A Framework for System Security [⋆]

Clark Thomborson

Department of Computer Science
The University of Auckland, New Zealand
cthombor@cs.auckland.ac.nz

March 4, 2009

Abstract. Actors in our general framework for secure systems can exert four types of control over other actors' systems, depending on the temporality (prospective vs. retrospective) of the control and on the power relationship (hierarchical vs. peering) between the actors. We make clear distinctions between security, functionality, trust, and distrust by identifying two orthogonal properties: feedback and assessment. We distinguish four types of system requirements using two more orthogonal properties: strictness and activity. We use our terminology to describe specialised types of secure systems such as access control systems, Clark-Wilson systems, and the Collaboration Oriented Architecture recently proposed by The Jericho Forum.

Keywords: Security, Trust, Distrust, Security Requirements, Security Models

1 Introduction

There are many competing definitions for the word "security", even in the restricted context of computerised systems. We prefer a very broad definition, saying that a system is *secure* if its owner ever estimated its probable losses from adverse events, such as eavesdropping. We say that a system is *secured* if its owner modified it, with the intent of reducing the expected frequency or severity of adverse events. These definitions are in common use but are easily misinterpreted. An unsupported assertion that a system is secure, or that it has been secured, does not reveal anything about its likely behaviour. Details of the estimate of losses and evidence that this estimate is accurate are necessary for a meaningful *assurance* that a system is safe to use. One form of assurance is a *security proof*, which is a logical argument demonstrating that a system can suffer no losses from a specific range of adverse events if the system is operating in accordance with the assumptions (*axioms*) of the argument.

In this chapter, we propose a conceptual framework for the design and analysis of secure systems. Our goal is to give theoreticians and practitioners a common language in which to express their own, more specialised, concepts. When

[⋆] This is a preprint version of a contributed chapter in the *Handbook of Computer and Information Security*, ed. Mark Stamp, Springer, to appear April 2009.

used by theoreticians, our framework forms a meta-model in which the axioms of other security models can be expressed. When used by practitioners, our framework provides a well-structured language for describing the requirements, designs, and evaluations of secure systems.

The first half of our chapter is devoted to explaining the concepts in our framework, and how they fit together. We then discuss applications of our framework to existing and future systems. Along the way, we provide definitions for commonly used terms in system security.

1.1 Systems, Owners, Security and Functionality

The fundamental concept in our framework is the *system* – a structured entity which interacts with other systems. We subdivide each interaction into a series of primitive actions, where each action is a *transmission event* of mass, energy, or information from one system (the provider) that is accompanied by zero or more *reception events* at other systems (the receivers).

Systems are composed of *actors*. Every system has a distinguished actor, its *constitution*. The minimal system is a single, constitutional, actor.

The constitution of a system contains a listing of its actors and their relationships, a specification of the interactional behaviour of these actors with other internal actors and with other systems, and a specification of how the system's constitution will change as a result of its interactions.

The listings and specifications in a constitution need not be complete descriptions of a system's structure and input-output behaviour. Any insistence on completeness would make it impossible to model systems with actors having random, partially unknown, or purposeful behaviour. Furthermore, we can generally prove some useful properties about a system based on an incomplete, but carefully chosen, constitution.

Every system has an *owner*, and every owner is a system. We use the term *subsystem* as a synonym for "owned system". If a constitutional actor is its own subsystem, i.e. if it owns itself, we call it a *sentient actor*. We say that a system is *sentient*, if it contains at least one sentient actor. If a system is not sentient, we call it an *automaton*. Only sentient systems may own other systems. For example, we may have a three-actor system where one actor is the constitution of the system, and where the other two actors are owned by the three-actor system. The three-actor system is sentient, because one of its actors owns itself. The other two systems are automata.

If a real-world actor plays important roles in multiple systems, then a model of this actor in our framework will have a different *aliased actor* for each of these roles. Only constitutional actors may have aliases. A constitution may specify how to create, destroy, and change these aliases.

Sentient systems are used to model organisations containing humans, such as clubs and corporations. Computers and other inanimate objects are modelled as automata. Individual humans are modelled as sentient actors.

Our insistence that owners are sentient is a fundamental assumption of our framework. The owner of a system is the ultimate judge, in our framework, of what the system should and shouldn't do. The actual behaviour of a system will, in general, diverge from the owner's desires and fears about its behaviour. The role of the system analyst, in our framework, is to provide advice to the owner on these divergences.

We invite the analytically-inclined reader to attempt to develop a general framework for secure systems that is based on some socio-legal construct other than a property right. If this alternative basis for a security framework yields any increase in its analytic power, generality, or clarity, then we would be interested to hear of it.

Functionality and Security. If a system's owner ascribes a net benefit to a collection of transmission and reception events, we say this collection of events is *functional behaviour* of the system. If an owner ascribes a net loss to a collection of their system's reception and transmission events, we say this collection of events is a *security fault* of the system. An owner makes judgements about whether any collection of system events contains one or more faults or functional behaviours. These judgements may occur either before or after the event. An owner may refrain from judging, and an owner may change their mind about a prior judgement. Clearly, if an owner is inconsistent in their judgements, their systems cannot be consistently secure or functional.

An analyst records the judgements of a system's owner in a *judgement actor* for that system. The judgement actor need not be distinct from the constitution of the system. When a system's judgement actor receives a description of (possible) transmission and reception events, it either transmits a summary judgement on these events or else it refrains from transmitting anything, i.e. it withholds judgement. The detailed content of a judgement transmission varies, depending on the system being modelled and on the analyst's preferences. A single judgement transmission may describe multiple security faults and functional behaviours.

A descriptive and interpretive report of a judgement actor's responses to a series of system events is called an *analysis* of this system. If this report considers only security faults, then it is a *security analysis*. If an analysis considers only functional behaviour, then it is a *functional analysis*. A summary of the rules by which a judgement actor makes judgements is called a *system requirement*. A summary of the environmental conditions that would induce the analysed series of events is called the *workload* of the analysis. An analysis will generally indicate whether or not a system meets its requirements under a typical workload, that is, whether it is likely to have no security faults and to exhibit all functional behaviours if it is operated under these environmental conditions. An analysis report is unlikely to be complete, and it may contain errors. Completeness and accuracy are, however, desirable aspects of an analysis. If no judgements are likely to occur, or if the judgements are uninformative, then the analysis should indicate that the system lacks effective security or functional requirements. If the judgements are inconsistent, the analysis should describe the likely inconsistencies and summarise the judgements that are likely to be consistent. If a judgement actor or a constitution can be changed without its owner's agreement, the analysis should indicate the extent to which these changes are likely to affect its security and functionality as these were defined by its original judgement actor and constitution. An analysis may also contain some suggestions for system improvement.

An analyst may introduce ambiguity into a model, in order to study cases where no one can accurately predict what an adversary might do and to study situations about which the analyst has incomplete information. For example, an analyst may construct a system with a partially-specified number of sentient actors with partially-specified constitutions. This system may be a subsystem of a complete system model, where the other subsystem is the system under attack.

An attacking subsystem is called a *threat model* in the technical literature. After constructing a system and a threat model, the analyst may be able to prove that no collection of attackers of this type could cause a security fault. An analyst will build a probabilistic threat model if they want to estimate a fault rate. An analyst will build a sentient threat model if they have some knowledge of the attackers' motivations. To the extent that an analyst can "think like an attacker", a war-gaming exercise will reveal some offensive manoeuvers and corresponding defensive strategies [10].

