Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits

Kymie M.C. Tan, Kevin S. Killourhy, and Roy A. Maxion

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

Abstract. Over the past decade many anomaly-detection techniques have been proposed and/or deployed to provide early warnings of cyber-attacks, particularly of those attacks involving masqueraders and novel methods. To date, however, there appears to be no study which has identified a systematic method that could be used by an attacker to undermine an anomaly-based intrusion detection system. This paper shows how an adversary can craft an offensive mechanism that renders an anomaly-based intrusion detector blind to the presence of on-going, common attacks. It presents a method that identifies the weaknesses of an anomaly-based intrusion detector, and shows how an attacker can manipulate common attacks to exploit those weaknesses. The paper explores the implications of this threat, and suggests possible improvements for existing and future anomaly-based intrusion detection systems.

1 Introduction

In recent years, a vast arsenal of tools and techniques has been accumulated to address the problem of ensuring the availability, integrity and confidentiality of electronic information systems. Such arsenals, however, are frequently accompanied by equally vast "shadow" arsenals of tools and techniques aimed specifically at subverting the schemes that were designed to provide system security. Although a shadow arsenal can be viewed negatively as a formidable threat to the security of computer systems, it can also be viewed positively as a source of knowledge for identifying the weaknesses of current security tools and techniques in order to facilitate their improvement.

A small part of the security arsenal, and the focus of this work, is the anomaly-based intrusion-detection system. Anomaly-based intrusion-detection systems have sought to protect electronic information systems from intrusions or attacks by attempting to detect deviations from the normal behavior of the monitored system. The underlying assumption is that such deviations may indicate that an intrusion or attack has occurred (or may still be occurring) on the system. Anomaly detection – detecting deviations from normal – is one of two fundamental approaches used in systems that seek to automate the detection of attacks or intrusions; the other approach is signature-based detection. Anomaly

A. Wespi, G. Vigna, and L. Deri (Eds.): RAID 2002, LNCS 2516, pp. 54–73, 2002.

detection is typically credited with a greater potential for addressing security problems such as the detection of attempts to exploit new or unforeseen vulnerabilities (novel attacks), and the detection of abuse-of-privilege attacks, e.g., masquerading and insider misuse [1].

The promise of the anomaly-detection approach and its incorporation into a number of current automated intrusion-detection strategies (e.g., AT&T's ComputerWatch, SRI's Emerald, SecureNet, etc. [1]) underscores the importance of studying how attackers may fashion counter-responses aimed at undermining the effectiveness of anomaly-based intrusion-detection systems. Such studies are important for two reasons:

- to understand how to strengthen the anomaly-based intrusion-detection system by identifying its weaknesses; and
- to provide the necessary knowledge for guiding the design and implementation of a new generation of anomaly-based intrusion detectors that are not vulnerable to the weaknesses of their forebears.

This paper lays out a method for undermining a well-known anomaly-based intrusion-detection system called stide [2], by first identifying the weaknesses of its anomaly-detection algorithm, and then by showing how an attacker can manipulate common attacks to exploit those weaknesses, effectively hiding the presence of those attacks from the detector's purview. Stide was chosen primarily because it is freely available to other researchers via the Internet. Its accessibility not only encourages independent verification and replication of the work performed here, but it also builds on, and contributes to, a large body of previously published work that uses the stide detection mechanism.

To undermine an anomaly-based intrusion detector, an attacker needs to know the three elements described in Table 1. These elements set the framework for the paper.

Table 1. Elements of methodology for undermining.

1.	Detection coverage (specifically, blind spots) of an anomaly detector.
2.	Where and how an attack manifests in sensor data.
3.	How to shift the manifestation from a covered spot to a blind one.

2 Approaches to Undermining Anomaly Detectors

There are two approaches that would most obviously cause an anomaly detector to miss detecting the anomalous manifestation of an attack. The first of these two items describes the approach commonly found in the literature; the second describes the approach adopted by this study.

– modify the normal to look like the attack, i.e., incorporate the attack manifestations into the model of normal behavior; or
– modify the attack to make it appear as normal behavior.

In the intrusion detection literature, the most cited way to undermine an anomaly-based intrusion detection system is to incorporate undesired, intrusive behavior into the training data, thereby falsely representing "normal" behavior [1,9,10]. By including intrusive behavior explicitly into the training data, the anomaly detector is forced to incorporate the intrusive behavior into its internal model of normal and consequently lose the ability to flag future instances of that intrusive behavior as anomalous. Note that the anomaly detector is viewed as that component of an anomaly-based intrusion detection system solely responsible for detecting deviations from normal; it performs no diagnostic activities.

For example, one way to incorporate intrusive behavior into an anomaly detector's model of normal behavior is to exploit the fact that behavior can change over time. Changing behavior requires the anomaly detector to undergo periodic on-line retraining. Should the information system undergo attacks during the retraining process, then the anomaly detector could inadvertently incorporate undesired attack behavior into its model of normal behavior [1]. The failure of an anomaly-based intrusion detector to detect intrusions or attacks can typically be attributed to contaminated training data, or to updating schemes that incorporate new normal behavior too quickly.

