What Is the Shape of Your Security Policy? Security as a Classification Problem

Sven Türpe
Fraunhofer Institute for Secure Information Technology (SIT)
Rheinstrasse 75
64295 Darmstadt, Germany
sven.tuerpe@sit.fraunhofer.de

ABSTRACT

This new paradigm defines security policies on cause-effect relations and models security mechanisms in analogy with pattern recognition classifiers. It augments the arsenal of formal computer security evaluation tools with new techniques. A causality model represents possible causes and effects; the causes include threats and the effects may be undesired. Target security policies derived from functional specifications select permitted causalities. Security mechanisms extract features from causes and effects and enforce mechanism-specific policies, approximating the target policy. Advantages of the classifier paradigm are the ability to generalize from incomplete information and examples, to measure classification error and mechanism performance, and to analyze mechanism ensembles and compositions. The classifier paradigm also offers a conception of problem complexity and suggests paying more attention to the impact of mechanisms rather than to their inner workings.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*access controls, information flow controls*; I.5.m [**Pattern Recognition**]: Miscellaneous; F.3.m [**Logics and Meanings of Programs**]: Miscellaneous

General Terms

Security, Theory

Keywords

Security policy, security analysis, classifier system, high-dimensional space, secure composition, defense in depth, threat model, causality

1. INTRODUCTION

We know two fundamentally different ways of evaluating the security of a system: formal, mechanism-centric evaluation and hacking. The most prominent representative of mechanism-centric evaluation is the Common Criteria [5]. A brief overview of the Common Criteria approach along with a critique of the approach can be found in [2]. The basic idea of the Common Criteria is to define a set of security mechanisms and requirements for each mechanism. One must provide a rationale for each of the mechanisms by relating it to a set of assumed threats. Security criteria are specified for each mechanism. To evaluate a system one uses the frame thus created and validates whether the system actually implements it. One problem with this approach is that one easily loses sight of relevant threats and security considerations once they have been excluded from the frame. The target of evaluation is not being validated against threats; it is being validated against an arbitrary frame which is merely assumed to represent the actual security requirements.

The other end of the spectrum is marked by hacking. A hacker, be it as an actual attacker or as white-hat security tester, does not care much about formalism regarding some security design. A hacker has certain capabilities, particularly capabilities of interacting with a target. The hacker also has a goal or a set of potential goals which, if achievable, would imply that the system is vulnerable to some attack. It is the task of the security mechanisms in a system to constrain the hacker's capabilities in such a way that none of the goals can be achieved. The hacker looks for situations where this is not the case. A problem with the hacker's approach is that it is not easily formalized.

The fundamental difference between these two approaches is emphasized by the fact that the same system can be certified under the Common Criteria and vulnerable to rather simple and effective attacks.

As a case in point, the Xerox WorkCentre 2xx series[1] multifunctional printer received a Common Criteria certificate on April 6th, 2006 [18, 19]. On August 3rd the same year, at the BlackHat USA conference, Brendan O'Connor presented several ways of compromising the security of just this type of device.[2] At least 4 serious vulnerabilities are known for the exact version certified,[3] and 19 more were reported for different versions but likely exist in the certified version as

[1]The WorkCentre 2xx series comprises the WorkCentre/WorkCentre Pro 232/238/245/255/265/275 models.
[2] *Vulnerabilities in Not-So-Embedded Systems* was the title of his talk.
[3]The certified versions of the system software are 12.027.24.015, 13.027.24.015, and 14.027.24.015, depending on the model within the series. Known vulnerabilities for this exact version are CVE-2006-0825 through CVE-2006-0828.

well.[4] In both categories there are vulnerabilities with CVSS ratings up to 10, which is the maximum rating and roughly means instant remote `root` access for everyone who desires it and knows the trick.

It is alluring to blame this failure of assessment to the low assurance level of the security evaluation. The WorkCentre series was evaluated to EAL 2, which is the second-lowest of the seven assurance levels defined by the Common Criteria framework. Formal verification and testing is required only at the upper end of the scale. However, choosing a higher assurance level would not likely have solved the problem. The WorkCentre is essentially a Linux system with a scanner and a printer attached to it, running a Web server and various other services. But only a small set of security functions was evaluated, ignoring those parts of the system that turned out to be vulnerable. This case study suggests that today we have elaborate techniques for evaluating only half the security factors. It has long been known that security tends to fail outside the core mechanism [1]. Just so do our analysis and evaluation techniques.

Since the ultimate goal of information security is to prevent relevant attacks, new techniques for security analysis are needed. They must be sufficiently formal to produce reliable statements about the security of a system but they must also take into account how attacks work in practice. The formal methods known to date seem to be located more towards the Common Criteria end of the spectrum.

The approach proposed in this paper is to extend and generalize the mechanism-centric view and to relate it to a model of the real world representing legitimate actions as well as attacker capabilities. When analyzing for instance an access control mechanism, we would attempt to understand who will be allowed to do what to which object under this mechanism. We can extend and generalize this question: what will the suite of security mechanisms in a system allow the environment to do to the system, and what will be the effects?

1.1 Paradigm Outline

The new paradigm roots in a single proposition: pattern recognition classifiers are a suitable conceptual metaphor for security mechanisms, lending concepts and analysis techniques to security. Applying this metaphor requires further ingredients. The paradigm thus comprises four key concepts:

- *A world model* representing causes and effects. This world model serves as a reference frame for defining target policies. The purpose and environment of a system determines the causes and effects that matter.
- *Permissions on cause-effect pairs* as security policies. While the world model represents possible causalities, a security policy selects those that one wishes to permit.
- *The business logic* of the system, or a specification or model of this business logic. The business logic implies a security policy.
- *Security mechanisms as classifiers*. A security mechanism takes cause-effect pairs as inputs, maps them into its own feature space, and decides within this space whether to accept an input or not.

Pattern recognition classifiers offer techniques for dealing with incomplete or noisy information. The above set of concepts makes these techniques applicable to the security analysis of systems.

1.2 Rationale

Security mechanisms prevent causes from having effects. Some causes from having some effects, to be precise. Causes and effects exist in the real world; the business logic of a computer system along with the semantics of inputs and outputs determines which causes lead to which effects. We might call a system secure if it abides by a defined set of permitted cause-effect relationships—a target security policy.

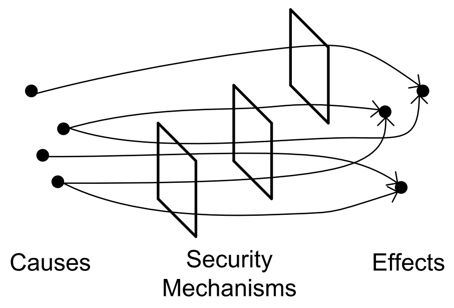

Figure 1: Defense in depth

Individual security mechanisms rarely enforce the target security policy of a system. They rather enforce mechanism-specific policies. Yet they do prevent certain causes from having effects altogether, certain effects from occurring regardless of their cause, or certain causes from having certain effects. So does the business logic of a system, which constitutes an implicit security mechanism by itself and requires additional mechanisms for any issue it cannot handle alone. Figure 1 illustrates this mental model.

Security analysis tries to answer the question whether or to which extent a system with all its business logic, security mechanisms, software defects, and other properties, enforces a given target policy—or which policy a system enforces at all.

1.3 Paper Organization

The remainder of this paper is organized as follows. Section 2 gives some background, describes the general model underlying considerations in this paper, gives a rationale for modeling security mechanisms as classifiers, and briefly introduces classifiers as they are used in pattern recognition. Section 3 semi-formally defines key concepts and notions, which section 4 illustrates using an example. Section 5 discusses possible analysis techniques that the classifier analogy might yield. The paper closes with concluding remarks on expected advantages, caveats, and questions for further research in section 6.