The accuracy of any system analysis will depend on the accuracy of the assumed workload. The workload may change over time, as a result of changes in the system and its environment. If the environment is complex, for example if it includes resourceful adversaries and allies of the system owner, then workload changes cannot be predicted with high accuracy.

1.2 Qualitative vs. Quantitative Security

In this section we briefly explore the typical limitations of a system analysis. We start by distinguishing qualitative analysis from quantitative analysis. The latter is numerical, requiring an analyst to estimate the probabilities of relevant classes of events in relevant populations, and also to estimate the owner's costs and benefits in relevant contingencies. Qualitative analysis, by contrast, is non-numeric. The goal of a qualitative analysis is to explain, not to measure. A successful qualitative analysis of a system is a precondition for its quantitative analysis, for in the absence of a meaningful explanation, any measurement would be devoid of meaning. We offer the following, qualitative, analysis of some other preconditions of a quantitative measurement of security.

A proposed metric for a security property must be *validated*, by the owner of the system, or by their trusted agent, as being a meaningful and relevant summary of the security faults in a typical operating environment for the system. Otherwise there would be no point in paying the cost of measuring this property in this environment. The cost of measurement includes the cost of designing and implementing the measurement apparatus. Some preliminary experimentation with this apparatus is required to establish the *precision* (or lack of noise) and *accuracy* (or lack of bias) of a typical measurement with this apparatus. These quantities are well-defined, in the scientific sense, only if we have confidence in the objectivity of an observer, and if we have a sample population, a sampling procedure, a measurement procedure, and some assumption about the ground truth for the value of the measured property in the sample population. A typical simplifying assumption on ground truth is that the measurement error is Gaussian with a mean of zero. This assumption is often invalidated by an experimental error which introduces a large, undetected, bias. Functional aspects of computer systems performance are routinely defined and measured [8], but computer systems security is more problematic.

Some security-related parameters are estimated routinely by insurance companies, major software companies, and major consulting houses using the methods of *actuarial analysis*. Such analyses are based on the premise that the future behaviour of a population will resemble the past behaviour of a population. A time-series of a summary statistic on the past behaviour of a collection of similar systems can, with this premise, be extrapolated to predict the value of this summary statistic. The precision of this extrapolation can be easily estimated, based on its predictive power for prefixes of the known time series. The accuracy of this extrapolation is difficult to estimate, for an actuarial model can be invalidated if the population changes in some unexpected way. For example, an actuarial model of a security property of a set of workstations might be invalidated by a change in their operating system. However, if the timeseries contains many instances of change in the operating system, then its actuarial model can be validated for use on a population with an unstable operating system. The range of actuarial analysis will extend whenever a population of similar computer systems becomes sufficiently large and stable to be predictable, whenever a timeseries of security-related events is available for this population, and whenever there is a profitable market for the resulting actuarial predictions.

There are a number of methods whereby an unvalidated, but still valuable, estimate of a security parameter may be made on a system which is not part of a well-characterised population. Analysts and owners of novel systems are faced with decision-theoretic problems akin to those faced by a 16th-century naval captain in uncharted waters. It is rarely an appropriate decision to build a highly accurate chart (a validated model) of the navigational options in the immediate vicinity of one's ship, because this will generally cause dangerous delays in one's progress toward an ultimate goal.

1.3 Security Requirements and Optimal Design

Having briefly surveyed the difficulty of quantitative analysis, and the prospects for eventual success in such endeavours, we return to the fundamental problem of developing a qualitative model of a secure system. Any modeller must create a simplified representation of the most important aspects of this system. In our experience, the most difficult aspect of qualitative system analysis is discovering what its owner wants it to do, and what they fear it might do. This is the problem of *requirements elicitation*, expressed in emotive terms. Many other expressions are possible. For example, if the owner is most concerned with the economic aspects of the system, then their desires and fears are most naturally expressed as benefits and costs. Moralistic owners may consider rights and wrongs. If the owner is a corporation, then its desires and fears are naturally expressed as goals and risks.

A *functional requirement* can take one of two mathematical forms: an acceptable lower bound or *constraint* on positive judgements of system events, or an *optimisation criterion* in which the number of positive judgements is maximised. Similarly, there are two mathematical forms for a *security requirement*: an upper-bounding constraint on negative judgements, or a minimisation criterion on negative judgements. The analyst should consider both receptions and transmissions. Constraints involving only transmissions from the system under analysis are called *behavioural constraints*. Constraints involving only receptions by the system under analysis are called *environmental constraints*.

Generally, the owner will have some control over the behaviour of their system. The analyst is thus faced with the fundamental problem in *control theory*, of finding a way to control the system, given whatever information about the system is observable, such that it will meet all its constraints and optimise all its criteria.

Generally, other sentient actors will have control over aspects of the environment in which the owner's system is operating. The analyst is thus faced with the fundamental problem in *game theory*, of finding an optimal strategy for the owner, given some assumptions about the behavioural possibilities and motivation of the other actors.

Generally, it is impossible to optimise all criteria while meeting all constraints. The frequency of occurrence of each type of fault and function might be traded against every other type. This problem can sometimes be finessed, if the owner assigns a monetary value to each fault and function, and if they are unconcerned about anything other than their final (expected) cash position. However, in general, owners will also be concerned about capital risk, cashflow, and intangibles such as reputation.

In the usual case, the system model has multiple objectives which cannot all be achieved simultaneously; the model is inaccurate; and the model, although inaccurate, is nonetheless so complex that exact analysis is impossible. Analysts will thus, typically, recommend suboptimal incremental changes to its existing design or control procedures. Each recommended change may offer improvements in some respects, while decreasing its security or performance in other respects. Each analyst is likely to recommend a different set of changes. An analyst may disagree with another analyst's recommendations and summary findings. We expect the frequency and severity of disagreements among reputable analysts to decrease over time, as the design and analysis of sentient systems becomes a mature engineering discipline. Our framework offers a language, and a set of concepts, for the development of this discipline.

1.4 Architectural and Economic Controls; Peerages; Objectivity

We have already discussed the fundamentals of our framework, noting in particular that the judgement actor is a representation of the system owner's desires and fears with respect to their system's behaviour. In this section we complete our framework's taxonomy of relationships between actors. We also start to define our taxonomy of control.

There are three fundamental types of relationships between the actors in our model. An actor may be an alias of another actor; an actor may be superior to another actor; and an actor may be a peer of another actor. We have already defined the aliasing relation. Below, we define the superior and peering relationships.

The superior relationship is a generalisation of the ownership relation we defined in section 1.1. An actor is the *superior* of another actor if the former has some important power or control over the latter, *inferior*, actor. In the case that the inferior is a constitutional actor, then the superior is the owner of the system defined by that constitution. Analysis is greatly simplified in models where the scope of control of a constitution is defined by the transitive closure of its inferiors, for this scoping rule will ensure that every subsystem is a subset of its owning system. This subset relation gives a natural precedence in cases of constitutional conflict: the constitution of the owning system has precedence over the constitutions of its subsystems.

