The following paper was originally published in the
Proceedings of the Second USENIX Workshop on Electronic Commerce
Oakland, California, November 1996

Organizing Electronic Services into Security Taxonomies

Sean Smith, IBM Research
Paul Pedersen, Los Alamos National Laborator

Organizing Electronic Services into Security Taxonomies

Sean W. Smith
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
sean@watson.ibm.com

Paul S. Pedersen
Los Alamos National Laboratory
Mail Stop B265
Los Alamos, NM 87545
pedersen@lanl.gov

Abstract

With increasing numbers of commercial and government services being considered for electronic delivery, effective vulnerability analysis will become increasingly critical. Organizing sets of proposed electronic services into security taxonomies will be a key part of this work. However, brute force enumeration of services and risks is inefficient, and ad hoc methods require re-invention with each new set of services. Furthermore, both such approaches fail to communicate effectively the tradeoffs between vulnerabilities and features in a set of electronic services, and fail to scale to large sets of services. From our experience advising players considering electronic delivery, we have developed a general, systematic, and scalable methodology that addresses these concerns. In this paper, we present this methodology, and apply it to the example of electronic services offered via kiosks (since kiosk systems are representative of a wide range of security issues in electronic commerce).

1. The Problem

As business—commercial services provided to customers as well as government services provided to citizens—migrates to electronic settings, the contributions of security research are many: from developing underlying technology, to applying this technology to construct particular methods for secure service delivery, to verifying (both formally and experimentally) the security of these methods.

However, between the decision to enter the electronic marketplace and the decision to deploy a specific service via a specific delivery method lies a period of exploration. We believe research needs to address this gap: how to illuminate the issues and tradeoffs between variations of services and delivery technologies. We offer this paper as an initial contribution: an attempt to extract a systematic methodology from our work in this area.

In general, vulnerability analysis consists of specifying the vulnerabilities and points of attack to which a given service is susceptible. However, in practice, we have found that managers considering electronic delivery usually provide a large set of *related* services, and implicitly expect the analysis to communicate the tradeoffs between vulnerabilities and features. Consequently, the ability to organize a set of services into a structure that clearly and concisely expresses their relations and their security risks is crucial for such an analysis to be effective. The understanding that an effective vulnerability analysis provides is, in turn, crucial for the decision on deploying electronic services to be sound.

The question, then, is how to construct and express this structure. Brute force enumeration of services and risks is inefficient and fails to meet the customer's implicit goals. Ad hoc methods require re-invention with each new set of services, and can fail to communicate the tradeoffs effectively. Furthermore, these methods do not scale to large sets of services—and communicating *anything* effectively for a large set requires understanding and exploiting organization inherent in that set.

On the other hand, a systematic methodology provides both efficiency for the analyst as well as effective communication for the customer. Furthermore, a sufficiently general methodology can also apply directly to organizing points of attack (since these are essentially just potential services unintentionally offered to the adversary). Additionally, a sufficiently expressive methodology can provide an understanding of the structure underlying a given set of services, which in turn can reveal when that set is incomplete.

From our experience in organizing electronic services into security taxonomies as part of vulnerability analyses, we have developed a general, systematic methodol-

This research was performed while the first author was with Los Alamos National Laboratory, and this paper is registered as a Los Alamos Unclassified Release. This research was sponsored in part by the Department of Energy, under contract number W-7405-ENG-36. The views and conclusions contained in this document are those of the authors alone.

ogy that uses the structure of a service set to organize and enumerate the corresponding vulnerability set. This methodology offers many advantages over ad hoc approaches, including making it easier to propagate changes in services to changes in vulnerabilities, to construct a complete list of vulnerabilities, to construct structured countermeasures, and to bring conciseness to service and vulnerability specifications (in some cases, reducing the length from exponential to polynomial).

Section 2 gives a theoretical presentation of the methodology. Section 3 applies the methodology to real-world examples from the domain of kiosk systems. We chose kiosk systems, since they are representative of a wide range of security issues in electronic commerce, from point-of-sale devices to networks to remote servers, from tamper-hardened public machines running dedicated browsers accessing a set of pages residing entirely on local CD-ROM to off-the-shelf private machines running Web browsers accessing the entire Web.

