ENCODING AND ISOLATING COMPUTER SECURITY CONCERNS

Marc Welz and Andrew Hutchison
Data Network Architectures Laboratory
Department of Computer Science
University of Cape Town
Rondebosch
7701
{mwelz,hutch}@cs.uct.ac.za

Abstract

We examine several different approaches which may be used to describe the security related activity of a computer system. Such descriptions are a first step toward managing these issues, as they facilitate the task of extracting security responsibilities and transferring them (to the extent possible) to external, specialised security subsystems. The framework structuring this analysis takes the form of a taxonomy of security mechanism, with particular emphasis on a particular attribute: The *level of abstraction* in terms of which the security events are generated and the policies specified. We propose three major categories for this attribute: *Infrastructure*, *application* and *security* abstractions. Infrastructure abstractions define events in terms of the components used to implement the monitored system (such as the system calls issued by a guarded web server), application abstractions define events in terms of the domain entities introduced by the monitored system (eg HTTP requests fielded by the web server), while security abstractions map system activity to domain independent concepts (for example, HTTP requests can be mapped to subject-object-access triples). With the aid of this taxonomy we examine some of the tradeoffs which are made by different approaches used to encode security activity.

Keywords: Security Taxonomy, Security Abstraction

ENCODING AND ISOLATING COMPUTER SECURITY CONCERNS

1 Background

As part of an investigation into how security concerns may be isolated from the rest of a computer system, we have generated a taxonomy of security measures. The taxonomy classifies systems using five attributes:

▷ **1 Time** : *When* does the measure operate ?

▷ **2 Location** : *Where* is the security measure placed ?

▷ **3 Human Involvement** : *Who* needs to support the security measure ?

▷ **4 Abstraction Level** : *What* events does the security measure analyse ?

▷ **5 Analysis Complexity** : *How* much time and space is needed to perform the analysis ?

This paper examines the fourth attribute, *level of abstraction*, in greater detail. This attribute seeks to describe the representations which can be used to encode security related activity or events.

2 Abstraction Categories

This section describes three categories which may be used to classify different security event representations. The intention is not to establish a detailed framework capable of defining every possible abstraction uniquely; instead the objective is to identify a general quality of the entities used to define security events.

The basis for the categories is the observation that a computer system can be divided into two parts — an infrastructure (which is used to construct or service the component under consideration) and the component of interest (here referred to as the *application*, the client of the infrastructure), while security concerns (if feasible to externalised) take a third form, as shown in Figure 1. The three security abstraction categories derived from this view are listed below:

- *Infrastructure* abstractions describe security events in terms of elements which are used to construct or service the guarded application. For example, if the guarded application is hosted by a conventional operating

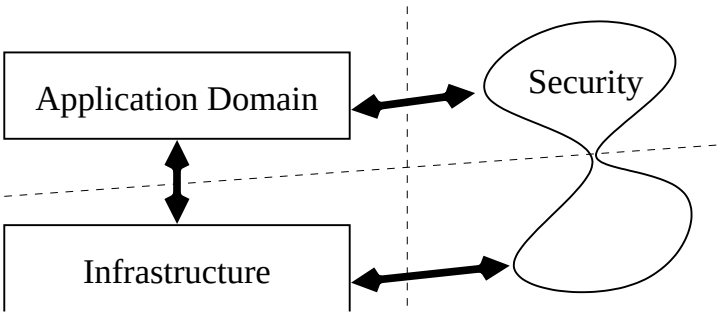

Figure 1: Abstraction Categories

system, then infrastructure security events could be defined in terms of the *files* opened or the *system calls* issued by the application.

- *Application* or *domain* abstractions are introduced by the developer of the system under consideration to describe the domain in which the system operates. For example, a word processor might operate on *paragraphs* and *chapters*.

- *Security* abstractions are independent of a particular application or infrastructure, but are introduced to capture a generic security issue. For example, a *security class* of a multi-level access control system labelled *confidential* is a construct which could include both a word processor paragraph or an operating system file.

3 Category Assignments

This section assigns several types of security measures or mechanisms to the above three categories. These will subsequently be used to illustrate the tradeoffs which are imposed by the respective category.