Undermining an anomaly-based intrusion detection system by simply incorporating intrusive behavior into its training data is too imprecise and abstract a method as to be practically useful to an attacker. Identifying and accessing the segment, feature or attribute of the data that will be used to train an anomaly detector, and then surreptitiously and slowly introducing the intrusive behavior into the training dataset, may require time, patience and system privileges that may not be available to an attacker. Moreover, such a scheme does not provide the attacker with any guarantees as to whether or not the act of subversion has been, or will be, successful. The incorporation of intrusive behavior into the training data is no guarantee that the anomaly detector will be completely blind to the attack when the attack is actually deployed. The attacker has no knowledge of how the anomaly detector perceives the attack, i.e., how the attack truly manifests in the data, and no knowledge concerning the conditions that may impede or boost the anomaly detector's ability to detect the manifestation of the attack. For example, even if intrusive behavior were to be incorporated into an anomaly detector's model of normal, it is possible that when the attack is actually deployed, it will interact with other conditions in the data environment (conditions that may not have been present during the training phase), causing anomalous manifestations that *are* detectable by the anomaly detector. It is also possible for the anomalous manifestation of an attack to be detectable *only* when the detector uses particular parameter values. These points illustrate why it is necessary to determine precisely what kinds of anomalous events an anomaly detector may or may not be able to detect, as well as the conditions that enable it to do so.

These issues are addressed by determining the coverage of the anomaly detector in terms of anomalies (not in terms of attacks); this forms the basis of the approach. Only by knowing the kinds of anomalies that are or are not detectable by a given anomaly detector is it possible to modify attacks to manifest in ways that are not considered abnormal by a given anomaly detector. The coverage of an anomaly detector serves as a guide for an attacker to know precisely *how* to modify an attack so that it becomes invisible to the detector.

3 Detection Coverage of an Anomaly Detector

Current evaluation techniques attempt to establish the detection coverage of an anomaly-based intrusion detection system with respect to its ability to detect attacks [21,4,5] , but without establishing whether or not the anomalies detected by the system are attributable to the attack. Typically, claims that an anomaly-based intrusion detector is able to detect an attack are based on *assumptions* that the attack must have manifested in a given stream of data, that the manifestation was anomalous, and that the anomaly detector was able to detect that specific kind of anomaly.

The anomaly-based evaluation technique described in this section establishes the detection coverage of stide [2,21] with respect to the types of anomalous manifestations that the detector is able to detect. The underlying assumption of this evaluation strategy is that no anomaly detection algorithm is perfect. Before it can be determined whether an anomaly-based intrusion detector is capable of detecting an attack, it must first be ascertained that the detector is able to detect the anomalous manifestation of the attack.

This section shows how detection coverage (in terms of a coverage map) can be established for stide. For the sake of completeness, and to facilitate a better understanding of the anomaly-based evaluation strategy, the stide anomaly-detection algorithm is described, followed by a description of the anomaly-based evaluation strategy used to establish stide's detection coverage. A description and explanation of the results of the anomaly-based evaluation is given.

3.1 Brief Description of the Stide Anomaly Detector

Stide operates on fixed-length sequences of categorical data. It acquires a model of normal behavior by sliding a detector window of size DW over the training data, storing each DW-sized sequence in a "normal database" of sequences of size DW. The degree of similarity between test data and the model of normal behavior is based on observing how many DW-sized sequences from the test data are identical matches to any sequences from the normal database. The number of mismatches between sequences from the test data and the normal database is noted. The anomaly signal, which is the detector's response to the test data, involves a user-defined parameter known as the "locality frame" which determines the size of a temporally local region over which the number of mismatches is summed up. The number of mismatches occurring within a locality frame is

referred to as the locality frame count, and is used to determine the extent to which the test data are anomalous. A detailed description of stide and its origins, can be found in [2,21].

3.2 Evaluation Strategy for Anomaly Detectors

It is not difficult to see that stide will only detect "unusual" or foreign sequences – sequences that do not exist in the normal database. Its similarity metric establishes whether or not a particular sequence exists in a normal database of sequences of the same size. Such a scheme means that any sequence that is foreign to the normal database would immediately be marked as an anomaly. However, this observation alone is not sufficient to explain the anomaly detector's performance in the real world. There are two other significant issues that must be considered before the performance of the anomaly detector can be understood fully. Specifically:

- how foreign sequences actually manifest in categorical data; and
- how the interaction between the foreign sequences and the anomaly detection algorithm affects the overall performance of the anomaly detector.

In order to obtain a clear perspective of these two issues, a framework was established in [12,19] that focused on the architecture and characteristics of anomalous sequences, e.g., foreign sequences. The framework defined the anomalous sequences that a sliding-window anomaly detector like stide would likely encounter, and provided a means to describe the structure of those anomalous sequences in terms of how they may be composed from other kinds of subsequences. The framework also provided a means to describe the interaction between the anomalous sequences and the sliding window of anomaly-detection algorithms like stide. Because the framework established how each anomalous sequence was constructed and composed, it was possible to evaluate the detection efficacy of anomaly detectors like stide on synthetic data with respect to examples of clearly defined anomalous sequences. The results of the evaluation showed the detection capabilities of stide with respect to the various foreign sequences that may manifest in categorical data, and how the interaction between the foreign sequences in categorical data and the anomaly-detection algorithm affected the overall performance of the anomaly detector.