2. The Methodology

Section 2.1 establishes definitions for the terms we use in the paper. Section 2.2 considers the challenges of effectively enumerating sets of implementations and vulnerabilities. Section 2.3 presents mathematical structure organizing services into a partial order, and considers the implications of this order for vulnerability sets. Section 2.4 uses this structure as a foundation for a systematic methodology for constructing security taxonomies.

2.1. Definitions

In this paper, we build a mathematical structure that models real-world concepts. Since many such structures are possible, and since real-world terminology tends to be inexact anyway, we begin by defining real-world terms in the context of our model.

> **Definition 1** An electronic *service* is an offered action characterized by some specific goal (which is usually defined loosely and imprecisely) and carried out via some computational device.

Examples of electronic services include transferring funds between savings and checking accounts via Automatic Teller Machines, and filing a 1040 tax form via the World Wide Web.

Typically, designers characterize a service by explicitly listing some correctness properties, but leaving many others unstated. Usually these properties divide into a small primary class (characterizing the basic functionality of the service) and a larger secondary class (characterizing all the extra conditions necessary for the service to be correct, fault-tolerant, secure, etc.). Fully enumerating the correctness properties that a service is implicitly expected to satisfy is often a substantial challenge in security analysis. For example, designers of a certain Automatic Teller Machine system failed to make explicit the assumption that no customer would ever initiate a "withdraw cash" transaction but leave the cash in the machine [4]. As a consequence, clients engaging in this unanticipated behavior forced machines into bizarre failure modes.

> **Definition 2** An *implementation* is a collection of hardware and software that realizes a particular service. In particular, an implementation must satisfy the primary correctness properties of a service. (However, in order to be fully correct, secure, and fault-tolerant, an implementation must satisfy all the properties.) For a service s, define IMP(s) to be the set of implementations of service s.

We assume that "implementations" are specified with granularity sufficiently coarse that two services can share an implementation. For example, if services s_1 and s_2 truly differ, than their implementations must differ at some close level of inspection—perhaps the code is not identical. However, such close scrutiny would prevent discussion of facts such as "a kiosk with sufficient database access to implement change-of-address can also implement a guide to local stores"—hence our assumption.

> **Definition 3** Let s be a service. For an implementation I of this service, a *vulnerability* of I is a potential action that causes some of the correctness properties of the service to fail to hold. A vulnerability of s is the set of vulnerabilities common to all implementations of s. Define VULN(I) to be set of vulnerabilities of implementation I, and VULN(s) to be the set of vulnerabilities of service s.

2.2. Concisely Specifying Sets

One of the motivations for building taxonomies is to increase the effectiveness of communication to the customer. However, a simple enumeration of the elements of a set (especially a large set) is not a good example of effective communication. Consequently, we need ways to concisely describe sets such as IMP(s) and VULN(s) for a specific service s.

Implementations The set of implementations that realize a service is open-ended: if I realizes service s, then I along with an implementation of a completely unrelated service also realizes s. For most services that we consider, however, the implementations can be characterized as all those hardware/software combinations meeting some minimal set of requirements. Usually these requirements are implied by the customer's specification of the service: e.g., our customer does not just want to provide change-of-address, but change-of-address via kiosks. In these situations, we can describe $\text{IMP}(s)$ simply by listing the minimal hardware and software requirements; the fact that a kiosk with appropriate database privileges *and* the ability to play digitized excerpts from the works of Beethoven comprises an implementation of a change-of-address service is not only irrelevant, but also directly implied by saying that a kiosk with appropriate database privileges comprises an implementation.

Vulnerabilities The structure of the set of vulnerabilities is potentially much more complex. For example, suppose A and B are both members of some $\text{VULN}(s)$ or some $\text{VULN}(I)$.

- Should "A followed by B" also be a member? What if A is "blow up the kiosk?"
- Should "A and B concurrently" be a member?
- If A' is not a member, should "A' followed by B" be a member? What if A' enables B?

We provide one sample solution to this problem. First, we define compound actions.

> **Definition 4** For actions A and B, define $[A; B]$ to be the compound action consisting of doing A first, then doing B.

We now examine when such compound actions can be vulnerabilities. We consider the system consisting of implementation plus attacker to be in a particular state. With each action A we can associate two items:

- A set PRE_A consisting of all possible states of the system in which A can be applied. (As with implementation sets, this set might be expressed succinctly as a list of conditions which a state must satisfy.)
- A function FUN_A that indicates how A transforms systems states.

