A Defense-Centric Taxonomy Based on Attack Manifestations

Kevin S. Killourhy, Roy A. Maxion and Kymie M. C. Tan
ksk@cs.cmu.edu, maxion@cs.cmu.edu and kmct@cs.cmu.edu

Dependable Systems Laboratory
Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213 / USA

Abstract

Many classifications of attacks have been tendered, often in taxonomic form. A common basis of these taxonomies is that they have been framed from the perspective of an attacker – they organize attacks with respect to the attacker's goals, such as privilege elevation from user to root (from the well known Lincoln taxonomy). Taxonomies based on attacker goals are attack-centric; those based on defender goals are defense-centric. Defenders need a way of determining whether or not their detectors will detect a given attack. It is suggested that a defense-centric taxonomy would suit this role more effectively than an attack-centric taxonomy. This paper presents a new, defense-centric attack taxonomy, based on the way that attacks manifest as anomalies in monitored sensor data.

Unique manifestations, drawn from 25 attacks, were used to organize the taxonomy, which was validated through exposure to an intrusion-detection system, confirming attack detectability. The taxonomy's predictive utility was compared against that of a well-known extant attack-centric taxonomy. The defense-centric taxonomy is shown to be a more effective predictor of a detector's ability to detect specific attacks, hence informing a defender that a given detector is competent against an entire class of attacks.

1. Introduction

There are many taxonomies of computer attacks (see, for example, [1, 3, 8, 9, 10, 11, 12, 14, 16, 30], and others; several are nicely surveyed in [17]). Although it was undoubtedly not their creators' intention, these taxonomies tend to serve the interests of attackers, as well as their original goals of providing organizational structure for classifying attacks. These and other similar taxonomies are attack-centric – they represent and classify attacks from an attacker's perspective. For example, the well-known MIT Lincoln Laboratory taxonomy [16] groups a number of attacks into a category called user-to-root. An attacker can choose attacks from this group to achieve the goal of elevating himself from user to root on the victim system. If the selected attack fails, another can be chosen from the same user-to-root taxonomic class, and then yet another, until the attacker's objective is achieved. In this sense, attack-centric taxonomies may constitute as much or more of a boon for the attacker as for the defender.

An important and sensible goal for an attack taxonomy, however, should be to help the defender. Much more useful to a defender than an attack-centric taxonomy would be a taxonomy of attacks arranged to aid the defender – a defense-centric taxonomy. Such a taxonomy would classify attacks in a way that groups together those attacks that could be defended against similarly. For example, it would be enormously useful to know that any attack in a given class could be detected by the same detector; if one attack in a class can be detected, they can all be detected. In such a case, when a new attack emerges, and it falls into a given class, it will be clear either that it can be detected with extant detectors, or that a new detector will need to be written.

This paper shows how a defense-centric attack taxonomy was constructed, in accordance with taxonomic rules, using attack manifestations – how attacks appear in sensor-stream data – as classification features. Categories from the well-known MIT Lincoln Laboratory attack-centric taxonomy [16] were used as a point of comparison to show that, for a defender, the new defense-centric taxonomy offers greater utility for defenders.

2 Objective and approach

The objective of the present work is to produce a new attack taxonomy, termed a defense-centric taxonomy, that organizes attacks by virtue of the way they manifest as anomalies in sensor data. An attack that can be detected in system-

call sensor data (monitored from the operating system) is said to *manifest* in system-call data; the *way* it manifests is central to the taxonomy.[1] Anomaly-based detectors play an important role in detecting not only extant attacks, but also novel attacks; hence the present emphasis on anomaly-based detection systems. Given an anomaly-based detection system, any attack that manifests as a particular kind of anomaly would be classified according to that anomaly type. Consequently, all attacks similarly classified would manifest as the same kind of anomaly, and hence be detectable by any detector that is capable of detecting that type of anomaly (ergo steps 9 and 10 below).

The approach is straightforward, albeit laborious, consisting of the following broad steps.

1. Develop an attacker-defender testbed.
2. Assemble a collection of system programs which are known to be vulnerable to attack; develop attacks to exploit the vulnerabilities of these system programs.
3. Run these system programs, launching the attacks against them, to observe their behavior under attack; gather attack records from sensor data.
4. Run these system programs again, this time native, to observe their normal behavior; gather normal records from sensor data.
5. Extract attack manifestations from the program's behavior as monitored by sensor data on the testbed.
6. Build a taxonomy that is defense-centric.
7. Check that it obeys classic taxonomic rules.
8. Validate the taxonomy by acquiring convergent evidence from an intrusion-detection system.
9. Choose an attack-centric taxonomy against which to compare the new defense-centric taxonomy.
10. Determine whether the manifestations mirror the classes of the two taxonomy types, or not.

3 Related work – attack taxonomies

This section provides an overview, inevitably and regrettably brief due to space limitations, of some of the (surprisingly voluminous) existing work that addresses the issue of attack taxonomies. The selected references below are believed to be the best known and most representative of the taxonomies in the open literature, but they are by no means the only ones available; there are many that could not be mentioned here. The taxonomies reviewed here have been grouped into rough categories: flaw classifications, signature classifications, and attack-effect classifications.