3.1 Infrastructure Abstractions

Network firewalls and less sophisticated network intrusion detection systems are security mechanisms which operate on infrastructure abstractions. Typically each packet is considered a security event, and policies are specified as a list of rules stating which hosts may send packets to which ports. Host addresses, ports and packets are all abstractions introduced by the infrastructure which are used by applications to implement higher level services.

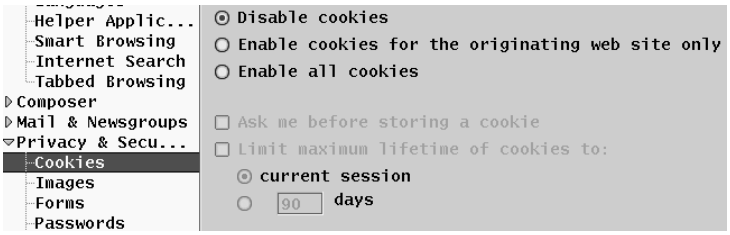

Figure 2: A Security Setting of the Mozilla Web Browser

Other, operating system related, examples are the likes of Medusa DS9 [14], the Wrapper Support System (WSS) [3] and Linux Security Module (LSM) Interface [13]) , which define policies in terms of kernel operations.

3.2 Application Abstractions

Application abstractions generate security activity (and define policy) in terms of the domain in which the application operates. An example is shown in Figure 2, which illustrates how a user may adjust the cookie acceptance policy of a web browser. Other application domain policies may be encoded as mail user agent filtering rules or chat client buddy lists.

3.3 Security Abstractions

Security abstractions differ from the above two categories that they require an effort on the part of the infrastructure or application developer — it is necessary to map native events to a model or framework at least partially independent of the particular system. A number of these mappings are described below:

- The conventional access control model [5] (subjects having a set of access rights to objects) can be considered a security abstraction which maps security related activity of a particular system to a set of *subject-object-access triples*. Usually this mapping is only partial, as the meaning of these values remains dependent on a particular system (an operating system may operate on files, a database on tables, *etc*). However, when a security labelling scheme (either in the form of security compartments or levels) is used, an access control abstraction can be separated from the application or infrastructure.

- Intrusion detection systems employing machine learning techniques to discover anomalies (an approach sufficiently general to be applied to

both applications and infrastructure activity) tend to require particularly *information dense representations* of security relevant information for optimal operation. Mapping system activity to such a representation usually involves feature selection (see [7] for an examination of this task in a security context) and encoding efforts (an example of a mapping from a TCP connection attempt to a compact 49 bit string is described in [4]).

- The classical `syslog` [9] interface is primarily used to report unstructured infrastructure or application-specific events. However it also includes an eight-valued *severity* field (ranging from routine debug and informational messages to high-importance alerts and emergencies) which makes it possible to prioritise events without knowledge of the domain in which the reporting system operates. Similar priority, importance or risk values are encountered in a number of other systems: For example, the `snort` IDS labels suspected intrusions using an one of eleven priorities.

- The Distributed Auditing Standard (XDAS) [12] defines 9 default event classes which in total contain 45 generic events to which the activity of a given system can be mapped. Event classes relate to tasks such as the administration and use of accounts, communications channels, services and applications. Within each of the event classes, activity is generally reported in terms of the *creation*, *access*, *modification* or *destruction* of an entity.

- Although the drafts of the IETF Intrusion Detection Working Group [2] as well as the Logging Data Map (LDM) of Ranum and Robertson [11] also operate on domain abstractions, primarily network related, it is possible to identify a security abstraction (not unrelated to the subject-object-access model) which defines activity in terms of a *source* (possible attacker) and a *target* (potential victim). In the case of the LDM, 10 of the 23 tags are used to describe source and target entities (example tags are `SRCPID`, `SRCPATH`, `SRCDEV` and `SRCUSER`), while others are used to label information such as event priority, time and error condition. Similarly alerts defined using the proposed IETF Intrusion Detection Message Exchange Format consist primarily of source and target elements, which may be nodes, users, processes or services.

4 Analysis

This section examines how a number of tradeoffs are distributed over the different categories, and how these influence the other four taxonomy attributes.