3.3 Stide's Performance Results

The most significant result provided by the anomaly-based evaluation of stide was that there were conditions that caused the detector to be completely blind to a particular kind of foreign sequence that was found to exist (in abundance) in real-world data [19]: a minimal foreign sequence. A minimal foreign sequence is foreign sequence whose proper subsequences all exist in the normal data. Put simply, a minimal foreign sequence is a foreign sequence that contains within it no smaller foreign sequences.

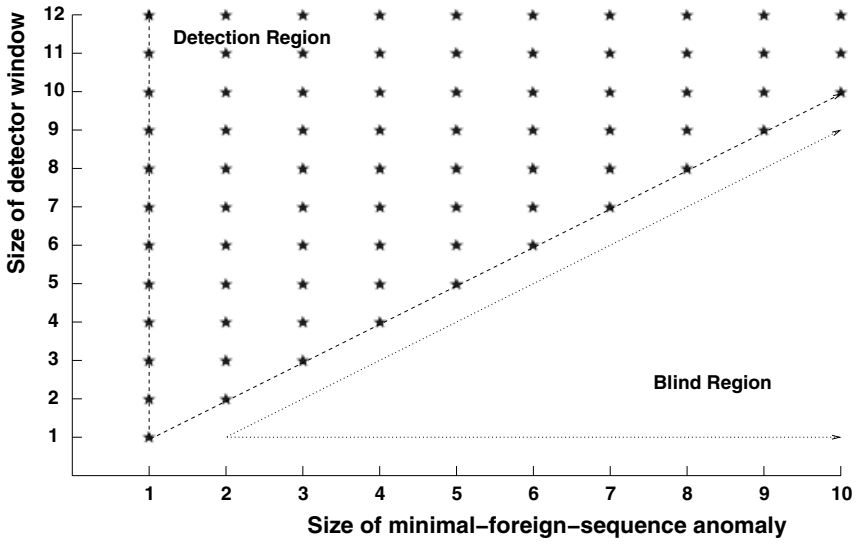

Fig. 1. The detector coverage (detection map) for stide; A comparison of the size of the detector window (rows) with the ability to detect different sizes of minimal foreign sequence (columns). A star indicates detection.

For stide to detect a minimal foreign sequence, it is imperative that the size of the detector window is set to be equal to or larger than the size of the minimal foreign sequence. The consequence of this observation can be seen in Figure 1 which shows stide's detection coverage with respect to the minimal foreign sequence. This coverage map for stide, although previously presented in [19], is shown again here as an aid to the reader's intuition for the coverage map's essential role in the subversion scheme.

The graph in the figure plots the size of the minimal foreign sequence on the x-axis and the size of the detector window on the y-axis. Each star marks the size of the detector window that successfully detected a minimal foreign sequence whose corresponding size is marked on the x-axis. The term *detect* for stide means that the minimal foreign sequence must have caused as at least one sequence mismatch. The diagonal line shows the relationship between the detector window size and the size of the minimal foreign sequence, a relationship that can be described by the function, $y = x$. The figure also shows a region of blindness in the detection capabilities of stide with respect to the minimal foreign sequence. This means that it is possible for a foreign sequence to exist in the data in such a way as to be completely invisible to stide. This weakness will presently be shown to be exploitable by an attacker.

4 Deploying Exploits and Sensors

At this point of the study, the first step in undermining an anomaly detector (see Table 1) has been completed; the detection coverage for stide has been established, and it was observed that the anomaly detector exhibited occasions of detection blindness with respect to the detection of minimal foreign sequences.

The following is a summary of the procedure that was performed in order to address the remaining two items in the method for subverting an anomaly detector listed in Table 1. The remaining two items are where and how an attack manifests in data, and how the manifestation of exploits can be modified to hide the presence of those exploits in the regions of blindness identified by the detection coverage for stide.

1. Install the sensor that provides the anomaly detector with the relevant type of data. In the present work, the sensor is the IMMSEC kernel patch for the Linux 2.2 kernel [18]. The kernel patch records to a file the system calls made by a pre-determined set of processes.
2. Download the `passwd` and `traceroute` exploits and determine the corresponding system programs that these exploits misuse.
3. Execute the system program under normal conditions to obtain a record of normal usage, to obtain normal data. An account of what is considered normal conditions and normal usage of the system programs that correspond to both exploits is described in section 5.2.
4. Deploy the exploits against the host system to obtain the data recording the occurrence of the attacks.
5. Identify the precise manifestation of the attacks in the sensor data.
6. Using the normal data obtained from step 3, and the intrusive data obtained from step 4, deploy stide to determine if the anomaly detector is capable of detecting the unmodified exploits that were simply downloaded, compiled and executed. This is performed in order to establish the effectiveness of the subversion process. If stide is able to detect the unmodified exploits but not the modified exploits, then the subversion procedure has been effective.
7. Using information concerning the kind of events that stide is blind to, modify the attacks and show that it is possible to make attacks that were once detectable by stide, undetectable for detector window sizes one through six.