For actions A and B, the compound action $[A; B]$ would be possible if B is enabled after performing A, and would have the result of doing A, then doing B. This allows us to specify the precondition set and transformation rule for $[A; B]$:

- $\text{PRE}_{[A;B]} \equiv \text{FUN}_A^{-1}(\text{PRE}_B)$
- $\text{FUN}_{[A;B]} \equiv \text{FUN}_B \circ \text{FUN}_A$

This calculus lets us build a closure rule, specifying when compound actions can be vulnerabilities:

- Suppose for $B \in \text{VULN}(s)$, action A satisfies $\text{FUN}_A^{-1}(\text{PRE}_B) \neq \emptyset$. Then $[A; B] \in \text{VULN}(s)$.

Enumerating some $\text{VULN}(s)$ or $\text{VULN}(I)$ reduces to enumerating some small set of atomic attacks, and then specifying the closure rule or rules which expand the atomic set into the full set.

This sample solution ignores the challenge of attacks composed of concurrent actions. Indeed, the problem of specifying all possible action combinations that comprise attacks is essentially the same as specifying all possible behaviors of computational devices (potentially with parallelism and randomization) – which is a problem that has received much attention in its own right (e.g., [8]). This solution also ignores the challenges raised by complex correctness properties – e.g., if a service must satisfy "X or Y," then an allowable sequence of two actions that subvert X and Y respectively may be a vulnerability (unless the second action re-enables X). Fully exploring these problems lies beyond the scope of this paper; we will use our simple solution, but note that a suite of more advanced solutions may apply.

2.3. Vulnerabilities and the Service Order

2.3.1. The Service Order

We can use the implementation sets to order the services according to relative power. That is, one service precedes another when any implementation of the latter also supports the former; the latter is strictly more powerful.

> **Definition 5** For services s_1 and s_2, define $s_1 \leq s_2$ when $\text{IMP}(s_2) \subseteq \text{IMP}(s_1)$. Define $s_1 < s_2$ when this containment is proper.

This relation is a *partial order*: It is *irreflexive*: for no service s_1 can it be the case that $s_1 < s_1$. It is *transitive*: for services s_1, s_2, s_3, if $s_1 < s_2$ and $s_2 < s_3$ then $s_1 < s_3$. (Partial orders and lattices show up in previous security work—e.g., [2, 3, 7]—but in different contexts from here.)

We can represent this structure by a directed acyclic graph:

> **Definition 6** For a finite set S and partial order $<$, define $\mathcal{G}(S, <)$ to be the directed graph consisting of:
>
> - for each $s \in S$, a node $\mathcal{N}_s$;

- for all $s_1, s_2 \in S$, an edge $\mathcal{N}_{s_1} \longrightarrow \mathcal{N}_{s_2}$ exactly when $s_1 < s_2$.

The graph $\mathcal{G}(S, <)$ is transitively closed, since the order $<$ is transitive. However, this graph is highly redundant: if edges exist from $\mathcal{N}_{s_1}$ to $\mathcal{N}_{s_2}$ and from $\mathcal{N}_{s_2}$ to $\mathcal{N}_{s_3}$, then an edge also exists from $\mathcal{N}_{s_1}$ to $\mathcal{N}_{s_3}$, even though the first two edges comprise a path from $\mathcal{N}_{s_1}$ to $\mathcal{N}_{s_3}$. It is easily shown that such a graph has a unique *transitive reduction*, where all redundant edges – such as the one in this example from $\mathcal{N}_{s_1}$ to $\mathcal{N}_{s_3}$ – are removed. This reduced graph still reflects the original order: $s_1 < s_2$ exactly when a *path* exists from $\mathcal{N}_{s_1}$ to $\mathcal{N}_{s_2}$.

Definition 7 For a finite set S and partial order $<$, define $\widehat{\mathcal{G}}(S, <)$ to be the directed graph consisting of:

- for each $s \in S$, a node $\mathcal{N}_s$;
- for all $s_1, s_2 \in S$, an edge $\mathcal{N}_{s_1} \longrightarrow \mathcal{N}_{s_2}$ exactly when
  - $s_1 < s_2$
  - for no $s_3 \in S$ does $s_1 < s_3 < s_2$

The edges that remain in $\widehat{\mathcal{G}}(S, <)$ represent the *quantum* steps in the service set. We introduce notation for these steps, and for the differences that separate a weaker service from a stronger one.