[4]Vulnerabilities reported for versions up to a higher version number are: CVE-2006-6427 through CVE-2006-6434, CVE-2006-6436 through CVE-2006-6438, and CVE-2006-6467 through CVE-2006-6473. For CVE-2006-5290 the version numbers affected remain unknown.

2. SECURITY AS A CLASSIFICATION PROBLEM

2.1 Background

Security once seemed easy, and still does so in some textbooks. Information was classified according to a policy. The policy followed from obvious security objectives like confidentiality. All the security mechanisms of a system had to do was to enforce the policy. Early developments in information and computer security were primarily concerned with ways of specifying and enforcing security policies [4]. Security policies were largely predetermined by concepts and regulations that independently existed outside the systems, as was the classification of information according to these policies. This has led to a number of useful and well-understood formal security models as well as mechanisms for their enforcement.

Soon it was noticed, however, that *"a wide range of considerations are pertinent to the engineering of security of information,"* and that *"The objective of a secure system is to prevent all unauthorized use of information, a negative kind of requirement."* (both quotes from Saltzer and Schroeder [14]). Many formal security policy models evade this wide range of considerations by making extensive use of assumptions, which are often not met by the systems we are dealing with today.

The security problems and our responses to them have coevolved since. Today's systems incorporate a large number of different security mechanisms reaching far beyond the relatively simple world of user authentication and access control. A typical operating system today comprises, among others, network firewall functions, virus scanning, and security functions to prevent buffer overflow defects from being exploited. Applications add their own specific measures, such as various filters for untrusted data to prevent injection attacks. Many of these mechanisms are responses to actual attacks that exploit defects anywhere in the software to circumvent security functions.

Along with this evolution of attacks and protection, security mechanisms have lost the strong external support that they once got from well-defined security policies. Security mechanisms no longer merely enforce a clear and simple security policy on the safe grounds of valid assumptions about the system and its environment. Increasingly, the suite of security mechanisms in a system is expected and required to avert arbitrary attacks while permitting any legitimate uses, without a precise and complete specification of either one.

As a consequence, we need techniques to analyze compositions of security mechanisms: the interplay of mechanisms, the limitations and vulnerabilities, and the effects of adding yet another patch. This paper explores one candidate framework, the theory of classifiers. Classifiers are a key concept in pattern recognition and machine learning and have been thoroughly examined there. I argue that the classifier is a meaningful abstraction of security mechanisms or, more precisely, important aspects of security mechanisms. Describing mechanisms in terms of their classification properties allows us to analyze the properties of a composition of mechanisms as we see it in computer systems today.

It seems that we have an intuitive idea of what is a "bad" versus a "good" approach to improving the security of a system or fixing a known issue. An example may illustrate this. One of the many proposals about how to fight the growing problem of phishing attacks was to introduce a new top-level domain, *.bank*, reserved exclusively for financial institutions [8]. This proposal has been criticized on various grounds and has not been implemented so far. The most compelling argument against it is also the simplest: it does not add anything to the overall security system (except for another price tag). Whichever semantics the use of such a domain may convey is already reliably encoded in SSL certificates, or could be encoded there. This implies that wherever and however the security system—which includes here the user making security decisions—fails to distinguish phishing attacks from legitimate interaction, the new domain would not substantially change anything. To outline a framework for this kind of reasoning is the purpose of this paper.

This paper does not propose to use machine learning techniques as a security mechanism, which has been tried on various occasions. Rather, the idea is to borrow concepts from this discipline and see how they might be developed into a security model. This paper does also not propose to classify attacks or vulnerabilities according to some scheme.

2.2 General Model

Conceptually we can conceive of a system as comprising business logic and a security system. The business logic is responsible for the specified functioning of the system. The security system has to prevent malicious interaction with the system from having an impact on the business logic or the environment. Figure 2 depicts this model.

The rationale for choosing this model is threefold. First, the model is generic and does not refer to any specific security objectives or mechanisms. The model is therefore unlikely to impose restrictions right away. Second, the model corresponds to (some) system architectures. Input into a system often has to pass through one or multiple layers performing security tasks before it will be processed: firewalls, input filters, access control, etc. The model is thus likely to be accurate at least in describing some aspects of some systems. Third, the model is consistent with the concept of a *protected subsystem* as described by Saltzer and Schroeder [14].

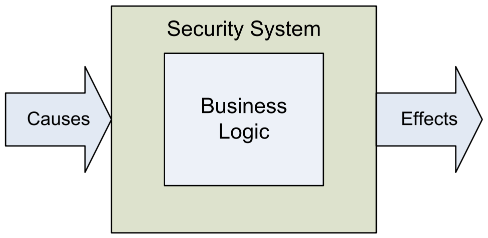

Figure 2: General model

For the sake of simplicity, let us assume all security-related functionality and processing be part of the security system. This is a simplification; in any real system there is no strict separation of security functions from business logic. Such a separation cannot even exist. Faults in the business logic may be exploitable to malicious ends, and some security functions may be designed and implemented as an integral part of the system rather than as a self-contained component. However, we can still make the conceptual distinction between aspects of a system that constitute the intended

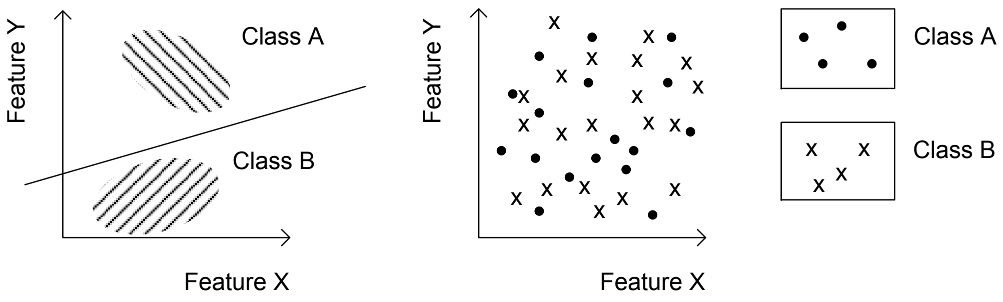

Figure 3: A simple and a complex classification problem

functionality of the system and those that are to be considered primarily for security reasons. While the distinction between business logic and security system may seem somewhat arbitrary, we may define the business logic as everything in the system that we ignore when thinking about security. Note that the model, although it may seem so, is not limited to activity within the boundaries of a physical system.

Within this high-level model there is only one single security objective: out of all possible inputs and other interactions with the system, the security system has to identify those that are safe to be processed by the business logic. This implies two requirements for the security system, completeness and accuracy. Completeness means that the security system covers all relevant inputs or interactions so it cannot be circumvented by a malicious attacker. Accuracy means that the security system reliably rejects those that would lead to unintended results. Unintended by the operator of the system, to be precise, as an attacker has the opposite intention. The security system can thus be conceived of as a classifier: its task is to separate the permitted from the forbidden.

2.3 Classification in a Nutshell

Classification is a well-known concept in computer security, but this paper uses the term in the way the pattern recognition [6, 9] community does. A pattern, or class is an entity that can be named. To recognize a pattern means to decide which one out of a set of patterns a vector of input data most likely represents. This is what a generic classifier does. It takes input vectors from $\mathbb{R}^n$ or a subset of $\mathbb{R}^n$ and maps them into a discrete and usually finite set of classes. The input vectors often originate from analog sources and sensors capture and digitize these data for further processing. This is, however, not strictly required. The domain of a classifier may comprise nominal and discrete dimensions. For typical classification problems the range of the classifier is much smaller than its domain. There are several approaches to pattern recognition, for instance statistical classification, neural networks, template matching, or fuzzy logic approaches.