As a precursory note, Puketza and his colleagues were implicit early promoters of the importance of taxonomy in the field of intrusion detection. Although they did not design a taxonomy, per se, they plainly stated a useful taxonomic criterion: the classes should ideally "be selected such that, within each class, either the IDS[2] detects each intrusion, or the IDS does not detect any intrusions" [19, p.723]. They note that software testers term this distinction *equivalence partitioning*, which bears a similarity to the present work on defense-centric taxonomies; it doesn't constitute a taxonomy, but it's suggestive of a mechanism that could be used by a defender. In a defense-centric taxonomy, all attacks in one class should be detectable by the same intrusion detection mechanism.

Flaw classification. Landwehr and his colleagues devised a taxonomy of the types of program security flaws (e.g., buffer overflows) that facilitate attacks [12]. Their taxonomy was meant to identify problematic aspects in the system design process, a period during which security flaws are likely to be introduced into code. As such, this taxonomy was designed to help system designers and programmers create more secure systems. Matt Bishop took a similar approach with his vulnerability taxonomy, designed to elucidate those characteristics of a program that might allow it to be exploited in an attack [3]. By being aware of such characteristics, design auditors can more easily detect vulnerabilities before they are found and exploited by attackers. In this regard, Bishop's taxonomy of vulnerabilities is similar to Landwehr's taxonomy of security flaws. Aslam [2] proposed a taxonomy of flaws that he referred to as "security faults." He broadly classified attacks into three high-level taxonomic classes: coding faults introduced during software development; operational faults which result from improper software installation; and environment faults when a program is used in an environment for which it was not intended. Operational faults and coding faults are further subdivided. The primary motivation of this taxonomy was to make attack classes unambiguous. Lack of ambiguity was important for Aslam's goal of organizing known security flaws into a vulnerability database. Krsul [10] built upon Aslam's work and constructed his own taxonomy of software vulnerabilities in which the class to which a vulnerability belongs depends primarily on programmers' assumptions. For example, one common assumption is that the length of an environment variable passed to a program is of at most a certain length. This assumption sometimes causes programmers to copy the environment variable to a memory location of insufficient length, thereby causing a buffer overflow. Vulnerabilities are grouped based on the similarity of the mistaken assumption that introduced the vulnerability. By identifying and organizing the flaws, vulnerabilities, or faults which have security implications, these taxonomies all aim to help system designers and programmers. Hence, designers and programmers, rather than system administrators and defenders, are the target audience of flaw taxonomies.

[1] Note that some attacks may not manifest in sensor data; either by accident or by design, an attack may not manifest in a way that makes it visible in a particular sensor stream.

[2] IDS: Intrusion Detection System

Signature classification. Kumar proposed a taxonomy of attacks based on the complexity of the signature by which an attack is detected [11]. Attacks manifest in sensor data by virtue of a detectable signature. The simplest class of attacks is what Kumar called the existence class; the attack manifests (in sensor data) as a single event which can be detected by recognizing that particular event. Other attacks may manifest as sequences of events that are detectable only with regular expressions, i.e., by finite automata. It takes less computational power to test each simple event in a sequence for equality than it takes to determine that that sequence of events matches a regular expression. Kumar's taxonomy expresses this "detection complexity" in an ordering from computationally facile to computationally demanding. Although Kumar classified attacks based on their manifestations in sensor data, as is done in the present work, his manifestations were brought to bear on the difficulty of signature detection, not on the presence of particular kinds of manifestations. In addition to being quite abstract, Kumar's taxonomy was specifically tailored as an aid in signature-based detection, not anomaly-based detection, which is of present concern. Since his taxonomy mainly rank orders signatures according to their complexity, it is unclear how it can be useful to defenders or researchers trying to anticipate new and novel attacks, for which signatures do not already exist.

Attack-effect classification. Many investigators have proposed taxonomies that classify attacks based on the intended effect of the attack, from the attacker's perspective. An example of an attack effect is the elevation of an attacker's privileges from user to root. Often these taxonomies incorporate the technique by which the attacker achieves this effect, such as automated password-guessing. Lindqvist and Jonsson [14] presented such a taxonomy using these two dimensions of an attack. Marcus Ranum [20] grouped attacks into eight intuitive categories based on techniques used by the attacker: social engineering, impersonation, exploits, transitive trust, data driven, infrastructure, denial of service, and magic. Howard and Longstaff [9] incorporated classes of attack techniques used, and attack effects desired, into their incident taxonomy, which also includes classes of attackers and objectives. Daniel Weber [30] proposed an attack taxonomy based on three elements: the level of privilege required by the attacker (e.g., a local user account on the target machine), the means by which the attack proceeded (e.g., exploitation of a software bug), and the intended effect of an attack (e.g., temporary denial of service) or the privilege level obtained in the attack (e.g., administrative or 'root' access for the attacker). Lippmann and his colleagues [15, 16] at MIT's Lincoln Laboratory adapted Weber's taxonomy, reducing it to fewer, more intuitive classes, based solely on the effect of the attack.

In taking an attack-effect perspective, these taxonomies are *attack-centric* in that they classify attacks based on features potentially unknowable by a defender; for example, a defender is not likely to be aware of vulnerabilities in software, at least not in a way that can facilitate effective detection-level defenses against exploitations. Consequently, the utility to a defender of these attack-centric taxonomies is questionable.

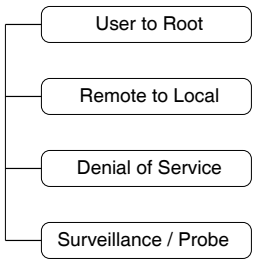

Figure 1: The four-class, attack-centric taxonomy used by Lincoln Laboratory in the 1998 DARPA IDS evaluations.