Our notion of superiority is extremely broad, encompassing any exercise of power that is essentially unilateral or non-negotiated. To take an extreme example, we would model a slave as a sentient actor with an alias that is inferior to another sentient actor. A slave is not completely powerless, for they have at least some observational power over their slaveholder. If this observational power is important to the analysis, then the analyst will introduce an alias of the slaveholder that is inferior to the slave. The constitutional actor of the slaveholder is a representation of those aspects of the slaveholder's behaviour which are observable by their slave. The constitutional actor of the slave specifies the behavioural responses of the slave to their observations of the slaveholder and to any other reception events.

If an analyst is able to make predictions about the likely judgements of a system's judgement actor under the expected workload presented by its superiors, then these superiors are exerting *architectural controls* in the analyst's model. Intuitively, architectural controls are all of the worldly constraints that an owner feels to be inescapable – effectively beyond their control. Any commonly-understood "law of physics" is an architectural control in any model which includes a superior actor that enforces this law. The edicts of sentient superiors, such as religious, legal, or governmental agencies, are architectural controls on any owner who obeys these edicts without estimating the costs and benefits of possible disobedience.

Another type of influence on system requirements, called *economic controls*, result from an owner's expectations regarding the costs and benefits from their expectations of functions and faults. As indicated in the previous section, these costs and benefits are not necessarily scalars, although they might be expressed in dollar amounts. Generally, economic controls are expressed in the optimisation criteria for an analytic model of a system, whereas architectural controls are expressed in its feasibility constraints.

Economic controls are exerted by the "invisible hand" of a marketplace defined and operated by a *peerage*. A peerage contains a collection of actors in a *peering* relationship with each other. Informally, a peerage is a relationship between equals. Formally, a peering relationship is any reflexive, symmetric, and transitive relation between actors.

A peerage is a system; therefore it has a constitutional actor. The constitutional actor of a peerage is an automaton that is in a superior relationship to the peers.

A peerage must have a *trusted servant* which is inferior to each of the peers. The trusted servant mediates all discussions and decisions within the peerage, and it mediates their communications with any external systems. These external systems may be peers, inferiors, or superiors of the peerage; if the peerage has a multiplicity of relations with external systems then its trusted servant has an alias to handle each of these relations. For example, a regulated marketplace is modelled as a peerage whose constitutional actor is owned by its regulator. The trusted servant of the peerage handles the communications of the peerage with its owner. The peers can communicate anonymously to the owner, if the trusted servant does not breach the anonymity through their communications with the owner, and if the aliases of peers are not leaking identity information to the owner. This is not a complete taxonomy of threats, by the way, for an owner might find a way to subvert the constitution of the peerage, e.g. by installing a wiretap on the peers' communication channel. The general case of a constitutional subversion would be modelled as an owner-controlled alias that is superior to the constitutional actor of the peerage. The primary subversion threat is the replacement of the trusted servant by an alias of the owner. A lesser threat is that the owner could add owner-controlled aliases to the peerage, and thereby "stuff the ballot box."

An important element in the constitutional actor of a peerage is a decision-making procedure such as a process for forming a ballot, tabulating votes, and determining an outcome. In an extreme case, a peerage may have only two members, where one of these members can outvote the other. Even in this case, the minority peer may have some residual control if it is defined in the constitution, or if it is granted by the owner (if any) of the peerage. Such imbalanced peerages are used to express, in our framework, the essentially-economic calculations of a person who considers the risks and rewards of disobeying a superior's edict.

Our simplified pantheon of organisations has only two members – peerages and hierarchies. In a *hierarchy*, every system other than the *hierarch* has exactly one superior system; the hierarch is sentient; and the hierarch is the owner of the hierarchy. The superior relation in a hierarchy is thus irreflexive, asymmetric, and intransitive.

We note, in passing, that the relations in our framework can express more complex organisational possibilities, such as a peerage that isn't owned by its trusted servant, and a hierarchy that isn't owned by its hierarch. The advantages and disadvantages of various hybrid architectures have been explored by constitutional scholars (e.g. in the 18th-Century *Federalist Papers*), and by the designers of autonomous systems.

Example. We illustrate the concepts of systems, actors, relationships, and architectural controls by considering a five-actor model of an employee's use of an outsourced service. The employee is modelled as two actors, one of which owns itself (representing their personal capacity) and an alias (representing their work-related role). The employee alias is inferior to a self-owned actor representing their employer. The outsourced service is a sentient (self-owned) actor, with an alias that is inferior to the employee. This simple model is sufficient to discuss the fundamental issues of outsourcing in a commercial context. A typical desire of the employer in such a system is that their business will be more profitable as a result of their employee's access to the outsourced service. A typical fear of the employer is that the outsourcing has exposed them to some additional security risks. If the employer or analyst has estimated the business's exposure to these additional risks, then their mitigations (if any) can be classified as architectural or economic controls. The analyst may use an information-flow methodology to consider the possible functions and faults of each element of the system. When transmission events from the aliased service to the service actor are being considered, the analyst will develop rules for the employer's judgement actor which will distinguish functional activity from faulting activity on this link. This link activity is not directly observable by the employer, but may be inferred from events which occur on the employer-employee link. Alternatively, it may not be inferrable but is still feared, for example if an employee's service request is a disclosure of company-confidential information, then the outsourced service provider may be able to learn this information through their service alias. The analyst may recommend an architectural control for this risk, such as an employer-controlled filter on the link between the employee and the service alias. A possible economic control for this disclosure risk is a contractual arrangement, whereby the risk is priced into the service arrangement, reducing its monetary cost to the employer, in which case it constitutes a form of self-insurance. An example of an architectural control is an advise-and-consent regime for any changes to the service alias. An analyst for the service provider might suggest an economic control, such as a self-insurance, to mitigate the risk of the employer's allegation of a disclosure. An analyst for the employee might suggest an architectural control, such as avoiding situations in which they might be accused of improper disclosures via their service requests. To the extent that these three analysts agree on a ground truth, their models of the system will predict similar outcomes. All analysts should be aware of the possibility that the behaviour of the aliased service, as defined in an inferior-of-an-inferior role in the employer's constitution, may differ from its behaviour as defined in an aliased role in the constitution of the outsourced service provider. This constitutional conflict is the analysts' representation of their fundamental uncertainty over what will really happen in the real world scenario they are attempting to model.

Subjectivity and Objectivity We do not expect analysts to agree, in all respects, with the owner's evaluation of the controls pertaining to their system. We believe that it is the analyst's primary task to analyse a system. This includes an accurate analysis of the owner's desires, fears, and likely behaviour in foreseeable scenarios. After the system is analysed, the analyst might suggest refinements to the model so that it conforms more closely to the analyst's (presumably expert!) opinion. Curiously, the interaction of an analyst with the owner, and the resulting changes to the owner's system, could be modelled within our framework – if the analyst chooses to represent themselves as a sentient actor within the system model. We will leave the exploration of such systems to post-modernists, semioticians, and industrial psychologists. Our interest and expertise is in the scientific-engineering domain. The remainder of this chapter is predicated on an assumption of objectivity: we assume that a system can be analysed without significantly disturbing it.

Our terminology of control is adopted from Lessig [11]. Our primary contributions are to formally state Lessig's modalities of regulation and to indicate how these controls can influence system design and operation.

1.5 Legal and Normative Controls

Lessig distinguishes the prospective modalities of control from the retrospective modalities. A prospective control is determined and exerted before the event, and has a clear affect on a system's judgement actor or constitution. A retrospective control is determined and exerted after the event, by an external party.