Definition 8 For $s_1, s_2 \in S$, define $s_1 \longrightarrow s_2$ when $\mathcal{N}_{s_1} \longrightarrow \mathcal{N}_{s_2}$ in $\widehat{\mathcal{G}}(S, <)$.

Definition 9 For any $s_1 < s_2$, define DIFF(s_1, s_2) to be the differences between s_2 and s_1. If the implementation sets are specified via a requirement list, then DIFF(s_1, s_2) would just be the set difference between the requirements for IMP(s_2) and those for IMP(s_1).

Definition 10 We write $s_1 \xrightarrow{\delta} s_2$ when $s_1 \longrightarrow s_2$ and $\delta =$ DIFF(s_1, s_2).

2.3.2. The Structure of the Service Order

Specifying and manipulating the quantum steps that separate services depends highly on the context of the particular service set. However, in each particular context, considering the differences that characterize quantum steps raises two questions:

- Are the individual quantum steps indeed quantum?
  - Can there exist sub-quanta? If $s_1 \xrightarrow{\delta} s_2$, can we split δ into δ_1 and δ_2, and then introduce a new service s' with $s_1 \xrightarrow{\delta_1} s' \xrightarrow{\delta_2} s_2$?
  - Is the "quantum" step in fact a continuum? Is the continuum even a total order?
- Are the quantum steps independent? Suppose:

  $$s_1 \xrightarrow{\delta_1} s_2 \xrightarrow{\delta_2} s_3$$

  Can there exist services that accumulate these steps in a different order? For example, consider the possibility of an s' such that:

  $$s_1 \xrightarrow{\delta_2} s' \xrightarrow{\delta_1} s_3$$

  In general, should straight lines be lattices? That is, suppose $\widehat{\mathcal{G}}(S, <)$ has a straight-line structure:

  $$s_1 \xrightarrow{\delta_1} s_2 \xrightarrow{\delta_2} ... \xrightarrow{\delta_k} s_{k+1}$$

  Often the rest of the lattice has been overlooked: the graph should really consist of all s_P such that:
  - $P \subseteq \{1, ..., k\}$ (with $s_{\emptyset} = s_1$ and $s_{\{1,...,k\}} = s_{k+1}$)
  - $s_P \xrightarrow{\delta_i} s_{P'}$ (when $i \notin P$ and $P' = P \cup \{i\}$)

2.3.3. Vulnerabilities and Ordering

The ordering on services leads to two rules on vulnerabilities. First, vulnerabilities move up in the order:

Theorem 1 Let s_1 and s_2 be services. If $s_1 < s_2$, then VULN$(s_1) \subseteq$ VULN(s_2).

Second, vulnerabilities—coupled with an attack that removes the difference between services—move down in the order:

Theorem 2 Let s_1 and s_2 be services, with $s_1 < s_2$. If $B \in$ VULN(s_2) and action A subverts DIFF(s_1, s_2), then $[A; B] \in VULN(S_1)$.

We also note the closure rule from Section 2.2 characterizing an implicit closure property for vulnerability sets:

Closure Rule If $B \in$ VULN(s) and action A satisfies FUN$_A^{-1}$(PRE$_B$) $\neq \emptyset$, then $[A; B] \in$ VULN(s).

2.4. A Systematic Methodology

The theoretical structure from Section 2.3.1 provides the foundation for a systematic approach to organizing services into security taxonomies.

We begin with a set S of proposed services. We first consider construction of the service hierarchy:

- First, specify, for each $s \in S$, the hardware and software configurations characterizing the implementations IMP(s).

- Use these implementation sets IMP(s) to order the services (e.g., to construct the transitively closed directed acyclic graph $\mathcal{G}(S,<)$).

- Reduce $\mathcal{G}(S,<)$ to its transitive reduction $\widehat{\mathcal{G}}(S,<)$, specifying the quantum steps in the service order.

- For each quantum step $s_1 \longrightarrow s_2$ in this service order, determine the δ that separates s_1 from s_2.

- Examine these quantum steps for completeness:

  – Is the step really quantum? If it can be further subdivided, then do so. If it is a continuum, then decide on a discrete approximation – and explicitly indicate that this is only an approximation.

  – Are there sets of independent steps? If so, complete the lattice by adding any necessary new services.