Possibly the best known of these taxonomies, due to its extensive exposure during the DARPA 1998-99 intrusion detection evaluation program, is the Lincoln Laboratory attack taxonomy [16]. The Lincoln taxonomy was attack-centric, and contained four classes: user-to-root, remote-to-local, denial-of-service and surveillance/probe. These will be discussed in greater depth in a subsequent section. The MIT Lincoln Laboratory taxonomy is probably one of the best known attack-centric attack taxonomies, and for that reason it is used here as a basis of comparison to highlight the differences and advantages of a defense-centric taxonomy. The Lincoln taxonomy is depicted in Figure 1.

What's wrong with the aforementioned taxonomies? Essentially, nothing is wrong with them; they simply serve purposes different from the defense-centric taxonomy addressed in this paper (except Kumar's taxonomy, whose limitations were discussed above). We would like to see if these taxonomies are as useful to a defender as one that is explicitly designed for a defender's use.

4 Criteria for an effective taxonomy

Before building a new taxonomy, it seems appropriate to consider what might constitute sensible criteria for judging its merits when finished. Although the heart of taxonomy lies in the biological sciences (e.g., [22, 23]) in which there is mild controversy regarding the correct way to construct a taxonomic classification scheme, many authors in computer security have nevertheless proposed such criteria. Recommendations tend to vary widely, with some unachievable, some redundant, and some not clearly applicable. Lough has done a rather thorough job of reviewing the various taxonomies offered by the computer security community, as well as the criteria for evaluating them [17].

A decision was made in the present work to select minimal criteria which seemed sensible and, at the same time, most consistent with guidelines in biology and systematics (the classification and study of entities with regard to their natural relationships). The criteria used here for judging the effectiveness of a taxonomy are:

- Mutual exclusivity: categories do not overlap, preventing ambiguities from arising;
- Exhaustivity: all objects or events are contained in the taxonomy;
- Replicability: repeated attempts at classification of the same objects or events, whether by the original or other experimenters, replicates faithfully the assignment of objects or events to taxonomic classes.

5 Methodology

This section presents the procedures by which the study was conducted, including a sketch of the hardware/software suite used for launching and monitoring attacks.

5.1 Construct attacker-defender testbed

An attacker-defender testbed is a carefully controlled environment on which to launch attacks on or against a target system, and to observe the effects of those attacks. It comprises, at minimum, an attacker machine and a victim (target) machine, plus a network joining the two. The testbed is isolated from non-testbed machines and from the Internet. The architecture for the attacker-defender testbed used in this work could have been selected from a range of systems: Windows, Mac, Linux, etc. Linux on an Intel-based computer was chosen because of the wide variety and ease of availability of attacks against it, and because much of the work in intrusion detection is rooted in Unix.

The victim hardware was a commercial, off-the-shelf machine equipped with a 450 MHz AMD K6 processor, 256 MB of memory, a single 8GB hard disk, and a 10 / 100 Mbps Ethernet network card. The victim operating system, installed on the hardware, was standard RedHat Linux 6.2.

The sensor stream of interest was system calls, based on the idea that sequences of operating-system calls contain anomalous manifestations of attacks; when an attack occurs, its presence is expected to be indicated by deviations from normal system-call behavior [6]. Special software (a kernel patch) is required for monitoring system calls. This work used the IMMSEC kernel patch by Somayaji and colleagues at the University of New Mexico [24]. The installation procedure is (a) obtain the source code for the version 2.2.13 Linux kernel, (b) apply the IMMSEC patch to this source code, (c) build the modified Linux kernel, and (d) install the new kernel on the victim system.

The victim system was configured to allow only `ssh` connections from the network. It was connected to a private research network largely isolated from the Internet by a firewall. Other machines on the same network were used for launching attacks and for assisting in the simulation of normal behavior.

5.2 Acquire vulnerable programs & attacks

A collection of vulnerable programs and corresponding attacks upon these programs was assembled so that attacks could be mounted against the testbed, and their manifestations observed in a controlled environment.

Programs. What makes a program vulnerable to attack is that (a) the program runs with privileges higher than those possessed by the would-be attacker and (b) some flaw in the program is susceptible to malicious exploitation. An example of such a flaw is writing to an unchecked buffer; arbitrary data is written to a fixed-length area of memory without first checking that the length of the data does not exceed the length of the buffer. A buffer overrun or buffer overflow occurs when more data is written into the buffer than the buffer can hold. When a buffer is overrun, the extra data overwrites other data structures (e.g., a return pointer which tells a program running a particular function the address of the machine instruction to execute after the function terminates). A buffer overrun can be exploited by an attacker so as to overwrite a return pointer, making it point to machine instructions that the attacker wishes to execute, e.g., to give the attacker an interactive command-line interface with superuser privileges (a so-called "rootshell"). With such a rootshell, the attacker has effectively gained all the privileges of the vulnerable program.

One kind of program that is often attacked is a system program, two examples of which are `passwd` and `tmpwatch`. A system program runs with elevated privileges (e.g., root privileges, which allow the program to read, write, modify, delete or execute any file) beyond those normally afforded a regular user of a system, making these programs favorite targets of attackers. For this study, eighteen vulnerable system programs were culled from the well-known public repository of vulnerability information, SecurityFocus [21], and installed on the victim system of the attacker-defender testbed. The eighteen programs are shown in Table 1.