Economic and architectural controls are exerted prospectively, as indicated in the previous section. The owner is a peer in the marketplace which, collectively, defined the optimisation criteria for the judgement actor in their system. The owner was compelled to accept all of the architectural constraints on their system.

The retrospective counterparts of economic and architectural control are respectively *normal control* and *legal control*. The former is exerted by a peerage, and the latter is exerted by a superior. The peerage or superior makes a retrospective judgement after obtaining a report of some alleged behaviour of the owner's system. This judgement is delivered to the owner's system by at least one transmission event, called a *control signal*, from the controlling system to the controlled system. The constitution of a system determines how it responds when it receives a control signal. As noted previously, we leave it to the owner to decide whether any reception event is desirable, undesirable, or inconsequential; and we leave it to the analyst to develop a description of the judgement actor that is predictive of such decisions by the owner.

Judicial and social institutions, in the real world, are somewhat predictable in their behaviour. The analyst should therefore determine whether an owner has made any conscious predictions of legal or social judgements. These predictions should be incorporated into the judgement actor of the system, as architectural constraints or economic criteria.

1.6 Four Types of Security

Having identified four types of control, we are now able to identify four types of security.

Architectural security. A system is architecturally secure if the owner has evaluated the likelihood of a security fault being reported by the system's judgement actor. The owner may take advice from other actors when designing their judgement actor, and when evaluating its likely behaviour. Such advice is called an assurance, as noted in the first paragraph of this chapter. We make no requirement on the expertise or probity of the assuring actor, although these are clearly desirable properties.

Economic security. An economically secure system has an insurance policy consisting of a specification of the set of adverse events (security faults) which are covered by the policy, an amount of compensation to be paid by the insuring party to the owner following any of these adverse events, and a dispute-mediation procedure in case of a dispute over the insurance policy. We include self-insurances in this category. A self-insurance policy needs no dispute-resolution mechanism and consists only of a quantitative risk assessment, the list of adverse events covered by the policy, the expected cost of each adverse event per occurrence, and the expected frequency of occurrence of each event. In the context of economic security, security *risk* has a quantitative definition: it is the annualised cost of an insurance policy. Components of risk can be attached to individual *threats*, that is, to specific types of adversarial activity. Economic security is the natural focus of an actuary or a quantitatively-minded business analyst. Its research frontiers are explored in academic conferences such as the annual Workshop on the Economics of Information Security. Practitioners of economic security are generally accredited by a professional organisation such as ISACA, and use a standardised modelling language such as SysML. There is significant divergence in the terminology used by practitioners [7] and theorists of economic security. We offer our framework as a discipline-neutral common language, but we do not expect it to supplant the specialised terminology that has been developed for use in specific contexts.

Legal security. A system is legally secure if its owner believes it to be subject to legal controls. Because legal control is retrospective, legal security cannot be precisely assessed; and to the extent a future legal judgement has been precisely assessed, it forms an architectural control or an economic control. An owner may take advice from other actors, when forming their beliefs, regarding the law of contracts, on safe-haven provisions, and on other relevant matters. Legal security is the natural focus of an executive officer concerned with legal compliance and legal risks, of a governmental policy-maker concerned with the societal risks posed by insecure systems, and of a parent concerned with the familial risks posed by their children's online activity.

Normative security. A system is normatively secure if its owner knows of any social conventions which might effectively punish them in their role as the owner of a purportedly-abusive system. As with legal security, normative security cannot be assessed with precision. Normative security is the natural province of ethicists, social scientists, policy-makers, developers of security measures which are actively supported by legitimate users, and sociologically-oriented computer scientists interested in the formation, maintenance and destruction of virtual communities.

Readers may wonder, at this juncture, how a service-providing system might be analysed by a non-owning user. This analysis will become possible if the owner has published a model of the behavioural aspects of their system. This published model need not reveal any more detail of the owner's judgement actor and constitution than is required to predict their system's externally-observable behaviour. The analyst should use this published model as an automaton, add a sentient actor representing the non-owning user, and then add an alias of that actor representing their non-owning usage role. This sentient alias is the combined constitutional and judgement actor for a subsystem that also includes the service-providing automaton. The non-owning user's desires and fears, relative to this service provision, become the requirements in the judgement actor.

1.7 Types of Feedback and Assessment

In this section we explore the notions of trust and distrust in our framework. These are generally accepted as important concepts in secure systems, but their meanings are contested. We develop a principled definition, by identifying another conceptual dichotomy. Already, we have dichotomised on the dimensions of temporality (retrospective vs. prospective) and power relationship (hierarchical vs. peer), in order to distinguish the four types of system control and the corresponding four types of system security. We have also dichotomised between function and security, on a conceptual dimension we call *feedback*, with opposing poles of *positive feedback* for functionality and *negative feedback* for security.

Our fourth conceptual dimension is *assessment*, with three possibilities: *cognitive assessment*, *optimistic non-assessment*, and *pessimistic non-assessment*. We draw our inspiration from Niklas Luhmann [12], a prominent social theorist. Luhmann asserts that modern systems are so complex that we must use them, or refrain from using them, without making a complete examination of their risks, benefits and alternatives.

The distinctive element of trust, in Luhmann's definition, is that it is a reliance without a careful examination. An analyst cannot hope to evaluate trust with any accuracy by querying the owner, for the mere posing of a question about trust is likely to trigger an examination and thereby reduce trust dramatically. If we had a reliable calculus of decision-making, then we could quantify trust as the irrational portion of an owner's decision to continue operating a system. The rational portion of this decision is their security and functional assessment. This line of thought motivates the following definitions.

To the extent that an owner has not carefully examined their potential risks and rewards from system ownership and operation, but “do it anyway”, their system is *trusted*. Functionality and security requirements are the result of a cognitive assessment, respectively of a positive and negative feedback to the user. Trust and distrust are the results of some other form of assessment or non-assessment which, for lack of a better word, we might call intuitive. We realise that this is a gross oversimplification of human psychology and sociology. Our intent is to categorise the primary attributes of a secure system, and this includes giving a precise technical meaning to the contested terms “trust” and “distrust” within the context of our framework. We do not expect that the resulting definitions will interest psychologists or sociologists; but we do hope to clarify future scientific and engineering discourse about secure systems.

Mistrust is occasionally defined as an absence of trust, but in our framework we distinguish a distrusting decision from a trusting decision. When an owner distrusts, they are deciding against taking an action, even though they haven’t analysed the situation carefully. The distrusting owner has decided that their system is “not good” in some vaguely apprehended way. By contrast, the trusting owner thinks or feels, vaguely, that their system is “not bad”.

The dimensions of temporality and relationship are as relevant for trust, distrust, and functionality as they are for security. Binary distinctions on these two dimensions allow us to distinguish four types of trust, four types of distrust, and four types of functionality.

We discuss the four types of trust briefly below. Space restrictions preclude any detailed exploration of our categories of functionality and distrust.