Having constructed the service hierarchy, we next consider the vulnerabilities. We note that $\widehat{\mathcal{G}}(S,<)$ has minimal services (that is, a service s_2 such that no s_1 exists with $s_1 < s_2$)—these are just the sources of the directed graph.

- For each minimal service s in $\widehat{\mathcal{G}}(S,<)$, enumerate the basic vulnerabilities of s.

- Suppose $s_1 \xrightarrow{\delta} s_2$ and we have enumerated the basic vulnerabilities of s_1. We then enumerate the additional vulnerabilities of s_2. (Depending on S and δ, these may be simply be the additional risks associated with the capabilities enabled by δ).

- We then note that the complete set of vulnerabilities VULN(s) consists of the basic vulnerabilities, along with:

  – VULN(s_1), for all $s_1 < s$ (from Theorem 1)

  – $[A; B]$, for all $B \in$ VULN(s_2), $s \leq s_2$, and suitable A (from Theorem 2 and the Closure Rule).

3. Examples

In this section, we apply our methodology to electronic services provided via kiosks, since kiosk systems are representative of a wide range of security issues, from point-of-sale devices to networks to remote servers. Section 3.1 organizes the space of services and Section 3.2 enumerates the vulnerabilities. Section 3.3 then applies this methodology to potential points of attack.

We note that, although modified for this presentation, these examples are not artificial; they not only originated from customer-directed research (e.g., [1, 5, 6]), but also demonstrated by their initially unorganized format the need for a systematic methodology for security taxonomies.

3.1. Kiosk Services

To quote our earlier work [5], a kiosk is “a computer unit (usually publicly accessible) that provides information and services to authorized clients (usually not experienced with computers).” For customers interested in deploying services via kiosks, we undertook to enumerate the potential vulnerabilities of these services, and the correlation of these vulnerabilities with the power of the kiosk system configuration.

3.1.1. The Initial Hierarchy

The first step in our work was examining the range of kiosk systems. We initially saw a progressive hierarchy of systems and services:

- $K0$ *(standalone)*: providing public, seldom-altered information via a standalone kiosk (e.g., a kiosk in a public square that shows the location of stores downtown).

- $K1$ *(networked)*: providing public, rapidly-changing information via a networked kiosk system (e.g., the above kiosk, modified to also show the latest specials at the stores).

- $K2$ *(private information)*: providing private, remote information via a networked kiosk system (e.g., the above kiosk, modified to allow customers to make queries about current credit limits and the status of special orders).

- $K3$ *(transactional)*: allowing clients to change private, remote information via a networked kiosk system (e.g., the above kiosk, modified to allow customers to pay bills electronically by transferring funds from bank accounts to store accounts).

This hierarchy expresses a range of services our customers were considering, and demonstrates several principles:

- that increasing the power of services requires increasing the power of the supporting kiosk system;

- that increasing power increases the potential vulnerabilities and the concomitant security measures (e.g., $K1$ requires protecting networks; $K2$ also requires authenticating clients)—the roots of Theorem 1;
- that attacks which increase the power of systems enable attacks that apply to these more powerful systems (e.g., penetrating user interface barriers in $K2$ enables data modification attacks natural to $K3$)—the roots of Theorem 2; and
- that an ordering even exists on "power"—the roots of the service ordering.

3.1.2. Applying the Methodology

Inadequacies of the Initial Hierarchy The structure from Section 2.3.1 easily applies to this hierarchy to produce a simple straight-line graph:

$$K0 \longrightarrow K1 \longrightarrow K2 \longrightarrow K3$$

Going from $K0$ to $K1$ adds a network; from $K1$ to $K2$ adds private information and the need for authentication; from $K2$ to $K3$ adds the ability to make permanent changes.

However, when we follow the methodology from Section 2.4, we see that this structure is incomplete: the δ that characterize the above steps are not quantum, but are independent.

- Whether a kiosk is networked is independent of other service characteristics. Networks add the ability to interact with remote systems and databases—to extend in space the reach of the kiosk. Furthermore, the degree of this extension is not binary, and indeed not even totally ordered.
- Whether a service involves private information is independent of other service characteristics. Furthermore, privacy concerns can be refined to apply to data provided by the client, data returned to the client, or both.
- Whether a service involves permanent changes is independent of other characteristics. Furthermore, defining what constitutes a "permanent change" is tricky. In some sense, all services involve permanent changes, since we cannot roll back the user's experience. However, if we limit "change" to databases, we may still include simple informational services that are logged, and we may exclude services (such as generating a ticket) that change states in ways not easily described in terms of database writes. (We have not even mentioned the ambiguity introduced by re-using the distributed systems term "transaction.")