`dip`	`diskcheck`	`dump`
`imwheel`	`kon2`	`ntop`
`restore`	`slocate`	`sudo`
`su`	`passwd`	`tmpwatch`
`traceroute`	`vim`	`xfs`
`xlock`	`xman`	`xterm`

Table 1: Vulnerable system programs used in the study. Vulnerabilities reside in the programs, except for `passwd`, whose vulnerability is due to a Linux kernel race condition.

Attacks. Once the vulnerabilities were established, attacks were needed to exploit the vulnerabilities. Attacks are often available as source code to a program that will automatically exploit the vulnerability. In some cases, this so called "exploit code" was available on public repositories and was downloaded. In other cases, exploit code was written from scratch using available information about the vulnerability. A selection of these exploit codes were copied and modified, making variations of an attack, each of which exploited the same vulnerability but which might manifest in different ways. The modifications were guided by previous work in which we identified methods to cloak an attack, making it harder to detect with an anomaly-based intrusion detection system [26, 29]. Once exploit code was downloaded or written, the attack consisted of compiling the exploit code and launching the resulting program against the target machine. It was confirmed that each attack worked as intended. Twenty-five attacks were collected for this project, and they are listed in Table 2.

crontabrace	kernelexecptrace[3]	sulocalefmt
dipbuff	killxfs	tmpwatchexec
diskcheckrace	kon2buff	traceroutefree
diskcheckrace[2]	ntopspy	traceroutefree[2]
dumpbx	restorecool	traceroutefree[3]
imwheelbuff	restorecool[2]	xlockfmtstring
imwheelbuff[2]	slocateheap	xmanprivs
kernelexecptrace	sudomem	xtermdos
kernelexecptrace[2]		

Table 2: The 25 attacks. Square brackets [] denote secondary [2] and tertiary [3] versions of attacks.

5.3 Gather sensor records of attack behavior

During each attack, the behavior of the vulnerable system program, in terms of sequences of system calls generated by the program, was monitored and recorded via the system-call sensors deployed on the testbed. Table 3 shows an excerpt from a system-call log. The numbers indicate the process IDs of the processes that made the system calls. In this example, four different processes were executing (i.e., processes with IDs 5260 through 5263). The names of the system calls being executed were `brk`, `lstat`, etc. The actual record specifies system-call numbers, rather than names (e.g., `brk` corresponds to system call number 45); the call names in the figure were transcribed for readability.

For each of the twenty-five attacks, sensors monitored the vulnerable system program that was exploited in the attack. In some cases, the attacker ran the system program directly, in such a way as to exploit its vulnerability. In other cases, the attacker modified the environment in which a system program ran, and waited for the system administrator (or a scheduling program that performs regular administrative tasks, e.g., the `cron` daemon) to run the vulnerable system program. Whether the attacker ran the system program directly, or it was run by other means, the sensors detected when the system program was run, and monitored the system calls it made.

5261 brk	5262 chdir	5262 fork
5261 brk	5262 lstat	5261 wait4
5261 brk	5262 open	5263 lstat
5261 lstat	5262 fstat	5263 chdir
5261 getdents	5262 fcntl	5263 lstat
5261 lstat	5262 brk	5263 open
5261 access	5262 lstat	5263 fstat
5261 fork	5262 getdents	5263 fcntl
5260 wait4	5262 lstat	5263 lstat
5262 lstat	5262 access	5263 getdents

Table 3: Excerpt of system-call stream (reading down the columns), as monitored by the testbed; numbers are process IDs, and names are system-call names.

The sensors were configured to monitor one system program at a time – just the system program being exploited in an attack. Sensors were enabled before the attack began, and remained operational for the attack's duration. When the system program under attack exited, either naturally (e.g., through an `exit` or `execve` system call) or prematurely (e.g., in an attempt to execute an illegal instruction, or a memory segmentation violation), the sensors were disabled, and the record of the system program's behavior (i.e., series of system calls) during attack was recorded to permanent storage. This record is the *attack record*. Twenty-five attack records were collected, one for each attack.

5.4 Gather sensor records of normal behavior

The attacks used to exploit the system programs were examined to identify the exact nature by which the misuse took place. For example, an attack could supply an extraordinarily long value for an environment variable, causing a buffer to overflow. In addition, the documentation accompanying each vulnerable system program (e.g., its "man page") was reviewed to collect examples of intended program usage. An intended-usage example was selected which correctly used the features of the program misused by the attacker. The vulnerable system program was invoked as described in the selected example to produce a representative instance of the normal behavior of the program.[3] The example was chosen so that this normal behavior would reflect the behavior during attack, excepting the presence of

[3] Space limitations preclude full exposition of the normal-behavior methodology which is available elsewhere.

the attack itself. As an example, if an attacker supplied an exceptionally long value for an environment variable, the selected example of intended usage would set the same environment variable to an appropriate value, as discerned from the documentation. The record of a system program's normal behavior (series of system calls) during intended usage was produced in much the same way as the record of that system program's behavior during attack. The resulting record is called the *normal record*. Twenty-five normal records were collected, one for each attack.

5.5 Extract attack manifestations

An attack that can be detected in system-call sensor data is said to *manifest* in system-call data. The manifestation comprises those sequences of system calls which are due to the presence or activity of the attack, and which would not appear if there were no attack. Attack manifestations were identified as follows.