1. An owner places *architectural trust* in a system to the extent they believe it to be lawful, well-designed, moral, or “good” in any other way that is referenced to a superior power. Architectural trust is the natural province of democratic governments, religious leaders, and engineers.
2. An owner places *economic trust* in a system to the extent they believe its ownership to be a beneficial attribute within their peerage. The standing of an owner within their peerage may be measured in any currency, for example dollars, by which the peerage makes an invidious distinction. Economic trust is the natural province of marketers, advertisers, and vendors.
3. An owner places *legal trust* in a system to the extent they are optimistic that it will be helpful in any future contingencies involving a superior power. Legal trust is the natural province of lawyers, priests, and repair technicians.
4. An owner places some *normative trust* in a system to the extent they are optimistic it will be helpful in any future contingencies involving a peerage. Normative trust is the natural province of financial advisors, financial regulators, colleagues, friends, and family.

We explore just one example here. In the previous section we discussed the case of a non-owning user. The environmental requirements of this actor are trusted, rather than secured, to the extent that the non-owning user lacks control over discrepancies between the behavioural model and the actual behaviour of the non-owned system. If the behavioural model was published within a peerage, then the non-owning user might place normative trust in the post-facto judgements of their peerage, and economic trust in the proposition that their peerage would not permit a blatantly false model to be published.

1.8 Alternatives to our classification.

We invite our readers to reflect on our categories and dimensions whenever they encounter alternative definitions of trust, distrust, functionality, and security. There are a bewildering number of alternative definitions for these terms, and we will not attempt to survey them. In our experience, the apparent contradiction is usually resolved by analysing the alternative definition along the four axes of assessment, temporality, power, and feedback. Occasionally, the alternative definition is based on a dimension that is orthogonal to any of our four. More often, the definition is not firmly grounded in any taxonomic system and is therefore likely to be unclear if used outside of the context in which it was defined.

Our framework is based firmly on the owner's perspective. By contrast, the SQuaRE approach is user-centric [1]. The users of a SQuaRE-standard software product constitute a market for this product, and the SQuaRE metrics are all of the economic variety. The SQuaRE approach to economic functionality and security is much more detailed than the framework described here. SQuaRE makes clear distinctions between the internal, external, and quality-in-use (QIU) metrics of a software component that is being produced by a well-controlled process. The internal metrics are evaluated by *white-box testing* and the external metrics are evaluated by *black-box testing*. In black-box testing, the judgements of a (possibly simulated) end-user are based solely on the normal observables of a system, i.e. on its transmission events as a function of its workload. In white-box testing, judgements are based on a subset of all events occurring within the system under test. The QIU metrics are based on observations and polls of a population of end-users making normal use of the system. Curiously, the QIU metrics fall into four categories, whereas there are six categories of metrics in the internal and external quality model of SQuaRE. Future theorists of economic quality will, we believe, eventually devise a coherent taxonomic theory to resolve this apparent disparity. An essential requirement of such a theory is a compact description of an important population (a market) of end-users which is sufficient to predict the market's response to a novel good or service. Our framework sidesteps this difficulty, by insisting that a market is a collection of peer systems. Individual systems are modelled from their owner's perspective; and market behaviour is an emergent property of the peered individuals.

In security analyses, behavioural predictions of the (likely) attackers are of paramount importance. Any system that is designed in the absence of knowledge about a marketplace is unlikely to be economically viable; and any system that is designed in the absence of knowledge of its future attackers is unlikely to resist their attacks.

In our framework, system models can be constructed either with, or without, an attacking subsystem. In analytic contexts where the attacker is well-characterised, such as in retrospective analyses of incidents involving legal and normative security, our framework should be extended to include a logically-coherent and complete offensive taxonomy.

Redwine recently published a coherent, offensively-focussed, discussion of secure systems in a hierarchy. His taxonomy has not, as yet, been extended to cover systems in a peerage; nor does it have a coherent and complete coverage of functionality and reliability; nor does it have a coherent and complete classification of the attacker's (presumed) motivations and powers. Even so, Redwine's discussion is valuable, for it clearly identifies important aspects of a offensively-focussed framework. His attackers, defenders, and bystanders are considering their benefits, losses, and uncertainties when planning their future actions [10]. His benefits and losses are congruent with the judgement actors in our framework. His uncertainties would result in either trust or distrust requirements in our framework, depending on whether they are optimistically or pessimistically resolved by the system owner. The lower levels of Redwine's offensive model involve considerations of an owner's purposes, conditions, actions and results. There is a novel element here: an analyst would follow Redwine's advice, within our framework, by introducing an automaton to represent the owner's strategy and state of knowledge with respect to their system and its environment. In addition, the judgement actor should be augmented so that increases in the uncertainty of the strategic actor is a fault, decreases in its uncertainty are functional behaviour, its strategic mistakes are faults, and its strategic advances are functional.

2 Applications

We devote the remainder of this chapter to applications of our model. We focus our attention on systems of general interest, with the goal of illustrating the definitional and conceptual support our framework would provide for a broad range of future work in security.

2.1 Trust Boundaries

System security is often explained and analysed by identifying a set of trusted subsystems and a set of untrusted subsystems. The attacker in such models is presumed to start out in the untrusted portion of the system, and the attacker's goal is to become trusted. Such systems are sometimes illustrated by drawing a *trust boundary* between the untrusted and the trusted portions of the system. An *asset*, such as a valuable good or desirable service, is is accessible only to trusted actors. A bank's vault can thus be modeled as a trust boundary.

The distinguishing feature of a trust boundary is that the system's owner is trusting every system (sentient or automaton) that lies within the trust boundary. A prudent owner will secure their trust boundaries with some architectural, economic, normative, or legal controls. For example, an owner might gain architectural security by placing a sentient guard at the trust boundary. If the guard is bonded, then economic security is increased. To the extent that any aspect of a trust boundary is not cognitively assessed, it is trusted rather than secured.

Trust boundaries are commonplace in our social arrangements. Familial relationships are usually trusting, and thus a family is usually a trusted subsystem. Marriages, divorces, births, deaths, feuds, and reconciliations change this trust boundary.

Trust boundaries are also commonplace in our legal arrangements. For example, a trustee is a person who manages the assets in a legally constituted trust. We would represent this situation in our model with an automaton representing the assets and a constitution representing the trust deed. The trustee is the trusted owner of this trusted subsystem. Petitioners to the trust are untrusted actors who may be given access to the assets of the trust at the discretion of the trustee. Security theorists will immediately recognise this as an access-control system; we will investigate these systems more carefully in the next section.

A *distrust boundary* separates the distrusted actors from the remainder of a system. We have never seen this term used in a security analysis, but it would be useful when describing prisons and security alarm systems. All sentient actors in such systems have an obligation or prohibition requirement which, if violated, would cause them to become distrusted. The judgement actor of the attacking subsystem would require its aliases to violate this obligation or prohibition without becoming distrusted.

A number of *trust-management systems* have been proposed and implemented recently. A typical system of this type will exert some control on the actions of a trusted employee. *Reputation-management systems* are sometimes confused with trust-management systems but are easily distinguished in our framework. A reputation-management system offers its users advice on whether they should trust or distrust some other person or system. This advice is based on the reputation of that other person or system, as reported by the other users of the system. A trust-management system can be constructed from an employee alias, a reputation-management system, a constitutional actor, and a judgement actor able to observe external accesses to a corporate asset. The judgement actor reports a security fault if the employee permits an external actor to access the corporate asset without taking and following the advice of the reputation management system. The employee in this system are architectually trusted, because they can grant external access to the corporate asset. A trust-management system helps a corporation gain legal security over this trust boundary, by detecting and retaining evidence of untrustworthy behaviour.

