Engineering Conference (RE'02)*, Germany, 9-13 Sep 2002.
13. D. Firesmith, "Engineering Security Requirements", *Journal of Object Technology,* 2003, 2(1).
14. D.G. Firesmith, "Analyzing and Specifying Reusable Security Requirements", *RE'03 International Workshop on Requirements for High Assurance Systems*, USA, 9 Sep 2003.
15. C. Haley, R. Laney, B. Nuseibeh and J.D. Moffett, "Using Trust Assumptions in Security Requirements Engineering", *Second Internal iTrust Workshop On Trust Management In Dynamic Open Systems*, Imperial College, London, 15-17 Sept. 2003.
16. C. Heitmeyer, "Applying 'Practical' Formal Methods to the Specification and Analysis of Security Properties", *Proceeding of Information Assurance in Computer Networks (MMM-ACNS 2001),* Russia, 21-23 May 2001.
17. G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller and R. Lutz, "A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System", *Requirements Engineering Journal*, 2002, 7(4): 207-220.
18. M. Jackson, "Domain Descriptions", *in Proceedings of the IEEE International Symposium on Requirements Engineering*, 1993.
19. M. Jackson, *Problem Frames: Analysing and structuring software development problems*, Addison Wesley, 2001.
20. J. Kirby, M. Archer and C. Heitmeyer, "SCR: A Practical Approach to Building a High Assurance COMSEC System", *Proceedings of 15th Annual Computer Security Applications Conference (ACSAC '99)*, Dec 1999.
21. M. Kis, "Information Security Antipatterns in Software Requirements Engineering", *9th Conference on Pattern Language of Programs 2002*, USA, 8-12 Sep 2002.
22. S. Konrad, B.H.C. Cheng, L.A. Campbell and R. Wassermann, "Using Security Patterns to Model and Analyze Security Requirements", *RE'03 International Workshop on Requirements for High Assurance Systems*, USA, 9 Sep 2003.
23. A.van. Lamsweerde and E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering", *IEEE Transactions on Software Engineering, Special Issue on Exception Handling,* 2000, 26(10): 978-1005.
24. A.van. Lamsweerde, S. Brohez, R.D. Landtsheer and D. Janssens, "From System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering", *RHAS-03 Workshop, RE'03*, USA, 9 Sep 2003.
25. N. Leveson, *Safeware: System Safety and Computers*, Addison Wesley, 1995.
26. L. Liu, E. Yu and J. Mylopoulos, "Security and Privacy Requirements Analysis within a Social Setting", *International Conference on Requirements Engineering (RE03)*, USA, 8-12 Sep 2003.

27. J. McDermott and C. Fox, "Using Abuse Case Models for Security Requirements Analysis", *Annual Computer Security Applications Conference*, USA, 6-10 Dec 1999.
28. J. Moffett and B. Nuseibeh, "A Framework For Security Requirements Engineering", Department of Computer Science, YCS368. University of York, UK, 2003.
29. B. Nuseibeh and S. Easterbrook, "Requirements Engineering: A Roadmap", *Proceedings of International Conference on Software Engineering (ICSE-2000)*, Ireland, 4-11 Jun 2000.
30. J. Rushby, "Security Requirements Specifications: How and What?" *Invited paper presented at Symposium on Requirements Engineering for Information Security (SREIS)*, USA, Mar 2001.
31. B. Schneier, *Secrets & Lies: Digital Security in a Networked World*, Wiley, 2000.
32. M. Schumacher and U. Roedig, "Security Engineering with Patterns", *9th Conference on Pattern Language of Programs 2002*, USA, 8-12 Sep 2002.
33. G. Sindre and A.L. Opdahl, "Eliciting Security Requirements by Misuse Cases", *37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-PACIFIC 2000)*, 2000.
34. G. Sindre and A.L. Opdahl, "Templates for misuse case description", *In Proceedings of Seventh International Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ'2001)*,, Switzerland, 4-5 Jun 2001.
35. G. Sindre, D.G. Firesmith and A.L. Opdahl, "A Reuse-Based Approach to Determining Security Requirements", *In Proc. 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03)*, Austria, 16-17 Jun 2003.

Appendix

This appendix supplements the case study section by providing the descriptions of the domain properties in the base problem frame (Figure 10) and of the Display domain, which is introduced in the composed abuse frame (Figure 11).