(1) The corresponding attack record and normal record were compared to expose sequences of system calls differing between the two records. Some sequences may appear only in the attack record, while others may be missing from the attack record. (2) Sequences of instructions were extracted which, when executed, resulted in differences between sequences of system calls in normal and attack records. The exploit code used in the attack, as well as the source code of the vulnerable system program, were consulted to help identify the sequences of instructions due to the presence of the exploit code. If such instructions appear in the exploit code, the corresponding system calls in the attack record are due to the attack. (3) Supplemental tools were used to gather supporting evidence regarding the effects of the attack on the system call record. For example, the `strace`[4] program can be used to provide a detailed report of the interaction between a program and the operating system. The `strace` program was used to associate the presence of an attack with specific sequences of system calls in the attack record. To ensure that the `strace` records showed an alternative view of the same behavior as the corresponding attack or normal record, it was confirmed that the information included in both records (i.e., sequences of system calls) matched; they did.

5.6 Build a defense-centric taxonomy

A taxonomy is a classification aid, and any classifier must do its job on the basis of features that can discriminate one object or event from another. Consequently, the first step in building a taxonomy is to determine the features upon which the classification will be based. In the present case, the features need to exist in the attack manifestations previously discussed, and they need to be accessible to anomaly-based detection systems, because these are the types of detection systems that will form the basis of future defense-centric strategies [4, 5, 13].

Features of anomalous sequences were previously studied by Maxion and Tan [18]. In their discussion of coverage of anomaly detectors, they identified foreign symbols and foreign sequences as specific types of anomalies that can occur in any set of sequential, categorical data (like the system-call data used here). In later work they identified, and discussed in detail, another fundamental anomaly type, the minimal foreign sequence, which is a refinement of the foreign-sequence concept [27, 28]. Foreign symbols and minimal foreign sequences were used in the present work as basic taxonomic features of attack manifestations in system-call sensor data. Close examination of the attack records revealed two additional manifestations that were also used as features – dormant sequences and non-anomalous sequences. The four features, observed in sensor data as attack manifestations, are defined as follows, with examples shown in Table 4:

1. Foreign symbol: the attack manifestation contains a system call which never appears in the normal record.

2. Minimal foreign sequence: the attack manifestation contains a system-call sequence which itself never appears in the normal record, but all of whose proper subsequences do appear in the normal record.

3. Dormant sequence: the attack manifestation contains a sequence of system calls which matches a proper subsequence from the normal record but does not match the full sequence, because it is, for example, truncated.

4. Non-anomalous sequence: the attack manifestation is a sequence of system calls which exactly and entirely matches a normal sequence; that is, the attack produced a set of system calls that matched the system calls for a normal, attack-free program.

Sequence type	Example
Normal Sequence	A A A B B B
Foreign Symbol	A A C B B B (the C is foreign)
Normal Sequence	A A A B B B
Min. Foreign Seq.	A A A A B B (4 As is min. foreign)
Normal Sequence	A A A B B B
Dormant Sequence	A A A B (missing 2 Bs)
Normal Sequence	A A A B B B
No Anomaly	A A A B B B (no difference)

Table 4: Sequence types and examples.

[4] The strace program is used to intercept and record the system calls that are made by a program and the signals that are received by the program. It is a commonly used debugging tool [25].

The hypothesis of this study is that the class to which an attack belongs ought to predict whether or not a particular intrusion-detection system will detect a given attack. The information upon which this prediction is based is the presence or absence of the features listed above. The features themselves were used as taxonomic classes.

The features are not mutually exclusive; for example, it is possible for a manifestation to contain both foreign symbols and minimal foreign sequences. There is a precedence, however, among the taxonomic classes which reflects the difficulty of detection. A foreign symbol takes precedence over a minimal foreign sequence, which is harder to detect. Likewise, a minimal foreign sequence takes precedence over a dormant sequence; and a dormant sequence takes precedence over a sequence containing no anomalies. To remove any ambiguity in the class to which an attack belongs, a decision procedure, shown in Table 5 , was constructed for uniquely identifying the class of any attack.

Class 1 (FS): If an attack's manifestation contains one or more foreign symbols, classify the manifestation as *foreign symbol*.
Class 2 (MFS): If an attack's manifestation contains no foreign symbols, but does contain one or more minimal foreign sequences, classify the manifestation as a *minimal foreign sequence*.
Class 3 (DS): If an attack's manifestation contains no foreign symbols or sequences but does contain a dormant sequence, classify the manifestation as a *dormant sequence*.
Class 4 (MNA): If an attack's manifestation contains no foreign symbols or sequences, and no dormant sequences, the manifestation is indistinguishable from the normal record; classify the *manifestation* as *not anomalous*.

Table 5: Procedure for determining class of attack.

For each of the twenty-five attacks in this study, the features of each attack were identified. In Table 6, each of the twenty-five attacks is listed, one per row. The four features are listed in the first four columns of the table. The presence or absence of each of the four features is denoted by a mark in the appropriate column. Using the decision procedure, the class to which each of the twenty-five attacks belongs is determined. It is listed in the fifth column of Table 6.

5.7 Check that taxonomic rules are obeyed

The new defense-centric taxonomy should meet the criteria established in Section 4 for acceptable taxonomies. Briefly, these criteria are (1) mutual exclusivity, (2) exhaustivity, and (3) replicability. Both theoretical and empirical evidence were collected to confirm that the taxonomy meets each of these criteria.