Optical character recognition is an example of a classification task. Inputs to the classifier in this task are two-dimensional pictures of printed characters. These inputs are noisy, which can be modeled using probability distributions. The more noisy the data are, the more difficult the task gets. Even without the noise, some pairs of characters are more similar than others, which makes the task more challenging. The expected output from the classifier is the abstract character that has been recognized or, in statistical terms, the label that is most likely correct for the input data. Instead of a single most likely label, some classifiers output probabilities of class membership or a list of classes ordered by their probability of correctness.

In most classification tasks one does not simply use raw input data as they are provided by sensors. Rather, a preprocessing step is added, which extracts features from the raw data. Consider biometrics for example. To recognize faces or fingerprints, one does not simply compare images but rather extracts features such as the distance between eyes or minutiae. The classification function then works with multiple such features represented in a feature space. Features may be discrete or continuous.

Classification problems often involve large numbers of input variables. To make a classifier more efficient, one attempts to reduce the number of variables—and thus, feature space dimensions—by picking those that do actually contribute to the classification, i.e. help distinguish members of different classes. A number of statistical analysis methods are available to this end.

While one might construct classifiers using pen, paper and brains, one typically uses sample patterns. The general idea is that the probability distribution of all possible inputs can be derived, to some degree of certainty, from these samples. The classifier is parameterized to represent the distribution of the samples, and expected to correctly classify most other inputs if they follow the same distribution. There are two distinct ways of approaching this learning task, supervised and unsupervised. In supervised learning one has a predetermined set of labels, such as the abstract characters in character recognition, and the target class of each sample (or training) pattern is known. Unsupervised methods on the other hand work with just the samples, and the learning algorithm has to determine a set of classes that is suitable as an abstract representation of the distribution of the inputs. Unsupervised leads to classifiers that summarize their input data.

No matter how it is being trained and parameterized, in the end the classifier needs to somehow represent the target classes within the feature space in order to classify elements of this space. It can do so e.g. by representing probability densities, decision boundaries, or in some cases by templates or rule sets. This is where classification problems differ in their characteristics and complexity [3], and a suitable type of classifier must be chosen that is capable of solving the problem. The simplest classifiers are linear, separating classes by hyperplanes in the feature space. This requires that classes are, at least approximately, linearly separable. Figure 3 illustrates a linearly versus a more complex classification problem.

There are two common ways of dealing with the more complex problems. One is to use a classifier capable of representing more complex subsets of the feature space, e.g. a non-linear instead of a linear classifier. The other approach is to approximate complex decision functions by combining several simpler classifiers [10, 11, 12]. The final decision is then made by combining the outputs of all classifiers according to some rule, e.g. majority voting. The individual classifiers may work simultaneously on the same input data, or one can go one step further and combine classifiers working on different feature spaces derived from the same raw input data.

To sum up, the key task in classification is to find a classification function that is a good approximate solution of the particular classification problem at hand. For such search problems one can show that *"For all possible performance measure, no search algorithm is better than another when its performance is averaged over all possible discrete functions."* [16] summarizing [17]. But this *No Free Lunch* theorem seems not to imply that all attempts at optimization are futile. In particular, problem-specific considerations may lead to better solutions than a blind heuristic search process, for which the theorem holds [16].

A conception loosely related to classification, and possibly more suitable for dealing with security problems than statistical methods, is formal concept analysis [13]. Formal concept analysis, roughly, deals with classes of objects that are defined by attributes, and with lattices formed by such classes.

2.4 Security Mechanisms are Classifiers

It should be obvious that many common security mechanisms can be described as classifiers or have important aspects that fit into the framework of classification. A security mechanism has to decide what is permitted and what is not, and enforce that decision. A few examples:

User authentication Requests to a system—usually the request to open a user session—are to be classified to one out of a set of system users or to the *invalid user* class. The features being used vary.

Firewalls A firewall classifies packets or streams of packets passing it on a network. The features being used include packet contents as well as contextual information such as the interface through which a packet was received.

Access control Access control mechanisms map tuples of subjects, actions, objects and possibly further parameters to the classes *allowed* or *denied*.

Antivirus software An antivirus program attempts to classify programs to identify those that are malicious.

Emphasis may be on one aspect or the other, classification or enforcement. In the examples above, enforcement is achieved through program routines that enforce decisions as long as they cannot be circumvented by manipulating the program itself. Encryption as a mechanism is different: there is no code to make a decision, but security properties are strongly enforced through mathematics under certain side conditions. Yet we can describe even this mechanism as an implicit classifier. Those in possession of the key can access the clear text, all others cannot.

Looking at individual mechanisms from a classification point of view does not promise much new insight. Most of the considerations from pattern recognition seem irrelevant since security mechanisms work on structured digital data rather than analog information. A vast amount of research is available for each common type of security mechanism, including design and evaluation criteria.

However, the classifier point of view might be helpful when considering a security system, the multitude of security mechanisms built into a system to fulfill its security objectives. Viewing all mechanisms as classifiers may provide an abstract layer on which we can analyze the interworking of the mechanisms, the degree to which the security system is capable of fulfilling security objectives, and the impact of modifications.

Security properties of a mechanism or of configurations of mechanism may be undecidable, e.g. [7]. This fundamentally limits formal analysis of mechanisms and their compositions. A different way of describing mechanisms cannot solve this problem. But a different mental model might help us to employ a coping strategy: if all else fails, lower your standards. The theory of classification offers two strategies that might be useful here, approximation and transformation. Typical classifiers represent approximate solutions derived from samples. Although security properties may be undecidable in the general case, we can easily obtain sample decisions of a mechanism for particular inputs. If we could find meaningful ways of generalizing from such samples, tolerating "small" or "few" errors, we might get results good enough for practical matters. The second strategy, transformation, reminds us that we are trying to solve problems rooted in the real world. There is more than one way of representing a problem, and some ways may be easier to handle than others. If some properties are not decidable, what can we learn from those that are?

3. MODELING FRAMEWORK

3.1 Causality Model

A causality model comprises a space of causes, a space of effects, and cause-effect relationships. This paper treats the spaces mainly as sets; they may, however, have meaningful structure and operations.

Definition 1. Let $\overline{C}$ and $\overline{E}$ be spaces of entities, $\mathcal{P}(\overline{C})$ and $\mathcal{P}(\overline{E})$ their power sets, and $\rightsquigarrow$: $\mathcal{P}(\overline{C})\times\mathcal{P}(\overline{E}) \rightarrow \{true, false\}$ a binary relation between $\mathcal{P}(C)$ and $\mathcal{P}(E)$. We call the triple $W = (\overline{C}, \overline{E}, \rightsquigarrow)$ a *causality model*. Entities $c \in \overline{C}$ are the *elementary causes*, entities $e \in \overline{E}$ are the *elementary effects*, and $\rightsquigarrow$ is the *causality relation* of W. We call a set $C \subset \overline{C}$ of elementary causes a *cause* and a set $E \subset \overline{E}$ of elementary effects an *effect*. We will say "C causes E in W" or "E is an effect of C in W" if $C \rightsquigarrow E$. We call the graph $G_{\rightsquigarrow} = \{(C, E) : C \rightsquigarrow E\}$ of the causality relation the *valid causalities* of W.