5 Where and How an Attack Manifests in the Data

This section addresses the second item in the list of requirements for undermining an anomaly detector – establishing where and how an attack manifests in sensor data (see Table 1) – by selecting two common exploits, deploying them, and establishing how and where they manifest in the sensor data. Steps 2 to 5 of the method laid out above are covered by this section.

5.1 Description and Rationale for the Exploits Chosen

The attacks selected for this study are examples of those that stide is designed to detect, i.e., attacks that exploit privileged UNIX system programs. UNIX system programs typically run with elevated privileges in order to perform tasks that require the authority of the system administrator – privileges that ordinary users are not typically afforded. The authors of stide have predominantly applied the detector towards the detection of abnormal behavior in such privileged system programs, because exploiting vulnerabilities to misuse privileged system programs can potentially bestow those extra privileges on an attacker [6].

Two attacks were chosen arbitrarily out of several that fulfill the requirement of exploiting UNIX system programs. The two attacks chosen will be referred to as the `passwd` and `traceroute` exploits. The `passwd` exploit takes advantage of a race condition between the Linux kernel and the `passwd` system program; the `traceroute` exploit takes advantage of a vulnerability in the `traceroute` system program.

`passwd` is a system program used to change a user's password [3]. The program allows an ordinary user to provide his or her current password, along with a new password. It then updates a system-wide database of the user's information so that the database contains the new password. The system-wide database is commonly referred to as the `/etc/passwd` or the `/etc/shadow` file. A user does not normally have permission to edit this file, so `passwd` must run with root privileges in order to modify that file. The exploit that misuses the `passwd` system program does so by employing a race condition that is present in the Linux kernel to debug privileged processes.

Normally, the `passwd` system process performs only a restricted set of actions that consists of editing the `/etc/passwd` and/or the `/etc/shadow` file. However, the `passwd` system process can be made to do more, because of a race condition in the Linux kernel which allows an unprivileged process to debug a system process. Using an unprivileged process, an attacker can alter or "debug" the `passwd` system process and force it to execute a command shell, granting the attacker elevated privileges.[1] Details of race conditions in the Linux kernel are given in [13] and [17]. The `passwd` exploit was obtained from [15].

The `traceroute` network diagnostic utility is a system program that is usually employed by normal users to gather information about the availability and latency of the network between two hosts [7]. To accomplish this task, the `traceroute` system program must have unrestricted access to the network interface, a resource provided only to privileged system programs. However, a logic error in the `traceroute` system program allows an attacker to corrupt the memory of the process by specifying multiple network gateways on the command line [16]. The `traceroute` exploit uses this memory corruption to redirect the process to instructions that execute a command shell with the elevated privileges of the `traceroute` system program [8]. More detail on this memory corruption vulnerability is provided in [16]. The `traceroute` exploit was obtained from [8].

[1] In industry parlance, the instructions injected by the exploit are termed "shellcode", and the shell in which an intruder gains elevated privileges is a "rootshell."

Several key features make certain attacks or exploits likely candidates for subverting sequence-based anomaly detectors such as stide. The subversion technique presented in this paper is more likely to be effective when:

- the vulnerability exploited involves a system program that runs with elevated (root) privileges;
- the vulnerability allows an attacker to take control of the execution of the system program, giving the attacker the ability to choose the system kernel calls or instructions that are issued by the system program;
- the attack does not cause the system program to behave anomalously (e.g. produce an error message in response to an invalid input supplied by the attacker) before the attack/attacker can take control of the execution of the system program;
- the system kernel calls occurring after the "point of seizure", i.e., the point in the data stream at which the attacker first takes control of the system program, include any or all of `execve`, or `open`/`write`, or `chmod`, or `chown`, or any other system kernel call that the attacker can use to effect the attack.

5.2 Choice of Normal Data

It is unreasonable to expect an attacker to be able to identify and access the precise segment, feature or attribute of the data that can be used to train an anomaly detector. The time, patience and system privileges required to do so may simply not be available to an attacker. However, since training data is vital to the function of an anomaly detector, the attacker has to construct an approximation of the training data that may have been used by the anomaly detector if he or she desires to exploit a blind spot of the detector.

For anomaly detectors like stide, i.e., anomaly detectors that monitor system programs, training data can be approximated more easily, because system programs typically behave in very set and regimented ways. For example, the `passwd` and `traceroute` system programs are limited in the number of ways that they can be used, and as a result it is possible to make reasonable assumptions about how these programs would be regularly invoked.

These assumptions may be aided by the wealth of easily accessible documentation that typically accompanies each system program, as well as by any general knowledge or experience already acquired by the attacker. It is important to note, however, that the success of this method for undermining stide relies on the attacker's being able to approximate normal usage of the system program.