Theoretical evidence that each of these criteria is satisfied was obtained by reviewing the decision procedure described in Table 5. For mutual exclusivity, the decision procedure was reviewed to determine that it was not possible for an arbitrary attack manifestation to be identified with more than one class. For exhaustivity, the decision procedure was reviewed to determine that it was not possible for an attack to "fall through the cracks" and belong to none of the classes. For replicability, the language of the decision procedure was reviewed to ensure that there was no ambiguity which would cause different taxonomists to decide that an arbitrary attack's manifestation might belong to different classes.

Empirical evidence was gathered by determining whether or not the collected attacks and the classifications of these attacks violated any of the taxonomic properties. For mutual exclusivity, it was confirmed that none of the attacks belonged to multiple classes. For exhaustivity, it was confirmed that all the attacks belonged to one of the four classes. For replicability, it was confirmed that diverse classifications of each of the attacks (e.g., different tools, different evaluators, etc.) produced the same results, i.e., that the same attacks were always assigned to the same categories.

5.8 Validate taxonomy using IDS evidence

For the purposes of this study, the taxonomy's utility to a defender lies in its ability to predict whether or not an intrusion detection system (IDS) will detect an attack based on the attack's taxonomic classification. The taxonomy's utility is determined by running all 25 attacks through an intrusion detection system to verify that the detector "sees" all the attacks in a class in the same way; i.e., if the detector scores attack detections on a scale from 1–3, then all attacks in a class should get the same score. The Stide anomaly-based IDS was used, because it operates on the same kind of sequential system-call data as used here, and because the taxonomy was designed around attack manifestation features significant to the performance of sequence-based detectors in general, of which Stide happens to be one. Stide is described elsewhere in great detail [7].

For each of the 25 attacks, the normal record for the attack was run through Stide's training mode to establish a model of normal behavior. The attack record was then run through Stide's detection mode, and a score was assigned on the basis of the extent to which Stide detected the attack. Stide was configured with a locality frame of 1, so that all of the anomalies visible in the current detector window would be reported, regardless of whether or not there was a recent

history of anomalies (this is the most stringent configuration of Stide). The detector window size was varied broadly, from 1 to 15, inclusive. If one or more anomalies were reported at every window-size setting, the attack was judged to be *always detectable*, and a score of 3 was assigned for the attack. If one or more anomalies were reported at some window settings, but none were reported at others, the attack was ruled to be *sometimes detectable*, and a score of 2 was assigned for the attack. If no anomalies were reported at any window setting, the attack was judged to be *never detectable*, and a score of 1 was assigned for the attack. The attack scores are given in the last column of Table 6.

To verify that the performance of the intrusion detection system was predicted by the defense-centric taxonomy, the defense-centric taxonomic class of each of the twenty-five attacks was compared to the detector score for the attack. If the taxonomic class is a good indicator of the score, then the taxonomy is said to predict detector performance. Table 6 shows that the predictions are perfect.

5.9 Choose contrasting attack-centric taxonomy

This paper asserts that a defense-centric taxonomy is a better predictor of detector performance than an attack-centric taxonomy. To validate this claim requires a comparison to be made between the two taxonomy types. The attack-centric taxonomy from Lincoln Laboratory [16] (hereafter denoted *Lincoln taxonomy*; also see Figure 1) was selected for side-by-side comparison, because it is well known and familiar to the computer security community, and because it is attack centric (i.e., attacks are classified according to the attacker's goal). With respect to the 25 attacks used in this study, the Lincoln taxonomy obeyed taxonomic requirements of mutual exclusivity and replicability, but not exhaustivity.

Four of the 25 attacks did not fit the Lincoln taxonomy. Two attacks (`slocateheap` and `xmanprivs`) would be used by an attacker to elevate privileges to those of a system-level (not root) user, e.g., users granted special privileges to manage online documentation or filesystem-wide indexes of files. Two other attacks (`crontabrace` and `diskcheckrace`) would be used by an attacker to create files in unauthorized locations. None of these four attacks can be assigned to any of the Lincoln classes.

To accommodate this shortcoming, the origins of the Lincoln taxonomy were examined. The Lincoln taxonomy was derived from a more elaborate taxonomy by Weber [30], apparently by grouping together classes from the Weber taxonomy into more general classes in the Lincoln taxonomy. The four attacks which could not be classified by the Lincoln taxonomy were found to belong to classes in the Weber taxonomy. By creating a fifth class and supplementing the Lincoln taxonomy with this fifth class, called "System access / Alter data," an attack-centric taxonomy was created which met all the requirements such that a comparison between the attack-centric Lincoln taxonomy and the present defense-centric taxonomy could proceed.

	Features					
	Contains a Foreign Symbol	Contains a Minimal Foreign Sequence	Contains a Dormant Sequence	Is Not Anomalous	Taxonomic Class	IDS Score
1. kernelexecptrace	x				FS	3
2. imwheelbuff	x				FS	3
3. slocateheap	x				FS	3
4. sudomem	x	x			FS	3
5. dipbuff	x	x			FS	3
6. traceroutefree	x	x			FS	3
7. crontabrace		x			MFS	2
8. dumpbx		x			MFS	2
9. kernelexecptrace[2]		x			MFS	2
10. killxfs		x			MFS	2
11. kon2buff		x			MFS	2
12. ntopspy		x			MFS	2
13. restorecool		x			MFS	2
14. sulocalefmt		x			MFS	2
15. traceroutefree[2]		x			MFS	2
16. xlockfmtstring		x			MFS	2
17. xmanprivs		x			MFS	2
18. xtermdos		x			MFS	2
19. imwheelbuff[2]			x		DS	1
20. kernelexecptrace[3]			x		DS	1
21. diskcheckrace				x	MNA	1
22. diskcheckrace[2]				x	MNA	1
23. restorecool[2]				x	MNA	1
24. tmpwatchexec				x	MNA	1
25. traceroutefree[3]				x	MNA	1