In plain English this definition says that sets of causes lead to sets of effects. A set of causes may be related to

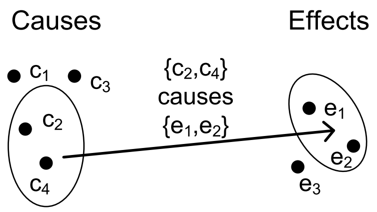

Figure 4: Causality model

multiple sets of effects and vice versa. The construction does not distinguish proximate from distal or ultimate causes and effects, and it offers no immediate way of expressing interdependencies between multiple causes or multiple effects. We can, however, express the fundamental distinction between necessary, sufficient and contributory causes. This requires an auxiliary definition:

Definition 2. Let $W = (\overline{C}, \overline{E}, \rightsquigarrow)$ be a causality model, $B \subset \overline{C}$, $C \subset \overline{C}$ causes in W with $B \supset C$, and $E \subset \overline{E}$, $F \subset \overline{E}$ effects with $F \subset E$. W is a *proper causality model* and $\rightsquigarrow$ a *proper causality relation* if $\forall C, \forall E : C \rightsquigarrow E \Rightarrow C \rightsquigarrow F \wedge B \rightsquigarrow E$.

In a proper causality relation, if C causes E it also causes any subset of E, and any superset of C also causes E. Adding elementary causes can only ever enlarge the set of effects, and ignoring an effect does not fundamentally change the causality.

Definition 3. Let $W = (\overline{C}, \overline{E}, \rightsquigarrow)$ be a proper world model, $B \subset \overline{C}$, $C \subset \overline{C}$ causes and $E \subset \overline{E}$ an effect. C is a *sufficient* cause of E if $C \rightsquigarrow E$. C is a *contributory* cause of E if $\exists B \supseteq C : B \rightsquigarrow E$. C is a *necessary* cause of E if $B \rightsquigarrow E \Rightarrow C \subseteq B$ and $\exists B \supseteq C : B \rightsquigarrow E$.

A sufficient cause implies its effect, the effect of a necessary cause implies the cause, and a contributory cause participates in at least one causality. These three definitions should suffice as an outline. We will use causality models as frames for security analyses. Conceptually they represent a world model of possible causes, effects, and causality relations. Causes include the intended use of a system as well as threats and attacks, and effects include desired effects as well as the possible results of malicious attacks.

Causality models facilitate simplifications where appropriate. In practical applications one might want to consider simpler models, for instance based only on elementary causes and elementary effects. This remains possible.

3.2 Security Policies and Vulnerabilities

A security policy specifies which causalities one whishes to permit. Defined with respect to a particular causality model, the security policy essentially specifies a subset of the causality relation of this model.

Definition 4. Let $W = (\overline{C}, \overline{E}, \rightsquigarrow)$ be a proper causality model, let $G_{\rightsquigarrow}$ denote the set of valid permissions in W, and let $\pi : \mathcal{P}(C) \times \mathcal{P}(E) \rightarrow \{permitted, forbidden\}$ be a binary relation with the graph $G_\pi = \pi^{-1}(permitted)$. We call π a *security policy* in W if $G_\pi \subset G_{\rightsquigarrow}$ and G_π the *permissions* of the policy π. We say "π permits (C, E)" if $(C, E) \in G_\pi$

Alternatively we could define a policy as an arbitrary binary relation on $\mathcal{P}(C) \times \mathcal{P}(E)$ and look at the intersection

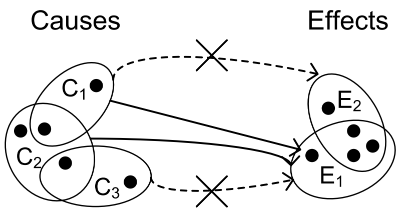

Figure 5: Security policy. Dashed arrows indicate forbidden causalities

between the graphs of the causality model and the policy. This technicality should not matter as we proceed. Note that $(\overline{C}, \overline{E}, \pi)$ is also a causality model, but not necessarily proper.

The business logic of a system implies a security policy. It can handle a certain set of acceptable causes, and it implies a set of effects that operations of the system can have. The functional specification determines the space of causes, which technical factors and software defects may modify and extend for the actual system. The environment of the system and the possible interpretations of outputs determine the space of effects. For acceptable causes the business logic, if implemented correctly, restricts the possible causalities and thus effects. For any other cause only the causality model restricts the effects. Effects achievable through acceptable causes and correct operation of the business logic are desired effects, all others are undesired.

To make a system secure with respect to a causality model, we have to equip it with security mechanisms such that the system abides by the security policy of its business logic for all causalities and effects under consideration. We thus define:

Definition 5. Let $W = (\overline{C}, \overline{E}, \rightsquigarrow)$ be a proper causality model, S a computer system and τ a security policy in W. We call τ the *target policy for* S with respect to W if S abides by this policy according to its functional specification, assuming correctness of the implementation and usage within the specified—or assumed—limits. We call $\{C \mid \exists E : \tau$ permits $(C, E)\}$ the set of *acceptable causes* and $\{E \mid \exists C : \tau$ permits $(C, E)\}$ the set of *desired effects* of S.

Note that a causality (C, E) is not necessarily permitted by τ if C is acceptable and E is desired. However, if C is not acceptable or E is not desired we can conclude that (C, E) is not permitted without further analysis.

The actual security policy which a system enforces may deviate from its target policy. The system may fail to handle permitted cause-effect pairs as it should and thus have a functional defect. And the system may support additional cause-effect relationships not permitted by its target policy, which makes the system vulnerable to attacks.

Definition 6. Let $W = (\overline{C}, \overline{E}, \rightsquigarrow)$ be a proper causality model, S a system, τ the target policy of S with respect to W, and π the actual policy that S enforces. Let G_τ and G_π denote the graphs of τ and π. We call $V = (C, E) \in G_\pi)$ a *vulnerability of* S if $V \in G_\pi \backslash G_\tau$. We call a system *vulnerable* with respect to W if it has at least one vulnerability.

The entire construction up to this point provides a rough framework allowing us to discuss security policies and vulnerabilities in terms of cause-effect relationships. The next subsection will attach security mechanisms to the conceptions of this framework.

3.3 Security Mechanisms

Security mechanisms impose their own abstractions and models upon a system, restricting the security policies that one can enforce using any particular mechanism. Mechanisms with their mechanism-specific policies may or may not be appropriate to enforce a given target policy with respect to a causality model. Ignoring the enforcement aspect of security mechanisms, we can model them as decision functions on feature spaces. The features represent aspects of causes and effects that a mechanism cares about. We will later map our causality model into these feature spaces.

Definition 7. Let $F = F_1 \times F_2 \times \ldots \times F_n$ be an n-dimensional space and $\overline{R}$ be a set of n-ary relations $F \rightarrow \{true, false\}$. We call the pair $M = (F, R \subset \overline{R}$ a *mechanism model*, F the *feature space* of the mechanism, and relations $\kappa \in R$ are *mechanism policies*. We say a mechanism *accepts* an n-tuple $f \in F$ under the policy κ if $\kappa(f) = false$. Otherwise we say that κ *denies* f. We write $\kappa_{accept}(F)$ for the set of accepted entities, and $\kappa_{deny}(F)$ for its complement, the set of denied entities.

We make no further assumptions regarding the structure of the feature space at this point. Its dimensions F_i may be discrete or continuous; the space or any of its dimensions may be finite or infinite.

A security mechanism does not uniquely define its model. Several distinct valid models may exist for a mechanism. We shall not explore validity criteria and feature space transformations here.

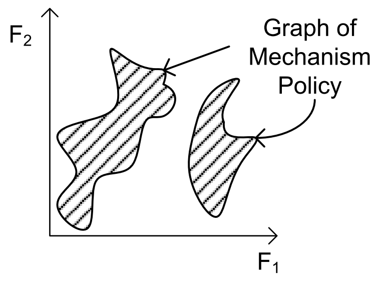

Figure 6: Security mechanism with graph of a mechanism policy

We can combine security mechanisms to build a more complex one. In terms of mechanism models this means to combine their feature spaces and their policies. Combining policies may have different meanings depending on how mechanisms interact in a system. For instance the combined policy may deny an entity if either of the two mechanisms denies it, or the combined mechanism may accept it if either mechanism does so.