To be successful at undermining stide, the attacker does not need to obtain every possible example of a system program's normal behavior. If the anomalous manifestation of an exploit can already be crafted by an extremely reduced subset of normal behavior, then it can only be expected that more examples of normal behavior contribute to an increased number of ways with which to construct the anomalous manifestation of the exploit.

For the `passwd` system program, normal data was obtained by executing the `passwd` system program with no arguments, and then by following the instructions displayed by the program to input the user's current password once, and

then their new password twice. In other words `passwd` was invoked to expire an old password and install a new one.

For the `traceroute` system program, normal data was obtained by executing `traceroute` to acquire diagnostic information regarding the network connectivity between the local host and the Internet site `nis.nsf.net`. This site was chosen because it is the simplest example of using `traceroute`, based on the documentation provided with the program itself [7].

5.3 Establishing Attack Manifestations in Sensor Data

Two issues are addressed in this subsection. The first is whether the attacks embodied by the execution of the chosen exploits actually manifested in the sensor data, and the second is whether the manifestation is an anomalous event detectable by stide. Simply because an attack can be shown to manifest in the sensor data does not necessarily mean that the manifestation is automatically anomalous. It is necessary to establish that the manifestation of the exploits are initially detectable by stide in order to show that any modifications to the same exploits effectively render them undetectable by the same detector.

Before proceeding any further it is necessary to define what is meant by the term *manifestation* within the scope of this study. The manifestation of an attack is defined to be that sequence of system calls issued by the exploited/privileged system program, and due to the presence and activity of the exploit. The remainder of this section describes how the manifestations of the two exploits, `passwd` and `traceroute`, were obtained.

`passwd`. The `passwd` exploit was downloaded from [15]; then it was compiled and deployed. There were no parameters that needed to be set in order to execute the exploit. The successful execution of the exploit was confirmed by checking that elevated privileges were indeed conferred.

The manifestation of the `passwd` exploit was determined manually. An inspection of the source code for both the `passwd` exploit and that portion of the Linux kernel responsible for the race condition vulnerability identified the precise system calls that were attributable to the attack. The sequence of system calls that comprise the manifestation of the attack embodied by the `passwd` exploit is `setuid`, `setgid`, `execve`. Stide was then deployed, using the normal data described above as training data, plus the test data comprised of the data collected while the `passwd` exploit was executed. Stide was run with detector window sizes ranging from 1 to 15. It was found that the attack was detectable at all detector window sizes. More precisely, the attack was detectable by stide because `setuid`, `setgid`, and `execve` were all foreign symbols. From the detection map of stide in Figure 1, it can be seen that stide is capable of detecting size-1 foreign symbols at any detector window size.

traceroute. The `traceroute` exploit was downloaded from [8]; then it was compiled and deployed. The `traceroute` exploit expects values for two arguments. The first argument identifies the local platform, and the second argument is a hexadecimal number that represents the address of a specific function in memory. This address is overwritten to point to an attacker-specified function. The successful execution of the exploit was confirmed by checking that elevated privileges were indeed conferred.

The manifestation of the `traceroute` exploit was determined manually. An inspection of the source code for the `traceroute` exploit as well as for the `traceroute` system program, identified the precise system calls that were attributable to the attack. The sequence of system calls that comprise the manifestation of the attack embodied by the `traceroute` exploit is: `brk`, `brk`, `brk`, `setuid`, `setgid`, `execve`. Mirroring the deployment strategy for `passwd`, stide was trained on the previously collected `traceroute` normal data, and run with detector-window sizes 1-15. The attack was shown to be detectable at all window sizes, because `setuid`, `setgid`, and `execve` were all foreign symbols.

6 Manipulating the Manifestation; Modifying Exploits

Three vital items of knowledge have been established up to this point: the characteristics of the minimal-foreign-sequence event that stide is sometimes unable to detect; the conditions ensuring that stide does not detect such an event (the detector-window size must be smaller than the size of the minimal foreign sequence); and the fact that stide is completely capable of detecting the two chosen exploits when they were simply executed on the host system without any modifications. This means that the anomaly detector is completely effective at detecting these exploits should an attacker decide to deploy them.

How can these exploits be modified so that the anomaly detector does not sound the alarm when the modified exploits are deployed? How can an attacker provide his or her attack(s) with every opportunity to complete successfully and stealthily? This section shows how both exploits, guided by the detection map established for stide, can be modified to produce manifestations (or signatures) in the sensor data that are not visible to the detector.

6.1 Modifying `passwd` and `traceroute`

In aiming to replace the detectable anomalous manifestations of the exploits with manifestations that are undetectable by stide, there are two points that must be considered. Recall that each exploit embodies some goal to be attained by the attacker, e.g., elevation of privileges.

First, because the method for achieving the goal that is embodied by each exploit, `passwd` and `traceroute`, produces an anomalous event detectable by stide, namely a foreign symbol, another method for achieving the same goal must be found to replace it. Note that the goal of both exploits is the typical one of securing an interactive shell with elevated privileges. Interestingly, the

Table 2. The system calls that implement each of three methods that attempt to achieve the same goal of securing an interactive root account accessible to the attacker.