Arguably the oldest and best developed abstractions are those developed as part of classical operating system access controls. Calls for the isolation of security concerns can, for example, be found in the three decade old Anderson Report [1] which argues for the construction of a security kernel, where *"the objective of a security kernel design is to integrate in one part of an operating system all security related functions"*. While not formally verified or completely minimal, most modern operating system do provide an approximation of a security kernel. It enforces policy by consulting access or capability lists to establish if a particular subject may perform the requested operation on the given object.

Despite the existence of such infrastructure security kernels, failures in other parts of a computer system, particularly applications, continue to compromise computer security. In part this can be attributed to deficiencies in the execution of the implementation and administration of the security kernel concept — a security kernel may be flawed or too diffuse, subjects may be poorly authenticated, and applications may execute with greater than necessary privilege [8].

However, we contend that the limitation does not only lie in the implementation of the reference monitor design — we argue that certain application classes are inherently *trusted* and thus difficult to protect at the infrastructure level. In this context we define a trusted applications to be one which possess some privilege which is not accessible to the parties interacting with it. This assertion can be illustrated using an example: Consider a trusted medical database application (and its infrastructure in the form of a conventional operating system) which permits a certain researcher to retrieve a particular disease incidence rate for a population, but not the disease status of an individual. The disease incidence rate is a domain specific abstraction introduced by the database and has its own security requirements.

Controlling access to such domain abstractions at the infrastructure level is nontrivial, as the mapping from infrastructure resources to abstractions can be arbitrarily complex, requiring either the duplication or inclusion of substantial application functionality in the policy enforcement system. For example, in the the above database the calculation of the population average may generate the same file system reads as a listing of all individuals — distinguishing between the two would require that the policy statements include the database logic.

The above example is related to the confinement problem [6] which also supports the view that infrastructure controls are necessary, but not sufficient to secure a conventional computer system. If conventional access controls are taken to occupy the lower right quadrant of Figure 1 (an security abstraction implemented by the infrastructure), then the medical database system illustrates that the abstractions of the lower right quadrant may be insufficient to track the activity of the upper layer — the process of constructing a security abstraction discards some information (present in the left lower quadrant) which may be used by the application (upper left quadrant) to construct entities with security implications (upper right quadrant).

In the case of the medical database, the host operating system security kernel may be invoked on opening a database volume or the connection of a client, but may not be provided with the content of network or disk buffers necessary to distinguish between a query which retrieves the list of diseased users instead of the population average. Even if such information were made available, the analysis subsystems of contemporary operating system reference monitors (designed to perform little more than table lookups) would lack the facilities to decode these queries.

It can be argued that this limitation has lead to the introduction of other security subsystems which are intended to protect applications, but operate more directly on infrastructure abstractions. Representatives include intrusion detection systems and operating system security extensions described previously. For example, a network intrusion detection system may be used to protect a server application by monitoring the packets sent and received. In order to follow application protocol exchanges, the analysis subsystem processing these packets may have to be substantially more complex than simple table lookup of a conventional reference monitor. The tracking of application logic and state by the security analysis subsystem has to be of high fidelity, as inconsistencies between it and the monitored application may provide opportunities for desynchronisation attacks — these disable or blind the monitor. A substantial number of desynchronisation possibilities in an IP environment have been described by Ptacek *et al* [10] and while a number of these have been remediated in newer network IDS implementations, it has been possible to identify further possibilities — for example, we found that TCP urgent data can be used to mask some attack payloads.

These limitations support the view that infrastructure security measures are poorly suited to the protection of trusted applications — they are either vulnerable to desynchronisation or duplicate application functionality to such an extent that it may be possible to do away with large parts of the application.

It is this weakness which makes it necessary to consider security mech-

anisms which involve the developer of a trusted application. From the perspective of the application developer, these measures are no longer non-bypassible, but have the advantage of being less vulnerable to desynchronisation. Consider the example of a web browser — a desynchronisation attempt which fragments and otherwise reformats an HTTP transaction may succeed in evading an infrastructure level security mechanism which blocks undesired content, but less effective if the web browser itself is configured to reject this (as shown in Figure 2).

A complication of this approach is that the security responsibilities are distributed among applications and thus no longer concentrated at a security kernel. Given that the various trusted applications differ in design and employ disparate abstractions, policy formulation is more expensive. In this context security abstractions may be particularly interesting, as they could be used to map the diverse application abstractions (top left quadrant of Figure 1) to general security concerns (top right quadrant), making it possible to delegate analysis functions to a general purpose system, while still depending on the application developer to perform the mapping and so defend against desynchronisation attempts.