Definition 8. Let $M_1 = (F_1, R_1)$ and $M_2 = (F_2, R_2)$ be mechanism models with $F_1 = F_{1,1} \times F_{1,2} \times \ldots \times F_{1,n}$ and $F_2 = F_{2,1} \times F_{2,2} \times \ldots \times F_{2,m}$, and let $F = F_{1,1} \times \ldots \times F_{1,n} \times F_{2,1} \times \ldots \times F_{2,m}$ be their combined (m+n)-dimensional feature space. For $F \ni f = (f_1, \ldots, f_n, f_{n+1}, \ldots, f_{n+m})$ let $f_1 = (f_1, \ldots, f_n)$ be the projection of f into F_1 and $f_2 = (f_{n+1}, \ldots, f_{n+m})$ the projection of f into F_2. Let $\odot$ be a logical operator. We call $R_{1\odot 2} = \{\kappa_{1\odot 2} \mid \kappa_1 \in R_1, \kappa_2 \in R_2, \kappa_{1\odot 2} : F \rightarrow \{true, false\}, \kappa_{1\odot 2}(f \in F) = \kappa_1(f_1) \odot \kappa_2(f_2)\}$ the set of $\odot$-*composed mechanism policies* of M_1 and M_2.

This definition illustrates only the general idea of composing mechanism models. We shall not explore here how one could simplify combined features spaces if they had dimensions in common, or how to define policy compositions beyond elementary logic.

Mechanism models can represent more than just security mechanisms. On can also describe vulnerabilities as mechanism models, using composition with the operator $\vee$ to model how vulnerabilities override decisions of actual security mechanisms.

3.4 Policy Mapping

Having two separate models, one of causalities and security policies and another of security mechanisms, we can connect the two by mapping one to the other. The analogous concept in pattern recognition classification is feature extraction. We assume a causality model constitutes a data space and a security mechanism provides a feature space, and represent feature extraction by a partial function.

Definition 9. Let $W = (\overline{C}, \overline{E}, \leadsto)$ be a proper causality model, $G_{\leadsto}$ the graph of its causality relation, and $M = (F, R)$ a mechanism model. We call a partial function $\phi : G_{\leadsto} \rightarrow F, f = \phi(C, E)$ a *policy mapping* of W into M.

Through a policy mapping, a mechanism policy defines a security policy in the causality model. Since we defined the mapping as a partial function, its domain may be a subset of the space in which we define security policies. Note that this definition implies that security mechanisms cannot create new causalities beyond the model considered. Assuming that a mechanism will not change causalities outside its domain, all causalities defined in our model remain unchanged outside this domain.

Definition 10. Let $W = (\overline{C}, \overline{E}, \leadsto)$ be a proper causality model, $M = (F, R)$ a mechanism model, ϕ a policy mapping of W into M, and π a security policy in W. We call the image $\{f \in F \mid f = \phi(C, E), (C, E) \in \pi_{permitted}\}$ of the permissions of π the *shape* of π in M under ϕ.

The shape of a security policy defines a relation in the feature space of M. This relation may or may not be a valid mechanism policy. We can also look at the effects of a policy mapping in the other direction.

Definition 11. Let $W = (\overline{C}, \overline{E}, \leadsto)$ be a proper causality model, $M = (F, R)$ a mechanism model, ϕ a policy mapping of W into M, and κ a mechanism policy in M. We define the *effective policy* $\phi^{-1}(M, \kappa)$ in W as: as $G_{\leadsto} \setminus \phi^{-1}(\kappa_{deny})$.

Figure 7 illustrates the notions. This construction enables us to analyze how a mechanism behaves and to translate the results back into our global model of causalities and security policies.

4. MODELING EXAMPLE

The following example illustrates the notions defined in section 3. It does not, however, illustrate how to apply the framework to analyze a system. The example rather shows how to describe a system using the framework in hindsight.

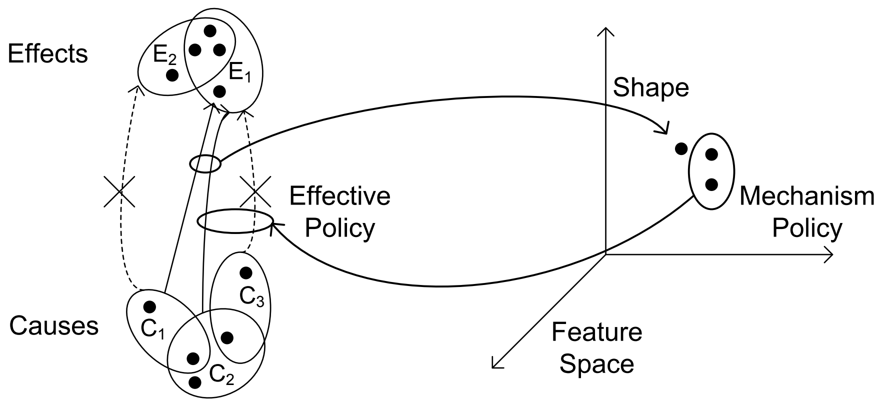

Figure 7: Policy mapping. Dots in the feature space represent the shape of a mapped policy

4.1 Scenario

Imagine a system handling applications for leave in an organization. Employees use this system to request permission from their bosses before going on vacation. For the sake of simplicity let us assume the system supports only the two roles *employee* and *boss*, the role *boss* including the role *employee*. We shall assume that the role *boss* is universal, that a boss is a boss for every employee. Let the system be implemented as a Web application with two operations: *create request* for all users and an additional operation *grant approval* for bosses. Let us assume that the application authenticates its users by username and password. We shall limit our considerations to attacks through the Web interface of our hypothetical application.

4.2 Causality Model

Elementary causes follow from possible inputs and interactions, events, actors, and side conditions that we want to consider. In our example we have users—or attackers—making inputs into a Web application. Each legitimate user has a role. Mind that we are modeling real-world roles here, not their enforcement inside the system. Attackers may thus not have a role assigned. Input comprises a user name, a password, an action—think of URLs—, and further parameters as a parameter string. Finally, the architecture of a Web application suggests that we should consider two input modes, through the browser or through plain HTTP. This gives us the raw material in table 1 to work with. How we use this material to define elementary causes remains up to our taste. For this example we shall build a seven-dimensional space and describe each elementary cause as a 7-tuple *(id, role, user, pass, action, param, mode)*.

Aspect	Range	Meaning
id	0. . . n	actors (real persons)
role	none, employee, boss	actual role of an actor
user	string	user name
pass	string	password
action	string	requested action
param	string	input parameters
mode	HTTP, browser	input mode

Table 1: Attributes of elementary causes

The purpose of our system dictates some of the possible effects. If operating normally, the system should produce as output permits for leave. To describe a permit we need a person for which it is valid, a person having granted the permit, and further parameters, such as dates. The set of elementary effects thus needs to contain, within reasonable ranges, at least all triples *(ID-req, ID-grant, param)* (table 2. Furthermore the system may produce error conditions, and it may produce other effects. To keep the model simple, let us just use *error* and *other* as two additional elementary effects. Error conditions and other effects do not carry the attributes of regular effects. This model of effects remains limited to the real-world interpretations of outputs. For more detailed analyses one might extend this model to include actual outputs. One could then also model possible misinterpretations.

Aspect	Range	Meaning
ID-req	0. . . n	applicant
ID-grant	0. . . n	person approving the request
param	string	encoded parameters

Table 2: Attributes of regular elementary effects

We have no reason to limit causalities in any way. A priori we do not know how the system behaves, so any combination of elementary causes may lead to any combination of results. Note that the causality model thus constructed cannot capture the more subtle aspects of security issues, such as the order of causes.