Furthermore, the hierarchy fails entirely to describe proposed services that involve a kiosk session initiating an action whose duration exceeds that of the session.

Addressing the Inadequacies Following the methodology, we address these shortfalls by decomposing the differences in the hierarchy into six steps that are independent, and are closer to being quanta. To simplify the presentation, we consider these properties to be binary.

This decomposition yields a set of axes that describe the space of electronic services we had been considering. (However, this description is not unique, since spaces can be described by multiple sets of axes.) To keep our presentation tractable, we use high-level, somewhat subjective criteria.

Three of the six steps characterize the type of *activity* in the service:

- **Spatial Extension.** Does the service's reach extend to machines other than the immediate kiosk?
- **Temporal Extension.** Does the duration of the electronic actions initiated by a service session extend beyond the duration of the service session?
- **Permanence.** Upon successful completion, does the service make permanent (and significant) changes to the system?

The other three steps characterize the type of *data* in the service:

- **Sensitivity.** Is it critical that the system provide correct service?
- **Input Privacy.** Does the user provide private information as part of the service session?
- **Output Privacy.** Does the system provide private information to the user as part of a service session?

This set induces 2^6 different service types: a type K_P for each subset P of the six properties. Following Section 2.3.1, we order these types using a lattice, putting $K_P < K_{P'}$ when $P \subset P'$.

We offer examples for a few service types:

- **Service Type:** $\emptyset$

  Example: $K0$, a standalone kiosk which a user queries for inconsequential public data which is returned immediately.
- **Service Type:** $\{$*sensitivity*$\}$

  Example: a kiosk like the one above, that instead provides emergency exit locations.

- **Service Type:** $\{$*spacial extension*$\}$

  Example: $K1$, discussed above.

- **Service Type:** $\{$*spacial extension, input privacy*$\}$

  Example: $K2$ above, for special order status, where users authenticate themselves with passwords.

- **Service Type:** $\{$*spacial extension, input privacy, output privacy*$\}$

  Example: $K2$ above, for credit limit information.

- **Service Type:** $\{$*spacial extension, permanence, input privacy, output privacy, sensitivity*$\}$

  Example: $K4$ above.

- **Service Type:** $\{$*sensitivity, input privacy*$\}$

  Example: a standalone kiosk through which a user accesses government services by entering private information, which the kiosk uses to fill out and print an application form (which the user then mails).

- **Service Type:** $\{$*spacial extension, temporal extension, sensitivity, input privacy, output privacy*$\}$

  Example: a networked kiosk through which a user accesses government service by entering private information, which the kiosk submits electronically. The response is returned via physical mail later.

We provide a more a thorough enumeration of examples in Chapter 1 in [5].

3.2. Kiosk Vulnerabilities

In Section 3.1.2 above, we presented 2^6 different classes of kiosk service. Enumerating the vulnerabilities associated with these types of services in a brute force or ad hoc way would be exhausting, and would also fail to communicate the tradeoffs between service features and service risks.

However, Section 3.1.2 presented these 2^6 classes by organizing them into a partial order (which in this case consists of a simple lattice). We can then follow our methodology by using this structure to clearly and concisely enumerate the vulnerabilities. In this section, we illustrate this by a set of quick examples.

Basic Vulnerabilities of the Minimal Service The minimal kiosk service is subject to physical and electronic attacks on the machine itself, in order to deny service or to change the information displayed.

Basic Vulnerabilities of Intermediate Services We then enumerate the additional vulnerabilities brought on by moving upward in the lattice. We cite an incomplete list of examples:

- **Spatial Extension.** Networked systems are subject to the vulnerabilities of disruptions, alteration, or eavesdropping of communications.

- **Temporal Extension.** Systems that permit users to initiate services that persist longer than the kiosk session introduce vulnerabilities associated with authentication, rights, and ease of launching these spawned processes. For example, temporal extension may permit using a brief session to launch attacks consisting of computationally-intensive and lengthy processes—attacks which temporal containment would render infeasible by forcing the perpetrator to remain exposed at the kiosk.