1. Description of Message

A message comprises of two parts: *header* and *content*. We use *Msg(h, c)* to denote a state that holds if and only if *h* is the header, and *c* is the content of the message. The header provides the identification of the sender (represented by *h.sender*) and receiver (represented by *h.receiver*). For messages transmitted from the PRD to the PR, content *c* comprises either *QuaryID(Pname)* or *QueryP(P_ID).* For messages transmitted from the PR and PRD, the content *c* comprises one of the symbolic phenomena *PatientID(P_ID, Pname)* and *Prescribed(P_ID, medication).*

We use *PR_PRD* to represent the header of the message that is transmitted from the PR to the PRD, and we use *PRD_PR* to represent the header of the message that is transmitted from the PRD to the PR.

2. Description of Patient Record

The symbolic states of the PR represent patient information. The PR has any number of named patients and any number of prescriptions. Each patient may be prescribed to any number of prescriptions. The resulting symbolic states of PR are therefore:

Prescribed(*P_ID, medication****):*** a state that holds if and only if the patient, who is identified by *P_ID,* has prescribed to the *medication.*

PatientID(*P_ID, Pname****):*** a state that holds if and only if the patient of the name *Pname* is identified by *P_ID*.

PatientPrescription(*Pname, medication****):*** a state which holds if and only if the patient *Pname* has the prescription *medication.*

The three phenomena are related by the relation:

Prescribed(P_ID, medication), PatientID(P_ID, Pname) → *PatientPrescription(Pname, medication)*

To communicate with the PRD, the PR controls the event phenomena *MsgWrite(msg_part, S).* This is an event in which the PR writes the symbolic phenomenon *S* to the message part (header or content). The PR recognises two kinds of query commands from the PRD: *QueryP(P_ID)* and *QuaryID(Pname)*. The former queries the prescriptions of the patient identified by *P_ID*, and the latter queries the *P_ID* of the patient *Pname*. The PR reacts in response to the queries received from the PRD. That is, for any message *Msg(h, c)* received from the PRD, the PR creates the message *Msg(r_h, r_c)* such that:

- if *c* == *QueryID(Pname)*, then *MsgWrite(PR_PRD, PatientID(P_ID, Pname)).*
- if *c* == *QueryP(P_ID)*, then *MsgWrite(PR_PRD, PatientP(P_ID, medication)).*

3. Description of ACL

The ACL has any number of named patients and any number of prescriptions. Each patient is assigned to any number of clinicians. The symbolic state ***AssignedTo(****P_ID, C_ID****)*** of the ACL indicates a state that holds if and only if the Clinician identified by *C_ID* is responsible for the patient identified by *P_ID*. The phenomenon is controlled by the ACL, and is shared with the PRD.

4. Description of Clinician and Patient Record Displayer

The PRD controls the event phenomena *MsgWrite()* and *Signal*. It shares the event *ViewRecordRequest(Pname, C_ID)* with the Clincian. *C_ID* identifies the Clinician making the request, and *Pname* is the patient whose record is displayed. The behaviour of the PRD is described by iterating through the following steps:

1. Once the CL initiates the event *ViewRecordRequest(Pname, C_ID)*, the PRD produces a network message *Msg(PRD_PR, c1),* where *c1 = QueryID(Pname).*
2. The PRD receives the message *Msg(PR_PRD, c2)*, where *c2 = PatientID(P_ID, Pname).* The PRD accesses the ACL to verify whether the state *AssignedTo(P_ID, C_ID)* holds in the ACL.
3. If the state *AssignedTo(P_ID, C_ID)* holds in the ACL, the PRD produces a network message *Msg(PRD_PR, c3),* where *c3 = QueryP(P_ID).* Return to 1 otherwise.
4. The PRD receives the message *Msg(PR_PRD, c4)*, where *c4 = PatientP(P_ID, med).* The PRD derives *PatientPrescription(Pname, med)*, using the property of the PR.
5. PRD sends *Signal* events to the MON according to *PatientPrescription(Pname, med)*.

5. Description of Display

The only phenomenon of the DSP that is considered in our analysis is *PatientPrescription()*. The DSP is a domain newly introduced during abuse frame composition. It can be any causal domain that exhibits the symbolic phenomena recognisable to an attacker. We assume that the DSP is an output device similar to a monitor. Its symbolic phenomena are caused by the event phenomena *Signal*, which are controlled by the NIM.