4.3 Security Policies

We can now use the causality model to specify security policies. To outline a policy, we expect the system:

- To produce a valid permit or an *error* condition if the cause comprises exactly two elementary causes: one person with the role *employee* making a valid request and one person with the role *boss* granting this same request. This imposes conditions on the cause as well as on the relationship between cause and effect.
- To correctly represent the identities of the persons acting in the output.
- To have no *other* effect for any combination of causes.

- To produce an *error* condition for any invalid cause or combination of causes, and to produce no valid permit in conjunction with such an error condition.

This is an outline of the strictest reasonable target policy. We may instead employ a looser policy, demanding only that the causes are correctly represented in their effects. This looser policy would allow anyone to grant permits to anyone as long as the permit correctly shows who made and who granted it. Such a policy might be sufficient in the scenario given; the organization will not accept a self-signed permit as valid even if the system allows an employee to create one.

We still do not know the effective policy which the system enforces, but we can derive some statements about it from knowledge about the business logic. Let us assume that the business logic takes just usernames, actions, and parameters as inputs. After code review and testing, we might be confident that the business logic ensures, for valid inputs through the user interface, that only the correct sequence of the operations *request* and *grant* produces a permit as output and that the output correctly represents all input parameters. We might also be confident that the business logic reliably produces error conditions for invalid inputs through the user interface within the length limits of input fields. Such observations allow us to check off some of the possible causalities as handled by the business logic, for instance those with causes representing inputs rejected by the business logic. For all others we might need security mechanisms—or more knowledge regarding the business logic. For instance the business logic does not guarantee that the roles will be valid for every output; we do not know how the system will behave for invalid HTTP requests; and our space of causes includes every possible combination of any person's identity and any username and password.

4.4 Security Mechanisms

To illustrate the modeling of security mechanisms, let us consider the two obvious mechanisms in our system: user authentication and the enforcement of roles. We can model user authentication in a two-dimensional discrete feature space, the Cartesian product of two sets of strings. One dimension represents all possible usernames, the other dimension represents all possible passwords. Any set of points—pairs of one username and one password—in this space might be a mechanism policy. Note that an actual implementation may impose restrictions on the possible policies. For instance the mechanism may enforce a minimum and a maximum length on passwords. Our definition of mechanism models allows us to represent such limitations as well but we will not consider them for this example. Figure 8 gives a idea what the feature space is like. Keep in mind, however, that we are discussing a discrete space.

We could also represent implementation bugs insofar as they change the policy that the authentication mechanism enforces. If, for instance, the mechanism accepted passwords of arbitrary length but verified only the first eight characters of every password, the mechanism policy would contain large equivalence classes of *(username, password)* pairs for every user account.

The second mechanism, access control on actions, decides which actions a user can carry out in the system. Our model of this mechanism thus uses a two-dimensional feature space. One dimension represents all possible usernames, the other dimension the two actions *request* and *grant*. A policy in this space is a set of *(username, action)* pairs permitted; an assignment of roles to users defines such a policy. Figure 9 illustrates the feature space for access control.

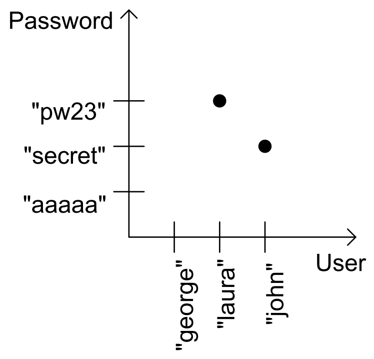

Figure 8: Feature space for password authentication. Dots represent the graph of the mechanism policy

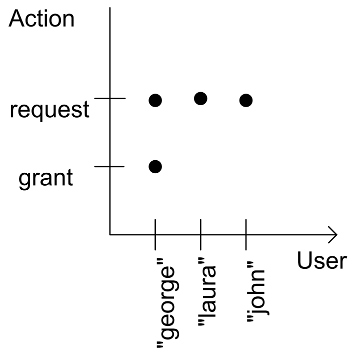

Figure 9: Feature space for access control. Dots represent the graph of a mechanism policy

We might also join the two feature spaces. The result would be four-dimensional according to the definition above, with the *username* dimension redundantly replicated. Defining a more reasonable 3-dimensional construction should be straightforward. Policies in this combined space would define which combinations of usernames and passwords could be used for which action.

4.5 Policy Mapping and Effective Policies

We constructed our causality model such that all three features used by the mechanism are present as attributes of our elementary causes. This makes the mapping appear simple—and raises interesting questions. Obviously, the *username*, *password* and *action* attributes should map to the respective dimensions and values of the mechanism feature spaces. But do other attributes have an impact? They do.

Consider two elementary causes differing only in their *mode* attribute. One represents an actor with a password requesting an action through the user interface, the other represents an actor doing the same through the plain HTTP interface. According to the target policy, the system should treat both causes in the same way. If, however, our security mechanisms were implemented on the client side, only the cause going through the user interface would be covered by their effective policy. The other cause, doing the same directly through HTTP, would not be in the domain of the policy mapping.

Whether this matters depends on the target policy. If, for instance, the system enforces authentication at the server side and access control on the client side—a typical issue in Web applications, where removing controls from the user interface does not imply that a function cannot be called any more—it may still be capable of enforcing the looser of the two target policies outlined above.

Or consider invalid actions specified by users. The causality model permits arbitrary strings as actions while the access control mechanism works only on the two logical actions that make sense for the business logic. Again we find a large subset of the possible causes not covered by the security mechanisms. This means our system may not be secure. We need either more information regarding system behavior for the causes not yet covered or further security mechanisms to ensure other causes will not have an effect.

Finally, when defining a policy mapping we notice that our security mechanisms do not work with the real identities of actors. Different actors can achieve the same effects if only they use the same usernames and passwords.

5. TOWARDS ANALYSIS TECHNIQUES

The conceptual framework defined in section 3 facilitates several ways of analyzing security mechanisms, compositions of mechanisms, and their suitability for a given system. This section outlines possible techniques and approaches.

5.1 Mechanism Analyses

Analyzing a mechanism as a classifier may tell us more about its capabilities.

Feature space structure.

Dimensions of a mechanism feature space represent aspects, factors, and parameters that influence decisions of a mechanism. Can we find or define meaningful subspaces, equivalence classes, or metrics in a particular space? In the space for password authentication for instance we might define a subset of *(username, password)* pairs with weak passwords. This allows us to define a metric on mechanism policies, representing the number of weak passwords used, or to exclude those policies that would contain weak passwords.

Mechanism comparison.

We can compare the feature spaces of two different security mechanisms. Two mechanisms are similar if their feature spaces comprise mostly the same dimensions and both mechanisms can enforce the same policies on their common subspace. Combining two such mechanisms may do little to improve the security of a system unless different policy mappings justify their combination. Spaces may even be isomorphic, indicating that two mechanisms are essentially the same, even if they seem different at the surface. For instance we know a variety of authentication schemes that require users to remember some secret and to authenticate themselves by entering an identifier and the secret.

Mechanism policies.

Mechanism policies may depend on hidden parameters not visible in the feature space. In our example above, the assignment of roles to users determines the policy for access control. What is the impact of such parameters upon the policy as a point set in the feature space? Are there for instance different combinations of hidden parameters that redundantly lead to the same policy? How does a policy change if one modifies parameters?

The inner workings of a mechanism may limit its capabilities. Are there meaningful policies one could define in the feature space of a mechanism but the mechanism is incapable of enforcing them? This would imply that the mechanism lacks complexity.

We may also analyze the contribution of individual feature space dimensions to decisions made by the mechanism. If we find dimensions that do not contribute, we might be able to simplify a mechanism, or a mechanism policy, without losing security. Likewise we might analyze the contribution of individual mechanisms in a composition. Perhaps a subset of the mechanisms would be sufficient to express those policies that we are interested in.