	Description of method	System calls that implement method
1	Changing the access rights to the `/etc/passwd` file in order to give the attacker permission to modify the file (write permission)	`chmod`, `exit`
2	Changing the *ownership* of the `/etc/passwd` file to the attacker	`chown`, `exit`
3	Opening the `/etc/passwd` file to append a new user with root privileges	`open`, `write`, `close`, `exit`

new means of achieving the same goal involves changing the value of only one variable in both exploit programs.

Second, the new method of achieving the same goal must not produce any manifestation that is detectable by stide. Although this could mean that both exploits are modified so that their manifestations appear normal, i.e., their manifestations match sequences that already exist in the normal data, it is typically more difficult to do this than to cause the exploits to manifest as foreign sequences. The difficulty lies in the fact that the kinds of normal sequences that can be used to effect an attack may be small. This makes it more likely that an attacker may require sequences that lie outside the normal vocabulary, i.e., foreign sequences.

6.2 New Means to Achieve Original Goals in Exploit Programs

In Section 5.3 it was shown that the execution of the `passwd` and `traceroute` exploits were detectable by stide because both exploits manifested anomalously as the foreign symbols `setuid`, `setgid`, and `execve`. Any attack that introduces a foreign symbol into the sensor data that is monitored by stide, will be detected. This is because foreign symbol manifestations lie in the visible region of stide's detection map. In order for the `traceroute` or `passwd` exploits to become undetectable by stide, they must not produce the system calls `setuid`, `setgid`, and `execve`. Instead an alternate method causing the exploits to manifest as minimal foreign sequences is required. Only system calls that are already present in the normal data can be the manifestation of the exploits.

For the `passwd` exploit, another method for achieving the same goal of securing an interactive shell with elevated privileges that does not involve the foreign symbols `setuid`, `setgid`, and `execve` would be to cause the exploit program to give the attacker permission to *modify* the `/etc/passwd` file. With such access, the attacker can then edit the accounts and give him or herself administrative privileges, to be activated upon his or her next login. The system calls required

to implement this method are `chmod` and `exit`. These two calls are found in the normal data for the `passwd` system program.

There at least two other methods that will achieve the same goal. A second method would be to give the attacker *ownership* of the `/etc/passwd` file, and a third method would be to make the affected system program directly edit the `/etc/passwd` file to add a new administrative (root) account that is accessible to the attacker. The system calls that would implement all three methods respectively are listed in Table 2.

For the `traceroute` exploit, the other method for achieving the same goal of securing an interactive shell with elevated privileges that does not involve the foreign symbols `setuid`, `setgid`, and `execve`, is to make the affected system program directly edit the `/etc/passwd` file to add a new administrative (root) account that is accessible to the attacker. The system calls required to implement this method are `open`, `write`, `close`, and `exit`. All these system calls can be found in the normal data for the `traceroute` system program.

6.3 Making the Exploits Manifest as Minimal Foreign Sequences

In the previous subsection, the two exploits were made to manifest as system calls that can be found in the normal data for the corresponding `passwd` and `traceroute` system programs. This is still insufficient to hide the manifestations of the exploits from stide, because even though system calls that already exist in the normal data were used to construct the new manifestation of each exploits, the order of the system calls with respect to each other can still be foreign to the order of system calls that typically occur in the normal data. For example, even if `chmod` and `exit` both appear in the `passwd` normal data, both calls never appear sequentially. This means that the sequence `chmod`, `exit`, is a foreign sequence of size 2, foreign to the normal data. More precisely, this is a minimal foreign sequence of size 2, because the sequence does not contain within it any smaller foreign sequences or foreign symbols.

As a consequence, stide with a detector window of size 2 or larger would be fully capable of detecting such a manifestation. In order to make the manifestation invisible to stide, it is necessary to increase the size of the minimal foreign sequence. Increasing the size raises the chances of falling into stide's blind spot. Referring to Figure 1, it can be seen that the larger the size of the minimal foreign sequence, the larger the size of the blind spot.

To increase the size of the minimal foreign sequence, the short minimal foreign sequences that are the manifestations of both exploits (`chmod`, `exit` for the `passwd` exploit, and `open`, `write`, `close`, and `exit` for the `traceroute` exploit) must be padded with system calls from the normal data that would result in larger minimal foreign sequences with common subsequences. For example, for `passwd` the short minimal foreign sequence that is the manifestation of the new method described in the previous section is `chmod`, `exit`. This is a minimal foreign sequence of size 2. To increase this minimal foreign sequence it can be seen that in the normal data for `passwd`, the system call `chmod` is followed by

the sequence `utime`, `close`, `munmap`, and elsewhere in the normal data, `munmap` is followed by `exit`. These two sequences

1. `chmod`, `utime`, `close`, `munmap`
2. `munmap`, `exit`
   can be concatenated to create a third sequence
3. `chmod`, `utime`, `close`, `munmap`, `exit`.