Table 6: Attacks, detected features, taxonomic classification, and IDS performance scores. Taxonomic classes: FS (class 1, foreign sequence), MFS (class 2, minimal foreign sequence), DS (class 3, dormant sequence), MNA (class 4, manifestation not anomalous). IDS scores: 1 (never detectable), 2 (sometimes detectable), 3 (always detectable). Classes 3 and 4 are assigned the same score of 1, because neither is detectable by the IDS. See discussion, Section 5.8.

5.10 Compare attack/defense-centric taxonomies

Each of the 25 attacks was classified according to both the Lincoln attack-centric taxonomy and the new defense-centric taxonomy. Each attack belongs to a pair of classes, one attack-centric and one defense-centric. If the two taxonomies are equivalent, from a defender's perspective, then a single attack-centric class should always be paired with a single defense-centric class, with no overlap. This was not found to be the case

Attacks in three of the Lincoln classes mapped to multiple defense-centric classes (the surveillance/probe class contained no attacks, because there is such controversy over whether or not a probe constitutes an attack). Four attacks in the Lincoln user-to-root class mapped to four different defense-centric classes. Attacks spanning three Lincoln classes mapped to just one defense-centric class. The mappings are shown in Figure 2; unfortunately, space limitations preclude a more detailed illustration.

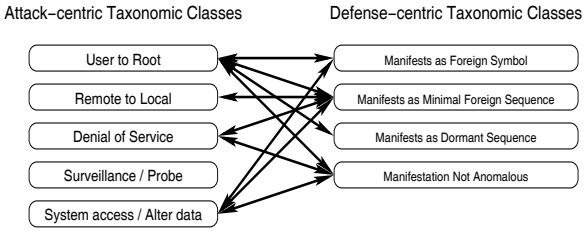

Figure 2: Mapping of 25 attacks between the five Lincoln Laboratory attack-centric classes (see Section 5.9) and the four new defense-centric classes.

6 Results and discussion

Twenty-five attacks were categorized in accordance with each of two taxonomies – an existing attack-centric taxonomy (from Lincoln Lab) and a new defense-centric taxonomy. Two major outcomes were: (1) a validation of the claim that a defense-centric taxonomy can predict whether or not an intrusion detection system is capable of detecting all attacks in particular classes of attacks; and (2) a comparison of the predictive power of an attack-centric taxonomy vs. a defense-centric taxonomy.

In every case, the classification of an attack according to the defense-centric taxonomy perfectly predicts the detector's ability to detect the attack. This outcome demonstrates that a defense-centric taxonomy is a capable predictor of whether or not an intrusion detection system can detect a given attack and all attacks in a class. Demonstrating this was a major goal of the research. The results are depicted in Table 6.

In terms of comparing the predictive capabilities of attack-centric vs. defense-centric taxonomies, from the perspective of a defender who needs to know whether or not his/her detector will detect a particular attack, the defense-centric taxonomy was an accurate predictor, whereas the attack-centric taxonomy was not. The primary reason for the inaccuracy of the attack-centric taxonomy is exemplified by there having been four different user-to-root (attack-centric) attacks that mapped to four different defense-centric classes. This suggests that knowing an attacker's goals, e.g., privilege elevation from user to root, tells a defender little about what evidence would be left behind in sensor data, should the attacker's objective be accomplished. Conversely, that many attack-centric classes map to a single defense-centric class suggests that it would be difficult for a defender, looking at the manifestation of an attack in the sensor data available, to discern what the attacker was attempting to accomplish with the attack. The results are illustrated in Figure 2.

It is not the purpose of this work to disparage any attack-centric taxonomy. Attack-centric taxonomies have their place, but not necessarily in the service of intrusion detection. For example, a defender could use the attack-centric class of an attack to estimate the severity of the attack and its likely effect on the defender's organization. If a novel remote-to-local attack were discovered on a weekend, the system administrator might be called into work to take immediate action to guard against the attack, whereas if the attack were denial-of-service, the defensive action could be postponed until Monday. The two taxonomy types could complement one another, with the attack-centric taxonomy being used to estimate the severity of an attack, and the defense-centric taxonomy being used to determine the suitability of defenses.

7 Conclusion

This work has demonstrated that the classes of an attack-centric taxonomy are not equivalent to those of a defense-centric taxonomy. While a defense-centric taxonomy can be used successfully to predict whether or not a set of defenses is capable of detecting a particular attack based on its classification in a taxonomy, an attack-centric taxonomy cannot be used in this way. This is more alarming than it is surprising, because defenders presently have no alternatives to various extant attack-centric taxonomies, many of which were noted in Section 1. This work has produced the first known, validated defense-centric taxonomy[5]. It is hoped that others will follow.

8 Acknowledgements

The authors thank Michael Drew for his contributions to the attack programs and validations. Three anonymous referees provided helpful reviews, one of which was quite extraordinary in its thoroughness, which we deeply appreciated. We thank Dr. Robert Stroud for thoughtful comments over the course of several conversations.

[5]Section 3 acknowledges Kumar's contribution, as well as its limitations, in terms of defense-centricity.