Mechanism vulnerabilities.

For many standard mechanisms, possible vulnerabilities and implementation defects are well-understood. We can interpret vulnerabilities as point sets in a mechanism's feature space and analyze how they modify a policy. This makes it possible to include for instance the results of security testing in a mechanism model.

5.2 Causality Model Analyses

Causality models define a problem space within which we discuss target security policies. Analyzing these models may give us more insight into properties of the security problem we are trying to solve.

Equivalence classes.

In a causality model, or in a security policy specified in such a model, we might be able to identify meaningful equivalence classes of causes, effects, or causalities. Obvious candidates are:

- All necessary causes for an undesired effect. Suppressing only one of them would be sufficient to suppress the effects.
- All effects for which a cause is necessary. This tells us what the impact is if a mechanism denies this cause.
- Classes of causes, such as attackers, users with particular roles, actors with certain capabilities, and so on. If they are clearly distinguishable, the security problem may be easier to solve.

If we find such equivalence classes, we can go on and analyze how mechanisms treat them.

Policy abstractions.

According to the definition in section 3.2, a security policy may be any subset of the possible causalities in a causality model. We may, however, find classes of causalities differing only in certain attributes that a policy consistently permits or rejects. For the example in section 4 we may for instance find that the target security policy effectively depends upon the roles of the actors and the actions carried out. Other properties of the causes are merely auxiliary properties that the system needs to distinguish actors from each other.

5.3 Mapping Analyses

Mapping analyses build upon the results of mechanism and causality model analyses by moving concepts from one part of the model to the other.

Shapes of policies in feature spaces.

Any given security policy mapped to the feature space of a mechanism defines a point set there. Is this point set a policy that the mechanism can enforce? If it is not, which part of the shape requires additional security measures? Which properties of causes and effects—if we model properties, as we did in the example above—determine the shape, and which properties are invariant to the mapping? For points or point sets in the feature space, which of the causalities are mapped to these points? Are multiple causalities mapped to the same point of the feature space? If so, does the security policy consistently permit or forbid all of them? What are the shapes of meaningful equivalence classes or policy abstractions and how are they related to enforceable mechanism policies?

Effective policies and vulnerabilities.

For a given policy mapping, a mechanism policy defines an effective security policy. We can analyze in the causality model the domain of the mapping—the coverage of the mechanism—and the effective policy itself. Ideally the effective policy of all mechanisms together should have the entire set of possible causalities as its domain, and it should enforce exactly the target policy. If the effective policy has permissions that are not in the target policy, we have found a vulnerability.

Searches.

The causality model represents the world with its assumed restrictions, while a mechanism model represents a security mechanism as it works and enforces some policy. Some attacks involve searching a system for vulnerabilities. We can model this as a search in the feature space of a mechanism. With a password authentication scheme for instance an attacker might try a number of different passwords for the same or for different usernames. We can discuss such searches in the mechanism model and look at their effects in the causality model. For instance an intermediate step of the search might imply an effect that ends the attack, either with success or with failure. We may also look at the causes of the causality model and analyze which ones an attacker might be capable of producing, and what the cost of an attack would be. We can then go on and investigate what this means to the mechanisms. Such considerations may as well be useful for security testers.

Problem-defined feature spaces.

Instead of analyzing a given solution one might use the causality model to design an appropriate security system. This requires that one models causes and effects in sufficient detail, including those properties that security mechanisms might use as features. The next step is to identify suitable equivalence classes of forbidden causalities and design mechanisms to handle them. Finally one takes care of the special cases that remain. This is essentially an extension of threat modeling [15] as part of the software development cycle.

Error metrics.

Considering the analogy of feature extraction with policy mappings, we might define error metrics for mechanism policies with respect to a target policy. Such metrics could express how many causalities are misclassified by the mechanism policy.

5.4 Probabilistic Extension

The focus of the conceptual framework outlined and discussed in this paper is on the classification properties of mechanisms, not on the strength of the enforcement of their decisions. However, some aspects of strength can possibly be modeled in a probabilistic extension of the framework. There are two candidates for probabilistic considerations, the classifier of the mechanism and the problem mapping.

A probabilistic classifier may represent actual probabilistic properties of the security mechanism considered. While many traditional security mechanisms are entirely deterministic, some contemporary security functions use heuristics. Antivirus software is an example, which besides scanning for static signatures often has the capability of evaluating the behavior of a process for patterns of malicious activity. Probabilistic or fuzzy classifier concepts may be an appropriate model, which captures and represents the risk of misclassification by the mechanism itself. Biometric authentication is another example of a class of mechanisms that are inherently probabilistic.

A probabilistic problem mapping may be suitable to model failures and attacks that are not easily expressed at the classifier level. Password guessing—with different success probabilities depending e.g. on the number of attempts possible and access to password hashes—is one example. Phishing attacks is another, where some success rate may be determined by factors beyond our control or precise understanding. A probabilistic extension of the problem mapping concept could model such misrepresentations.

6. CONCLUDING REMARKS

This paper has outlined a conceptual framework for analyzing the classification capabilities of a security mechanism. The basic model stems from classifiers as they are used in pattern recognition applications. Based on this classifier model of security mechanisms, several possible ways have been proposed in which a security mechanism can be analyzed as a solution of a security problem. Besides individual security mechanisms, compound mechanisms or security systems can be analyzed. The framework outlined here supplements existing approaches and tools for security analysis and evaluation.

The classification framework and its analysis methods are based on considerations that we commonly make when evaluating the security of systems and applications beyond their specific security mechanisms. A variety of actions, inputs, or other factors in the real world has to be considered in such an evaluation; only some valid subset must be allowed to affect the processing by the business logic of an application. A security mechanism or a suite of mechanisms is responsible for proper classification.

6.1 Expected Advantages

Generalization The framework as a modeling approach is generic in that it does not make specific assumptions on the inner workings of a security mechanism. It does not even strictly require that a mechanism is present. Vulnerabilities through software defects, which are often exploited in attacks, can be modeled as classification failures regardless of the mechanisms present in a system. The framework can thus capture both aspects of a security evaluation, mechanisms and implementation quality.

Analyze effects, not inner workings Traditional approaches to security analysis focus on the mechanics of security. Formal methods in particular tend to deal with the behavior, properties and strength of individual security mechanisms and policies. Abstract security objectives like confidentiality, integrity, and availability often frame the analysis in a limited model. The new paradigm proposed shifts attention from *how* mechanisms and policies work towards *what* they achieve. This complements the approaches and techniques that we already know.

Facilitate out-of-the-box thinking An independent model of the real-world security problem frames analyses under the classifier paradigm. This could make analyses less prone to errors and omissions caused by the assumptions and biases of individual mechanisms. The paradigm does not prevent us from using inappropriate models but at least it suggests to look beyond limited security mechanism and policy models.

Reasoning with incomplete information Aspects of *learning* classifiers have not been discussed here since this is not the point of this paper. But the essence of machine learning is to generalize from examples, from incomplete information. We often encounter incomplete security information, for instance if individual vulnerabilities are found in a system. There may be further instances of similar vulnerabilities, which just have not been found yet. The classification framework may be suitable for reasoning and generalization on the basis of such incomplete information.

A common abstraction If nothing else, a common abstraction for a variety of security mechanisms and policies is useful in itself. Even if the theory of classifiers does not lead to useful techniques after further investigation—the next subsection will discuss caveats—a unified way of representing important aspects of different security mechanisms is a valuable tool for security analysis.

6.2 Caveats and Limitations

Complexity and Computability One of the issues in security analysis is the size and complexity of the problem. A huge number of system states and actions would have to be considered for the analysis to be complete. The classification framework does not change this situation. Another potential issue is computability. In the end, security analysis means making statements about the behavior of programs. One might hit fundamental limits when trying to do this.