A method of attack can be developed which manifests as this concatenated sequence. This method is functionally equivalent to the method developed in the previous subsection; it gives the attacker permission to modify `/etc/passwd` with the `chmod` system call and exits with the `exit` system call. The three system calls `utime`, `close`, and `munmap` are made in such a way that they do not alter the state of the system.

If stide employed a detector window of size 2, and the detector window slid over the manifestation of the exploit that is the sequence `chmod`, `utime`, `close`, `munmap`, `exit`, no anomalies would result; no alarms would be generated because the manifestation no longer contains any foreign sequences of size 2. However, if stide employed a detector window of size 3, a single anomaly would be detected, namely the minimal foreign sequence of size 3, `close`, `munmap`, `exit`, which would result in an alarm.

The simple example given above describes the general process for creating the larger minimal foreign sequences required to fool stide. By performing an automated search of the normal data it is possible to find all sequences that can be used by an attacker as padding for the manifestation of a particular exploit. The general process for creating larger minimal foreign sequences was automated and used to modify both the `passwd` and `traceroute` exploits.

It is important to note that because stide only analyzes system calls and not their arguments, it is possible to introduce system calls to increase the size of minimal foreign sequences without affecting the state of the system. Executing system calls introduced by the attacker that are aimed at exploiting stide's blind spot need not cause any unintended side-effects on the system because the arguments for each system call is ignored. It is therefore possible to introduce system calls that do nothing, such as reading and writing to an empty file descriptor, or opening a file that cannot exist. This point argues for using more diverse data streams in order to provide more effective intrusion detection. Analyzing only the system call stream may be a vulnerability in anomaly detectors.

7 Evaluating the Effectiveness of Exploit Modifications

A small experiment is performed to show that the modified exploits were indeed capable of fooling stide. As shown in the previous section, a single deployment of a modified exploit is accompanied by a parameter that determines the size of the minimal foreign sequence that will be the manifestation of the exploit. Each exploit was deployed with parameter values that ranged between 2 and 7. A minimum value of 2 was chosen, because it is the smallest size possible for a

minimal foreign sequence. The maximum value chosen was 7, because a minimal foreign sequence of size 7 would be invisible to stide employing a detector window of size 6. In the literature, stide is often used with a detector window of size 6 [6, 21]. 6 has been referred to as the "magic" number that has caused stide to begin detecting anomalies in intrusive data [6,11]. Using a detector window of size 6 in this experiment serves to illustrate a case where 6 may not be the best size to use because it will miss detecting exploits that manifest as minimal foreign sequences of size 7 and higher.

Each of the two exploits were deployed 6 times, one for each minimal foreign sequence size from 2 to 7. For each execution of an exploit, stide was deployed with detector window sizes 1 to 15. 1 was chosen as the minimum value simply because it is the smallest detector window size that the detector can be deployed with, and 15 was chosen as the maximum arbitrarily.

7.1 Results

The x-axis for the graph in Figure 2 represents the size of the minimal foreign sequence anomaly, and the y-axis represents the size of the detector window. Each star marks the size of the detector window that successfully detected a minimal foreign sequence whose corresponding size is marked on the x-axis. The term detect for stide means that the manifestation of the exploit must have registered as at least one sequence mismatch. Only the results for `traceroute` are presented. The results for `passwd` are very similar and have been omitted due to space limitations.

The graph in Figure 2 mirrors the detection map for stide, showing that the larger the minimal foreign sequence that is the manifestation of an exploit, the larger the detector window required to detect that exploit. Each circle marks the intersection between the size of the minimal foreign sequence that is the manifestation of the exploit and the size of the detector window used by stide, namely 6. Within each circle the presence of the star indicates that the manifestation of the exploit was detected by stide with a window size of 6.

Each successive circle along the x-axis at $y = 6$ depicts a shift in the manifestation of the exploit in terms of the increasing size of the minimal foreign sequence. These shifts are due to having modified the exploit. The arrows indicate a succession of modifications. For example, without any modification the exploit will naturally manifest as a foreign symbol in the data stream; this is represented by the circle at $x = 1, y = 6$. The first modification of the exploit resulted in a minimal foreign sequence of size 2; this is represented by the circle at $x = 2, y = 6$ pointed to by the arrow from the circle at $x = 1, y = 6$. The second modification yields a size-3 foreign sequence, and so forth. There is no circle at $x = 7$ because it was impossible to modify the exploit to shift its manifestation to a size-7 minimal foreign sequence, given the normal data for `traceroute`.