Competent security architects are careful when defining trust boundaries in their system. Systems are most secure, in the architectural sense, when there is minimal scope for trusted behaviour, that is, when the number of trusted components and people is minimised and when the trusted components and people have a minimal range of permitted activities. However, a sole focus on architectural security is inappropriate if an owner is also concerned about functionality, normative security, economic security, or legal security. A competent system architect will consider all relevant security and functional requirements before proposing a design. We hope that our taxonomy will provide a language in which owners might communicate a full range of their desires and fears to a system architect.

2.2 Data Security and Access Control

No analytic power can be gained from constructing a model that is as complicated as the situation that is being modelled. The goal of a system modeller is thus to suppress unimportant detail while maintaining an accurate representation of all behaviour of interest. In this section, we explore some of the simplest systems which exhibit security properties of practical interest. During this exploration, we indicate how the most commonly-used words in security engineering can be defined within our model.

The simplest automaton has just a single mode of operation: it holds one bit of information which can be read. A slightly more complex single-bit automaton can be modified (that is, written) in addition to being read. An automaton that can only be read or written is a *data element*.

The simplest and most studied security system consists of an automaton (the *guard*), a single-bit read-only data element to be protected by the guard, a collection of actors (users) whom the guard might allow to read the data, and the sentient owner of the system. The trusted subsystem consists of the guard, the owner, and the data. All users are initially untrusted. Users are inferior to the guard. The guard is inferior to the owner.

The guard in this simple *access control system* has two primary responsibilities – to permit authorised reads, and to prohibit unauthorised reads. A guard who discharges the latter responsibility is protecting the *confidentiality* of the data. A guard who discharges the former responsibility is protecting the *availability* of the data.

Confidentiality and availability are achievable only if the guard distinguishes authorised actors from unauthorised ones. Most simply, a requesting actor may transmit a secret word (an *authorisation*) known only to the authorised actors. This approach is problematic if the set of authorised users changes over time. In any event, the authorised users must be trusted to keep a secret. The latter issue can be represented by a model in our framework. A data element represents the shared secret, and each user has a private access control system to protect the confidentiality of an alias of this secret. User aliases are inferiors of the guard in the primary access control system. An adversarial actor has an alias inferior to the guard in each access control system. The adversary can gain access to the asset of the primary access control system if it can read the authorising secret from any authorised user's access control system. An analysis of this system will reveal that the confidentiality of the primary system depends on the confidentiality of the private access control systems. The owner thus has a trust requirement if any of these confidentiality requirements is not fully secured.

In the most common implementation of access control, the guard requires the user to present some *identification*, that is, some description of its owning human or its own (possibly aliased) identity. The guard then consults an *access control list* (another data element in the trusted subsystem) to discover whether this identification corresponds to a currently-authorised actor. A guard who demands identification will typically also demand *authentication*, i.e. some proof of the claimed identity. A typical taxonomy of authentication is "what you know" (e.g. a password), "what you have" (e.g. a security token possessed by the human controller of the aliased user), or "who you are" (a biometric measurement of the human controller of the aliased user). None of these authenticators is completely secure, if adversaries can discover secrets held by users (in the case of what-you-know), steal or reproduce physical assets held by users (in the case of what-you-have), or mimic a biometric measurement (in the case of who-you-are). Furthermore, the guard may not be fully trustworthy. Access control systems typically include some additional security controls on their users, and they may also include some security controls on the guard.

A typical architectural control on a guard involves a trusted recording device (the *audit recorder*) whose stored records are periodically reviewed by another trusted entity (the *auditor*). Almost two thousand years ago, the poet Juvenal pointed out an obvious problem in this design, by asking "quis custodiet ipsos custodes" (who watches the watchers)? Adding additional watchers, or any other entities to a trusted subsystem will surely increase the number of different types of security fault but may nonetheless be justified if it offers some overall functional or security advantage.

Additional threats arise if the owner of a data system provides any services other than the reading of a single bit. An *integrity* threat exists in any system where the owner is exposed to loss from unauthorised writes. Such threats are commonly encountered, for example in systems that are recording bank balances or contracts.

Complex threats arise in any system that handle multiple bits, especially if the meaning of one bit is affected by the value of another bit. Such systems provide *meta-data services*. Examples of meta-data include an author's name, a date of last change, a directory of available data items, an authorising signature, an assertion of accuracy, the identity of a system's owner or user, and the identity of a system. Meta-data is required to give a context, and therefore a meaning, to a collection of data bits. The performance of any service involving meta-data query may affect the value of a subsequent meta-data query. Thus any provision of a meta-data service, even a meta-data read, may be a security threat.

If we consider all meta-data services to be potential integrity threats, then we have an appealingly short list of security requirements known as the CIA triad: confidentiality, integrity, and availability. Any access control system requires just a few security-related functions: identification, authentication, authorisation, and possibly audit. This range of security engineering is called *data security*. Although it may seem extremely narrow, it is of great practical importance. Access control systems can be very precisely specified (e.g. [9]), and many other aspects have been heavily researched [6]. Below, we attempt only a very rough overview of access control systems.

The *Bell-La Padula* (BLP) structure for access control has roles with strictly increasing levels of read-authority. Any role with high authority can read any data that was written by someone with an authority no higher than themselves. A role with the highest authority is thus able to read anything, but their writings are highly classified. A role with the lowest authority can write freely, but can read only unclassified material. This is a useful structure of access control in any organisation whose primary security concern is secrecy. Data flows in the BLP structure are secured for confidentiality. Any data flow in the opposite direction (from high to low) may either be trusted, or it may be secured by some non-BLP security apparatus [13].

The *Biba* structure is the dual, with respect to read/write, of the BLP structure. The role with highest Biba authority can write anything, but their reads are highly restricted. The Biba architecture seems to be mostly of academic interest. However, it could be useful in organisations primarily concerned with publishing documents of record, such as judicial decisions. Such documents should be generally readable, but their authorship must be highly restricted.

In some access control systems, the outward-facing guard is replaced by an inward-facing *warden*, and there are two categories of user. The prisoners are users in possession of a secret, and for this reason they are located in the trusted portion of the system. The outsiders are users not privy to the secret. The warden's job is to prevent the secret from becoming known outside the prison walls, and so the warden will carefully scrutinise any write operations that are requested by prisoners. Innocuous-looking writes may leak data, so a high-security (but low-functionality) prison is obtained if all prisoner-writes are prohibited.

The *Chinese wall* structure is an extension of the prison, where outsider reads are permitted, but any outsider who reads the secret becomes a prisoner. This architecture is used in financial consultancy, to assure that a consultant who is entrusted with a client's sensitive data is not leaking this data to a competitor who is being assisted by another consultant in the same firm.

2.3 Miscellaneous Security Requirements

The fundamental characteristic of a secure system, in our definition, is that its owner has cognitively assessed the risks that will ensue from their system. The fundamental characteristic of a functional system is that its owner has cognitively assessed the benefits that will accrue from their system. We have already used these characteristics to generate a broad categorisation of requirements as being either security, functional or mixed. This categorisation is too broad to be very descriptive, and additional terminology is required.

As noted in the previous section, a system's security requirements can be sharply defined if it offers a very narrow range of simple services, such as a single-bit read and write. Data systems which protect isolated bits have clear requirements for confidentiality, integrity, and availability.