- **Permanent Changes.** Systems which permit users to make substantial, permanent changes to system state are directly subject to violation of integrity errors.

- **Input Privacy** Systems that require users to enter private information are subject to the vulnerability of exposure or alteration of that information.

- **Output Privacy.** Similarly, systems that return private information to the user are also subject to the vulnerability of exposure or alteration of that information.

- **Spatial Extension and Privacy.** A networked kiosk system which involves private information risks exposing this information via the network.

- **Temporal Extension and Output Privacy.** A system in which users launch services that later return private information risks exposing this information to unauthorized persons, because user authentication performed for the service session may no longer apply.

Upward Inheritance Following Theorem 1, a kiosk system with some set P of the properties from Section 3.1.2 will be subject to the vulnerabilities from any $P' \subseteq P$ in the enumeration of basic vulnerabilities..

Downward Inheritance Following Theorem 2, a kiosk system with some set P of the properties from Section 3.1.2 will also be subject to the vulnerabilities from any $P' \supseteq P$, providing an enabling attack exists that adds the properties DIFF(P', P). For example, a kiosk system consisting of a customized Web browser may enforce temporal containment through a line of code that terminates all applets when the user completes his or her session. An attacker who has access to the internals of the software might add temporal extension—and thus enable new forms of attack—by disabling this constraint.

3.3. Kiosk Points of Attack

Another aspect of vulnerability analysis consists of enumerating the points of attack in a system (and then considering how the vulnerabilities of this system can be realized via threats carried out at these points of attack).

The Points of Attack For example, a networked kiosk system might consist of the following components:

- a set of kiosks, each of which consists of a physical shell housing the internal kiosk operating system and software, accessed via the user interface;
- a network connecting the kiosks to each other and to the central host; and
- various remote sites, connected to the central host via remote lines.

Each of these components suggests a point of attack:

- the kiosk user interface;
- the physical environment of the kiosk;
- a fake kiosk;
- the internal kiosk software;
- the network;
- the insider who writes and controls the software at the kiosks and the central site;
- the remote lines.

An Initial Attempt to Enumerate Threats The next step is to enumerate the threats possible at each point of attack. However, we found in our practical work that using the above set alone leads to a disorganized, awkwardly referential presentation. For example, exhaustively enumerating the threats that can be carried out at the kiosk user interface includes listing threats such as the following:

- Using a bug or trapdoor to gain access to the operating system, then adding a Trojan Horse to tamper with how user data is processed.
- Using a bug or trapdoor to gain access to the operating system, then using this control to penetrate the software front-end at the main host, and then plant Trojan Horses there.

Enumerating the threats applicable at the kiosk software point of attack, the network point of attack, and the insider point of attack requires referring back to this subset of the user interface threats, with implied modifications.

Exhaustive enumeration leads to an extreme lack of clarity. Certain points of attack have natural threats. Not recognizing the inherent structure in the vulnerability set can result in burying these natural threats in the lists compound attacks at other points of attack. In the above example, the natural threats possible at the kiosk operating system level are never explicitly listed. The reader would go to the OS row, only to find a pointer back to the user interface row, where sufficiently close inspection reveals the natural OS attacks buried in compound attacks of the above form.

Not recognizing the inherent structure in the vulnerability sets also leads to a lack of flexibility. Suppose one discovers another attack possible for an adversary with operating system access. This attack makes new compound attacks possible. Since no central list exists of natural OS attacks, one must manually generate and insert all the new compound attacks throughout the list. This disorganization introduces many opportunities for error.

Similarly, suppose one wishes to use the enumeration of vulnerabilities to consider countermeasure. One can defend against a compound attack of the above forms by defending against the natural attack at the OS level, and also by defending against the enabling attack that granted access to the OS. Exhaustive enumeration will not make this defense structure clear—the defenses against the natural OS attacks will be buried in defenses against compound attacks, scattered throughout the table.

Applying the Methodology Our approach to constructing taxonomies of services and vulnerabilities provides a way to cleanly and concisely enumerate these threats. This applicability is suggested by the very structure of the above-quoted threats: "subvert the difference between the user interface and the kiosk software, then carry out a kiosk software attack."