Threat modeling and prioritization While the classification framework comprises a way of modeling what might happen in the real world, it does not tell us which part of that model is actually relevant. If we can, however, prioritize certain parts of a security problem, the classifier approach allows us to focus on these parts. We can specifically look at their projections into feature spaces and analyze how well mechanisms do for these parts.

Static models The classification framework supports static models but ignores the dynamics of attacks. In reality, an attack often consists of multiple steps, each modifying the state of the target or some other entity. It might be possible to extend the framework to capture such aspects, e.g. by introducing parameterized classifiers as descriptions of security mechanisms.

Local models The general model underlying the classification framework is the model of a single system. Applications of the framework might be scaled down to subsystems and components, but the framework seems unsuitable for networks of interacting systems with complex relationships and interactions.

Ambiguity Once the target policy and the mechanisms are described as classifications in their respective spaces, the framework can be applied. But there might be multiple valid models of a security mechanism within the framework, and the model of the security problem is left to arbitrary choice.

Feedback loop into the world model Artificial features imply that the world model depends on the security mechanisms discussed. The classifier paradigm as discussed here does not address this interdependency. It may complicate analyses.

Analogy breakdown The classifier paradigm roots in a conceptual metaphor. While striking on the surface, the analogy may break down as soon as we look at the details. Pattern recognition builds on similarity among members of a class and dissimilarity between members of different classes, on notions of distance, and on statistical concepts like signal versus noise. Whether and where such notions have a meaning in security remains to be investigated, as does the question whether these concepts are necessary for the paradigm as such.

6.3 Open Questions

This paper outlines an idea without investigating the details. A number of open questions remain for further research:

Exploring the modeling framework The modeling framework in section 3 thus far consists only of preliminary definitions. As a prerequisite for further work it needs to be extended and refined. Transformations and operations on mechanism models are of particular interest, as are formal definitions of the analysis techniques outlined in section 5.

Modeling techniques The example in section 4 is based on many ad-hoc decisions. For effective application of the paradigm we need techniques that help us to build suitable causality models for a given system. Choosing an unsuitable model spoils the analysis. Modeling techniques thus need to guide their users towards appropriate considerations.

Does proximity matter in feature spaces? As noted under *caveats* above, the classifier analogy may break down if proximity has no meaning in feature spaces. On the one hand, modeling typical policies as classifiers might lead to sets of isolated points where we cannot draw meaningful conclusions from a point about its neighborhood. On the other hand, similarity is obviously meaningful with some common vulnerabilities. A buffer overflow defect for instance supports a number of individual attacks that essentially exploit the same behavior of the software in the same way, differing only in their payload and effects.

Two examples came up during the workshop. A broken password hashing function may lead to a system accepting multiple different passwords for the same user. While these passwords may not be close to each other as points in the feature space, they would have the username in common. The other example is a bug making the system ignore the second character of each password during verification. This bug should also translate to some sort of pattern in the feature space.

Searching the space as an attacker Attackers often start with a broad search on a system to find potential vulnerabilities and then follow traces to find an exploit. Representing all the security mechanisms of a system in a high-dimensional space might help us to model this behavior. We might project search paths back and forth between the problem space—the causality model—and the mechanisms space—the implemented security model. Restrictions differ in these two spaces. An attack is successful if the attacker obeys the rules of the world yet finds a hole in the mechanisms space.

Besides representing such behavior, we might take advantage of it as defenders. If we could systematically increase the size of the space the attacker has to search, attacking would become harder. This strategy is well-known e.g. from cryptography.

How can we determine the mapping of problems to mechanisms? Are there automated ways of determining the mappings into feature spaces? To complicate things, the mapping depends on the particular world model chosen. Also, some aspects could be modeled equally well in the feature space or in the mapping.

Distinguishing attacks from mistakes An interesting idea from the workshop discussion is to apply the paradigm to event classification. Given a space describing a mechanism, accidental input errors, for instance a user mistyping a password, and attacks may exhibit different characteristics. The mistyping user may hit a point close to the correct password while a brute force guessing attack either hits random points or even shows a clear pattern over time.

As an afterthought, different classes of input errors may also exhibit different patterns. Mistyping a password can mean a number of different mistakes: hitting a wrong key on the keyboard; typing the old password instead of the new one after having changed the password; or mistakenly typing the password for another system. It might therefore be interesting to explore the classifier paradigm from a usable security point of view.

Representing time The model so far is admittedly static, which seems not very appropriate. Security mechanisms may base their decisions not only on input but also on internal state, and attacks may comprise a series of actions. Adding time as a dimension to all spaces involved may be a straightforward way of making the model dynamic.

Availability of labeled data The paradigm proposes a world model, which is essentially a labeled dataset. Obtaining such labeled data is expensive. If we could obtain labeled data for typical security problems, it would be worthwhile to put them down for reuse.

7. ACKNOWLEDGEMENTS

I owe gratitude to a number of people: to my NSPW shepherds, Hilary Hosmer and Steven Greenwald, as well as to the anonymous reviewers, for many insightful and encouraging remarks helping me to improve my paper; to all NSPW participants for a lively discussion and a great workshop; and to my colleagues Katja Seitz and Timo Winkelvos for reviewing early drafts. This work was supported by CASED (www.cased.de).

8. REFERENCES

[1] Ross Anderson. Why cryptosystems fail. In *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*, pages 215–227, New York, NY, USA, 1993. ACM.

[2] Ross J. Anderson. *Security Engineering: A guide to building dependable distributed systems.* Wiley, 2008.

[3] Mitra Basu and Tin Kam Ho, editors. *Data complexity in pattern recognition.* Springer-Verlag New York Inc, 2006.

[4] David Elliot Bell. Looking back at the Bell-La Padula model. In *Proceedings of the 21st Annual Computer Security Applications Conference*, pages 337–351. IEEE Computer Society Washington, DC, USA, 2005.

[5] Common criteria for information technology security evaluation v3.1. available online, `http://www.commoncriteriaportal.org/`, 2006.

[6] Menahem Friedman and Abraham Kandel. *Introduction to pattern recognition: statistical, structural, neural and fuzzy logic approaches.* World scientific, 1999.

[7] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, 1976.

[8] Mikko Hypponen. 21 Solutions to Save the World: Masters of Their Domain. *Foreign Policy, May/June*, 2007.

[9] A.K. Jain, R.P.W. Duin, and Jianchang Mao. Statistical pattern recognition: a review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1):4–37, Jan 2000.

[10] J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas. On combining classifiers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(3):226–239, Mar 1998.

[11] J. Kittler, M. Hatef, and Duin R. P. W. Combining classifiers: A theoretical framework. *Pattern Analysis & Applications*, 1(1):18–27, 1998.

[12] Nikunj C. Oza and Kagan Tumer. Classifier ensembles: Select real-world applications. *Information Fusion*, 9(1):4 – 20, 2008. Special Issue on Applications of Ensemble Methods.

[13] U. Priss. Formal concept analysis in information science. *Annual review of information science and technology*, 40(1), 2006.

[14] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[15] Frank Swiderski and Window Snyder. *Threat Modeling*. Microsoft Press, 2004.

[16] D. Whitley and J.P. Watson. Complexity theory and the no free lunch theorem. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, page 317, 2005.

[17] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. Technical report, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[18] Xerox workcentre/workcentre pro 232/238/245/255/265/275 multifunction systems security target. available online, `http://www.commoncriteriaportal.org/files/epfiles/ST_VID10135-ST.pdf`, 2005.

[19] Xerox workcentre/workcentre pro 232/238/245/255/265/275 multifunction systems validation report. available online, `http://www.commoncriteriaportal.org/files/epfiles/ST_VID10135-VR.pdf`, 2006.