If an audit record is required, we have an *auditability* requirement. If a user or owner can delegate an access right, then these delegations may be secured, in which case the owner would be placing a *delegatibility* requirement on their system. When an owner's system relies on any external system, and if these reliances can change over time, then the owner might introduce a *discoverability* requirement to indicate that these reliances must be controlled. We could continue down this path, but it seems clear that the number of different requirements will increase whenever we consider a new type of system.

2.4 Negotiation of Control

In order to extend our four-way taxonomy of requirements in a coherent way, we consider the nature of the signals that are passed from one actor to another in a system. In the usual taxonomy of computer systems analysis, we would distinguish data signals from control signals. Traditional analyses in data security are focussed on the properties of data. Our framework is focussed on the properties of control. Data signals should not be ignored by an analyst, however we assert that data signals are important in a security analysis only if they can be interpreted as extensions or elaborations of a control signal.

Access control, in our framework, is a one-sided negotiation in which an inferior system petitions a superior system for permission to access a resource. The metaphor of access control might be extended to cover most security operations in a hierarchy, but a more balanced form of intersystem control occurs in our peerages.

Our approach to control negotiations is very simple. We distinguish a service provision from a non-provision of that service. We also distinguish a forbiddance of either a provision or a non-provision, from an option allowing a freedom of choice between provision or a non-provision. These two distinctions yield four types of negotiated controls. Below, we discuss how these distinctions allow us to express access control, contracts between peers, and the other forms of control signals that are transmitted commonly in a hierarchy or a peerage.

An *obligation* requires a system to provide a service to another system. The owner of the first system is the *debtor*; the owner of the second system is a *creditor*; and the negotiating systems are authorised to act as agents for the sentient parties who, ultimately, are contractual parties in the legally or normatively enforced contract which underlies this obligation. A single service provision may suffice for a complete discharge of the obligation, or multiple services may be required.

Formal languages have been proposed for the interactions required to negotiate, commit, and discharge an obligation [2, 3, 5]. These interactions are complex and many variations are possible. The experience of UCITA in the US suggests that it can be difficult to harmonise jurisdictional differences in contracts, even within a single country. Clearly, contract law cannot be completely computerised, because a sentient judiciary is required to resolve some disputes. However an owner may convert any predictable aspect of an obligation into an architectural control. If all owners in a peerage agree to this conversion, then the peerage can handle its obligations more efficiently. Obligations most naturally arise in peerages, but they can also be imposed by a superior on an inferior. In such cases, the superior can unilaterally require the inferior to use a system which treats a range of obligations as an architectural control.

An *exemption* is an option for the non-provision of a service. An obligation is often accompanied by one or more exemptions indicating the cases in which this obligation is not enforceable; and an exemption is often accompanied by one or more obligations indicating the cases where the exemption is not in force. For example, an obligation might have an exemption clause indicating that the obligation is lifted if the creditor does not request the specified service within one year.

Exemptions are diametrically opposed to obligations on a qualitative dimension which we call *strictness*. The two poles of this dimension are *allowance* and *forbiddance*. An obligation is a forbiddance of a non-provision of service, whereas an exemption is an allowance for a non-provision of service.

The second major dimension of a negotiated control is its *activity*, with poles of *provision* and *non-provision*. A forbiddance of a provision is *prohibition*, and an allowance of a provision is called a *permission*.

A superior may require their inferior systems to obey an obligation with possible exemptions, or a prohibition with possible permissions. An access control system, in this light, is one in which the superior has given a single permission to its inferiors – the right to access some resource. An authorisation, in the context of an access control system, is a permission for a specific user or group of users. The primary purpose of an identification in an access control system is to allow the guard to retrieve the relevant permission from the access control list. An authentication, in this context, is a proof that a claimed permission is valid. In other contexts, authentication may be used as an architectural control to limit losses from falsely-claimed exemptions, obligations, and prohibitions.

We associate a class of requirements with each type of control in our usual fashion, by considering the owner's fears and desires. Some owners desire their system to comply in a particular way, some fear the consequences of a particular form of non-compliance, some desire a particular form of non-compliance, and some fear a particular form of non-compliance. If an owner has feared or desired a contingency, it is a security or functionality requirement. Any unconsidered cases should be classified, by the analyst, as trusted or distrusted gaps in the system's specification depending on whether the analyst thinks the owner is optimistic or pessimistic about them. These gaps could be called the owner's assumptions about their system, but for logical coherence we will call them requirements.

Below, we name and briefly discuss each of the four categories of requirements which are induced by the four types of control signals.

An analyst generates *probity requirements* by considering the owner's fears and desires with respect to the obligation controls received by their system. For example, if an owner is worried that their system might not discharge a specific type of obligation, this is a security requirement for probity. If an owner is generally optimistic about the way their system handles obligations, this is a trust requirement for probity.

Similarly, an analyst can generates *diligence requirements* by considering permissions, *efficiency requirements* by considering exemptions, and *guijuity requirements* by considering prohibitions. Our newly-coined word guijuity is an adaptation from the Mandarin word guiju. This is a Confucian ethic of right action through the following of rules: "GuiJu FangYuan ZhiZhiYe". Guijuity can be understood as the previously unnamed security property which is controlled by the X (execute permission) bit in a Unix directory entry, where the R (read) and W (write) permission bits are controlling the narrower, and much more well-explored, properties of confidentiality and availability. In our taxonomy, guijuity is a broad concept encompassing all prohibitive rules. Confidentiality is a narrower concept, because it is a prohibition only of a particular type of action, namely a data-read.

The confidentiality, integrity, and availability requirements arising in access control systems can be classified clearly in our framework, if we restrict our attention to those access control systems which are implementing data security in a BLP or Biba model. This restriction is common in most security research. In this context, confidentiality and availability are subtypes of guijuity, and availability is a subtype of efficiency. The confidentiality and integrity requirements arise because the hierarch has prohibited anyone from reading or writing a document without express authorisation. The availability requirement arises because the hierarch has granted some authorisations, that is, some exemptions from their overall prohibitions. No other requirements arise because the BLP and Biba models cover only data security, and thus the only possible control signals are requests for reads or writes.

If a system's services are not clearly dichotomised into reads and writes, or if it handles obligations or exemptions, then the traditional CIA taxonomy of security requirements is incomplete. Many authors have proposed minor modifications to the CIA taxonomy in order to extend its range of application. For example, some authors suggest adding authentication to the CIA triad. This may have the practical advantage of reminding analysts that an access-control system is generally required to authenticate its users. However, the resulting list is neither logically coherent, nor is it a complete list of the requirement types and required functions in a secured system.

We assert that all requirements can be discovered from an analysis of a system's desired and feared responses to a control signal. For example, a *non-repudiation* requirement will arise whenever an owner fears the prospect that a debtor will refuse to provide an obligated service. The resulting dispute, if raised to the notice of a superior or a peerage, would be judged in favour of the owner if their credit obligation is non-repudiable. This line of analysis indicates that a non-repudiation requirement is ultimately secured either legally or normally. Subcases may be transformed into either an architectural or economic requirement, if the owner is confident that these subcases would be handled satisfactorily by a *non-repudiation protocol* with the debtor. Essentially, such protocols consist of a creditor's assertion of an obligation, along with a proof of validity sufficient to convince the debtor that it would be preferable to honour the obligation than to run the risks of an adverse legal or normal decision.