References

[1] G. Alvarez and S. Petrovic. A new taxonomy of web attacks suitable for efficient encoding. *Computers and Security*, 22(5):435–449, July 2003.

[2] T. Aslam. A taxonomy of security faults in the Unix operating system. Master's thesis, Purdue University, 1995.

[3] M. Bishop. A taxonomy of Unix and network security vulnerabilities. Technical report, Department of Computer Science, University of California at Davis, May 1995.

[4] H. Debar, M. Dacier, and A. Wespi. A revised taxonomy for intrusion-detection systems. *Annals of Telecommunications*, 55(7–6):361–378, July–August 2000.

[5] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In *IEEE Symposium on Security and Privacy*, pages 120–128, Los Alamitos, CA, 1996. IEEE Computer Society Press. 6–8 May, Oakland, California.

[6] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *IEEE Symposium on Security and Privacy*, pages 202–212, Los Alamitos, California, 1994. IEEE Computer Society Press. 16–18 May, Oakland, California.

[7] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[8] J. D. Howard. *An analysis of security incidents on the Internet, 1989-1995*. PhD thesis, Carnegie Mellon University, Department of Engineering and Public Policy, April 1997.

[9] J. D. Howard and T. A. Longstaff. A common language for computer security incidents. Technical Report SAND98-8667, Sandia National Laboratories, Albuquerque, New Mexico and Livermore, California, October 1998.

[10] I. V. Krsul. *Software Vulnerability Analysis*. PhD thesis, Comp. Sci. Dept., Purdue University, May 1998.

[11] S. Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue University, August 1995.

[12] C. E. Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi. A taxonomy of computer program security flaws, with examples. *ACM Computing Surveys*, 26(3):211–254, September 1994.

[13] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *IEEE Symposium on Security and Privacy*, pages 130–143, Los Alamitos, California, 2001. IEEE Computer Society Press. 14–16 May, Oakland, California.

[14] U. Lindqvist and E. Jonsson. How to systematically classify computer security intrusions. In *IEEE Symposium on Security and Privacy*, pages 154–163, Los Alamitos, CA, 1997. IEEE Computer Society Press. 4–7 May, Oakland, California.

[15] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. Analysis and results of the 1999 DARPA off-line intrusion detection evaluation. In *Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, pages 162–182, 2–4 October 2000, Toulouse, France, 2000. Springer-Verlag, Heidelberg, Germany.

[16] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. J. McClung, D. J. Webber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *First International Workshop on Recent Advances in Intrusion Detection*, 14–16 September 1998, Louvain-la-Neuve, Belgium, 2000.

[17] D. L. Lough. *A Taxonomy of Computer Attacks with Applications to Wireless Networks*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, April 2001.

[18] R. A. Maxion and K. M. C. Tan. Anomaly detection in embedded systems. *IEEE Transactions on Computers*, 51(2):108–120, February 2002.

[19] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson. A methodology for testing intrusion detection systems. *IEEE Transactions on Software Engineering*, 22(10):719–729, October 1996.

[20] M. J. Ranum. A taxonomy of Internet attacks. Slide Presentation, available on the Internet at `http://pubweb.nfr.net/~mjr/pubs/pdf/internet-attacks.pdf`, 1997.

[21] SecurityFocus vulnerability archive. `http://www.securityfocus.com/bid`, May 2003.

[22] G. G. Simpson. *Principles of Animal Taxonomy*. Columbia University Press, New York, 1961, Fourth printing 1969.

[23] P. H. A. Sneath and R. A. Sokal. *Numerical Taxonomy*. W. H. Freeman and Company, San Francisco, 1973.

[24] A. Somayaji and G. Hunsicker. IMMSEC kernel-level system call tracing for Linux 2.2, version 991117. Obtained through private communication. Previous version available on the Internet: `http://www.cs.unm.edu/~immsec/software`, March 2002.

[25] Strace(1) general command manual. Included in the `strace` version 4.2 software package, distributed on the RedHat 6.2 Linux installation CD (disc 1), 1999.

[26] K. M. C. Tan, K. S. Killourhy, and R. A. Maxion. Undermining an anomaly-based intrusion detection system using common exploits. In A. Wespi, G. Vigna, and L. Deri, editors, *Fifth International Symposium on Recent Advances in Intrusion Detection (RAID-2002)*, pages 54–73, 16–18 October 2002, Zurich, Switzerland, 2002. Lecture Notes in Computer Science #2516, Springer-Verlag, Berlin, 2002.

[27] K. M. C. Tan and R. A. Maxion. "Why 6?" Defining the operational limits of stide, an anomaly-based intrusion detector. In *IEEE Symposium on Security and Privacy*, pages 188–201, Los Alamitos, CA, 2002. IEEE Computer Society Press. 12–15 May, Berkeley, California.

[28] K. M. C. Tan and R. A. Maxion. Determining the operational limits of an anomaly-based intrusion detector. *IEEE Journal on Selected Areas in Communications*, 21(1):96–110, January 2003.

[29] K. M. C. Tan, J. McHugh, and K. Killourhy. Hiding intrusions: From the abnormal to the normal and beyond. In *Information Hiding: 5th International Workshop, IH 2002*, pages 1–17, Heidelberg, Germany, 2003. Springer-Verlag. 7-9 October 2002, Noordwijkerhout, The Netherlands.

[30] D. J. Weber. A taxonomy of computer intrusions. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1998.