Although the same security abstractions employed by infrastructure systems can be used at the application level, it may be useful to develop alternatives to take account of the fact that the policies may be intended to control the internal behaviour of the application, rather than the hosted entities as would be the case in an infrastructure system. This can be illustrated by comparing an operating system security kernel with the likes of a web browser — the primary purpose of the security kernel is to control the applications it hosts, not to defend against flaws internal to the security kernel, particularly given the these tend to be relatively rare as the security kernel tends to be small and well validated. This contrasts with a web browser, which is a larger, less well tested system — in such a case it would be useful to report activity in terms of security abstractions which may be useful to detect failures internal to the system.

For this purpose we propose (and have implemented a system which, amongst other abstractions, exercises it) a security abstraction which assigns application modules functionality labels (analogous to subject clearances and object classifications of a military multi-level security system). This abstraction can be used to induce a requirement and dependency hierarchy on the application components, which in turn can be used by an external analysis subsystem to implement an orderly contraction of application functionality, whether to enforce explicit policy or in response to anomalous activity. We suggest that the construction of further abstractions designed to capture application concerns may provide an interesting line of inquiry.

5 Synopsis

This paper has reviewed several forms of representing security related information. In the context of the five taxonomy attributes given earlier, the analysis can be summarised as a set of statement describing the tradeoffs made by the various abstractions:

- Classical security abstractions used by security systems located ($\triangleright$ 2) in the infrastructure are desirable, but do not provide sufficient information to analyse trusted applications, furthermore the associated analysis systems lack the computational resources ($\triangleright$ 5) to perform this task even if this information were provided.
- Security systems operating directly on infrastructure abstractions may have access to sufficient information and may possess the needed computational resources ($\triangleright$ 5) to perform the analysis. However tracking of a trusted application at this level is expensive, as it may require the re-implementation of significant application functions in the policy enforcement subsystem — a duplication of application programmer effort by administrators ($\triangleright$ 3).
- A strategy of involving the application developer ($\triangleright$ 3) in the security effort locates ($\triangleright$ 2) part of the security subsystem in the application and, since it depends on the integrity of the guarded application, requires that the measure act before ($\triangleright$ 1) an attack compromises the security system.
- In order to reduce the administrative burden ($\triangleright$ 3) imposed by different application domain abstractions, it is interesting to develop security abstractions which can be used by application providers to describe the security characteristics in a domain independent form.

References

[1] J. P. Anderson. Computer security technology planning study. Technical report, USAF Electronic Systems Division, October 1972.

[2] M. Erlinger, S. Staniford-Chen, et al. IETF intrusion detection working group. http://www.ietf.org/html.charters/idwg-charter.html, 1999.

[3] T. Fraser, L. Badger, and M. Feldman. Hardening COTS software with generic software wrappers. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 2–16, May 1999.

[4] S. A. Hofmeyr. *An Immunological Model of Distributed Detection and Its Application to Computer Security*. PhD thesis, University of New Mexico, May 1999.

[5] B. W. Lampson. Protection. In *5th Princeton Conference on Information Sciences and Systems*, pages 437–443, March 1971.

[6] B. W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.

[7] W. Lee, S. J. Stolfo, and K. Mok. Mining audit data to build intrusion detection models. In *International Conference on Knowledge Discovery and Data Mining*, September 1998.

[8] S. B. Lipner. Security criteria, evaluation and the international environment. *Forum on Risks to the Public in Computers and Related Systems*, 12(46), October 1991.

[9] C. Lonvick. RFC 3164 the BSD syslog protocol, August 2001.

[10] T. H. Ptacek and T. N. Newsham. Insertion, evasion and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, 1998.

[11] M. J. Ranum and P. D. Robertson. Logging data attribute map. http://www.ranum.com/logging/logging-data-map.html, August 2002.

[12] The Open Group. Distributed audit service (XDAS) base. http://www.opengroup.org/pubs/, 1997.

[13] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman. Linux security module framework. In *Ottawa Linux Symposium*, 2002.

[14] M. Zelem, M. Pikula, and M. Ockajak. Medusa DS9 security system. http://medusa.fornax.sk/, 1999.