We offer one more example of the use of our requirements taxonomy, in order to indicate that probity requirements can arise from a functional analysis as well as from a security analysis. An owner of a retailing system might desire it to gain a reputation for its prompt fulfilment of orders. This desire can be distinguished from an owner's fear of gaining a bad reputation or suffering a legal penalty for being unacceptably slow when filling orders. The fear might lead to a security requirement with a long response time in the worst case. The desire might lead to a functional requirement for a short response time on average. A competent analyst would consider both types of requirements when modeling the judgement actor for this system.

In most cases, an analyst need not worry about the precise placement of a requirement within our taxonomy. The resolution of such worries is a problem for theorists, not for practitioners. Subsequent theoreticians may explore the implications of our taxonomy, possibly refining it or revising it. Our main hope when writing this chapter is that analysts will be able to develop more complete and accurate lists of requirements by considering the owner's fears and desires about their system's response to an obligation, exemption, prohibition, or permission from a superior, inferior, or peer.

3 Dynamic, Collaborative, and Future Secure Systems

The data systems described up to this point in our exposition have all been essentially static. The population of users is fixed, the owner is fixed, constitutional actors are fixed, and judgement actors are fixed. The system structure undergoes, at most, minor changes such as the movement of an actor from a trusted region to an untrusted region.

Most computerised systems are highly dynamic, however. Humans take up and abandon aliases. Aliases are authorised and de-authorised to access systems. Systems are created and destroyed. Sometimes systems undergo uncontrolled change, for example when authorised users are permitted to execute arbitrary programs (such as applets encountered when browsing web-pages) on their workstations. Any uncontrolled changes to a system may invalidate its assessor's assumptions about system architecture. Retrospective assessors in legal and normative systems may be unable to collect the relevant forensic evidence if an actor raises a complaint or if the audit-recording systems were poorly designed or implemented. Prospective assessors in the archectural and economic systems may have great difficulty predicting what a future adversary might accomplish easily, and their predictions may change radically on the receipt of additional information about the system, such as a bug report or news of an exploit.

In the *Clark-Wilson* model for secure computer systems, any proposed change to the system as a result of a program execution must be checked by a guard before the changes are committed irrevocably. This seems a very promising approach, but we are unaware of any full implementations. One obvious difficulty, in practice, will be to specify important security constraints in such a way that they can be checked quickly by the guard. Precise security constraints are difficult to write even for simple, static systems. One notable exception is a standalone database systems with a static data model. The guard on such a system can feasibly enforce the *ACID* properties: atomicity, consistency, isolation, and durability. These properties ensure that the committed transactions are not at significant risk to threats involving the loss of power, hardware failures, or the commitment of any pending transactions. These properties have been partly extended to distributed databases. There has also been some recent work on defining privacy properties which, if the database is restricted in its updates, can be effectively secured against adversaries with restricted deductive powers or access rights.

Few architectures are rigid enough to prevent adverse changes by attackers, users, or technicians. Owners of such systems tend to use a modified form of the Clark-Wilson model. Changes may occur without a guard's inspection. However if any unacceptable changes have occurred, the system must be restored ("rolled back") to a prior untainted state. The system's environment should also be rolled back, if this is feasible; alternatively, the environment might be notified of the rollback. Then the system's state, and the state of its environment, should be rolled-forward to the states they "should" have been in at the time the unacceptable change was detected. Clearly this is an infeasible requirement, in any case where complete states are not retained and accurate replays are not possible. Thus the Clark-Wilson apparatus is typically a combination of filesystem backups, intrusion detection systems, incident investigations, periodic inspections of hardware and software configurations, and ad-hoc remedial actions by technical staff whenever they determine (rightly or wrongly) that the current system state is corrupt. The design, control, and assessment of this Clark-Wilson apparatus is a primary responsibility of the IT departments in corporations and governmental agencies.

We close this chapter by considering a recent set of guidelines, from The Jericho Forum, for the design of computing systems. These guidelines define a *collaboration oriented architecture* or COA [4]. Explicit management of trusting arrangements are required, as well as effective security mechanisms, so that collaboration can be supported over an untrusted internet between trusting enterprises and people. In terms of our model, a COA is a system with separately-owned subsystems. The subsystem owners may be corporations, governmental agencies, or individuals. People who hold an employee role in one subsystem may have a trusted-collaborator role in another subsystem, and the purpose of the COA is to extend appropriate privileges to the trusted collaborators. We envisage a desirable COA workstation as one which helps its user keep track of and control the activities of their aliases. The COA workstation would also help its user make good decisions regarding the storage, transmission, and processing of all work-related data.

The COA system must have a *service-oriented architecture* as a subsystem, so that its users can exchange services with collaborators both within and without their employer's immediate control. The collaborators may want to act as peers, setting up a service for use within their peerage. Thus a COA must support peer services as well as the traditional, hierarchical arrangement of client-server computing. An *identity management* subsystem is required, to defend against impersonations and also for the functionality of making introductions and discoveries. The decisions of COA users should be trusted, within a broad range, but security must be enforced around this trust boundary.

The security and functionality goals of trustworthy users should be enhanced, not compromised, by the enforcement of security boundaries on their trusted behaviour. In an automotive metaphor, the goal is thus to provide air bags rather than seat belts. Regrettably, our experience of contemporary computer systems is that they are either very insecure, with no effective safety measures; or they have intrusive architectures, analogous to seat belts, providing security at significant expense to functionality. We hope this chapter will help future architects design computer systems which are functional and trustworthy for their owners and authorised users.

References

1. M. Azuma. SQuaRE: The next generation of the ISO/IEC 9126 and 14598 international standards series on software product quality. In *Project Control: Satisfying the Customer (Proceedings of ESCOM 2001)*, pages 337–346. Shaker Publishing BV, 2001.
2. C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera. Provisions and obligations in policy management and security applications. In *Proc. of the 28th Int'l Conf. on Very Large Databases*, pages 502–513, 2002.
3. A. D. H. Farrell, M. J. Sergot, M. Sallé, and C. Bartolini. Using the event calculus for tracking the normative state of contracts. *Int. J. Cooperative Inf. Syst.*, 14(2-3):99–129, 2005.
4. P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Requirements engineering for trust management: model, methodology, and reasoning. *Int. J. Inf. Sec.*, 5(4):257–274, 2006.
5. D. Gollman. Security models. In K. de Leeuw and J. Bergstra, editors, *The History of Information Security: A Comprehensive Handbook*. Elsevier, 2007.
6. R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley-Interscience, 1991.
7. S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *IEEE Symposium on Security and Privacy*, pages 31–42, 1997.
8. S. T. R. Jr. Towards an organization for software system security principles and guidelines, version 1.0. Technical Report 08-01, Institute for Infrastructure and Information Assurance, James Madison University, Feb. 2008.
9. L. Lessig. *Code version 2.0.* Basic Books, 2006.
10. N. Luhmann. *Trust and Power.* Wiley, 1979. English translation by Howard Davis et al.

11. R. O'Brien and C. Rogers. Developing applications on LOCK. In *Proc. 14th National Security Conference*, pages 147–156, Washington, D.C., 1991.
12. The Jericho Forum. Position paper: Collaboration oriented architectures, Apr. 2008.
13. The Open Group. Risk taxonomy, Technical standard C081, Jan. 2009.

Index