To follow our methodology, we construct the service order by enumerating the quantum steps:

- The user interface precedes the kiosk software, since any attack possible via the user interface is possible by accessing the internal software. A bug or trapdoor in the user interface is an attack that subverts the difference.
- The physical kiosk similarly precedes the kiosk software. Physical penetration is an attack that subverts the difference.

- A fake kiosk similarly precedes the kiosk software. Copying software and connecting to a real kiosk (or its network) are attacks that subvert the difference.

- The kiosk software precedes the insider, since anything that can be done via tampered software can be done by an insider. Penetration of the software front-end to provide full system access is a potential attack subverting the difference.

- The network similarly precedes the insider. Penetration of the network front end to provide full system access is a potential attack subverting the difference.

- The remote line similarly precedes the insider. Penetration of the front end on this connection is a potential attack subverting the difference.

We can enumerate the threats possible at these points of attack simply by:

- moving up the order and enumerating the basic attacks possible at each point;

- noting that Theorem 1 means that any attack possible at lower point is possible at a higher point; and

- noting that Theorem 2 means that attacks at higher points (e.g., using software access to plant Trojan Horses) can be carried out at lower points (e.g., the user interface) by first carrying out an attack subverting the difference (e.g., using a bug or trapdoor in the user interface).

In Chapter 2 in [5], we use this new approach to provide a more complete enumeration.

4. Conclusions

Using the inherent organization of a service set to enumerate vulnerabilities removes redundancy and awkwardness from an ad hoc or brute force approach.

- Collecting in one spot the natural threats for a point of attack makes it easier to communicate them, and to add and delete threats.

- Using the structure of a system to bring out the compound and inherited structure of many threats makes it easier to produce a complete list, and to construct structured countermeasures.

- Using the inherent structure of threats to organize a listing leads to a more concise presentation. Suppose we have a linear order of n points of attack, each with k natural threats and r subverting attacks that transform the point of attack to the next level. A brute force enumeration requires listing $\Omega(kn^2 + kr^n)$ threats.[1] Our methodology lets us specify this complete set with by listing only $O(nk + nr)$ threats.

As we discussed in Section 1, we offer this work as an attempt to fill a perceived gap. One area of future work will lie in refining and validating our proposed methodology: not against specific services but against classes of services. How well does our session security taxonomy express risks in various approaches to payment over the Internet, or to the latest Web-based front-ends to existing information and data services? Other promising areas may lie in identifying and addressing other gaps.

Acknowledgments

We are grateful to our colleagues Gary Christoph, Howard Gobioff, Judy Hochberg, Marianna Kantor, Mike Murphy, Tony Warnock, and Bonnie Yantis for their helpful comments and advice.

References

[1] S. Adelson, R. Rivenburgh and S.W. Smith. *Enforcing Closure of Subwebs and Expansive Web Sites*. Los Alamos Unclassified Release LA-UR-95-4410, Los Alamos National Laboratory. December 1995.

[2] D. Bell and L. LaPadula. *Secure Computer Systems: Mathematical Foundations and Model.* Technical Report M74-244, MITRE Corporation. May 1973.

[3] D.E. Denning. "A Lattice Model of Secure Information Flow." *Communications of the ACM*. 19: 236-243. May 1976.

[4] D. Fiske. "Another ATM Story." *The Risks Digest*. Volume 6, Issue 66. 21 April 1988.

[5] J. Hochberg, S.W. Smith, M. Murphy, P. Pedersen, and B. Yantis. *Kiosk Security Handbook*. Los Alamos Unclassified Release LA-UR-95-1657, Los Alamos National Laboratory. May 1995.

[6] G. Morris, T. Sanders, A. Gilman, S. Adelson, and S.W. Smith. *Kiosks: A Technological Overview*. Los Alamos Unclassified Release LA-UR-95-1672, Los Alamos National Laboratory. May 1995.

[7] S. W. Smith. *Secure Distributed Time for Secure Distributed Protocols.* Ph.D. thesis. Computer Science Technical Report CMU-CS-94-177, Carnegie Mellon University. September 1994.

[8] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993.

[1] The ith level has k natural threats, $k(i-1)$ inherited threats, and $k\sum_1^{n-i} r^j$ subversion threats. The $\sum_i k(i-1)$ gives the kn^2, and the $\sum_i \sum_j r^j$ reduces to $\sum_{l=1}^{n} lr^{n-l}$ which we can bound below by r^n.