To summarize, if stide were deployed with a detector window of size 6, then it is possible to modify the `traceroute` exploit incrementally, so that it manifests as successively larger minimal foreign sequences, until a size is reached (size 7) at which the manifestation falls out of stide's visible detection range, and into its

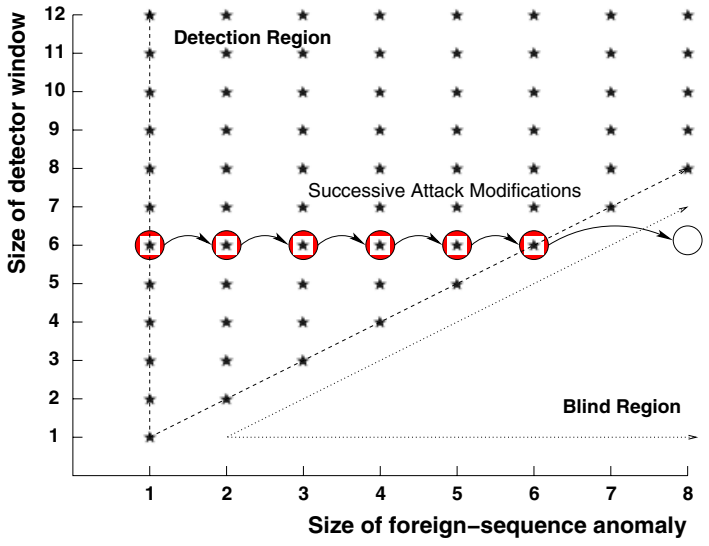

Fig. 2. The manifestation of each version of the `traceroute` exploit plotted on the detector coverage map for stide, assuming that stide has been configured with a detector window size of 6. Each version of the exploit can be detected by fewer and fewer configurations of stide until the last is invisible.

blind spot. This shows that it is possible to exert control over a common exploit so that its manifestation is moved from an anomaly detector's detection region, to its region of complete blindness. Such movement of an exploit's manifestation effectively hides the exploit from the detector's view.

8 Discussion

The results show that it is possible to hide the presence of the `passwd` and `traceroute` common exploits from stide by modifying those exploits so that they manifest only within stide's detection blind spot. Achieving an attack's objectives is not affected by the modifications to the exploit programs; neither is the training data tampered with in order to render an anomaly detector blind to the attacks. Note that, at present, these results can be said to be relevant only to other anomaly-based intrusion detection systems that employ anomaly detectors operating on sequences of categorical data. Although the results make no claims about other families of anomaly detectors that, for example, employ probabilistic concepts, it is possible that the methods described in this study may be applicable to a broader range of anomaly detectors.

The results presented in this paper show that it is also possible to control the manifestation of an attack so that the manifestation moves from an area of

detection blindness to an area of detection clarity for stide. Figure 2 shows the results of modifying the manifestation of an exploit in controlled amounts until the manifestation falls outside the anomaly detector's detection range.

8.1 Implications for Anomaly Detector Development

By identifying the precise event and conditions that characterize the detection blindness for stide and showing that real-world exploits can be modified to take advantage of such weaknesses, one is forewarned not only that such weaknesses exist, but also that they present a possible and tangible threat to the protected system. It is now possible to mitigate this threat by, for example, combining stide with another detector that could compensate for the problems inherent in the stide detection algorithm. The variable sequence size model explored by Marceau [11] seems to be a promising step toward addressing the weakness in stide. Because detection coverage has been defined in a way that is pertinent to anomaly detectors, i.e., in terms of the kinds of anomalies that can be detected by a given anomaly detector rather than in terms of intrusions, it is also possible to compose anomaly detectors to effect full coverage over the detection space.

It is interesting to note that the ease with which an attacker can introduce sequences into the system call data suggests that sequences of system calls may not be a sufficiently expressive form of data to allow an anomaly detector to more effectively monitor and defend an information system. Increasing the number of different kinds of data analyzed, or changing the kind of data analyzed by an anomaly detector, may make an impact on the effectiveness of the intrusion-detection capabilities of an anomaly detector.

8.2 Implications for Anomaly Detector Evaluation

There are a few benefits of an anomaly-based evaluation method, an evaluation method focused on how well anomaly detectors detect anomalies. First, the results of an anomaly-based evaluation increases the scope of the results. In other words, what was established to be true with respect to the anomaly-detection performance of the anomaly detector on synthetic data will also be true of the anomaly detector on real-world data sets. This cannot be said of current evaluation procedures for anomaly detectors because current evaluation schemes evaluate an anomaly detector in terms of how well it detects attacks. This constrains the scope of the results to the data set used in the evaluation because an attack that manifests as a specific kind of anomaly in one data set may no longer do so in another data set due to changing normal behavior.

Second, the results of an anomaly-based evaluation can only contribute to increasing the accuracy of anomaly detectors performing intrusion detection. Current evaluation methods do not establish the detection capabilities of an anomaly detector with regard to the detection of the anomalous manifestations of attacks. The fact that attacks may manifest as different types of anomalies also means that different types of anomaly detectors may be required to detect them. If anomaly detectors are not evaluated on how well they detect anomalies,

it is difficult to determine which anomaly detector would best suit the task of detecting a given type of attack. The events that anomaly detectors directly detect are anomalies, but typically anomaly detectors are evaluated on how well they detect attacks, the events that anomaly detectors do not detect except by making the assumption that attacks manifests as those anomalies detected by an anomaly detector.

9 Related Work

The concept of modifying an attack so that it successfully accomplishes its goal while eluding detection is not a new one. Ptacek and Newsham [14] highlighted a number of weaknesses that the network intrusion detection community needed to address if they were to defeat a wily attacker using network signature-based intrusion detection systems. Although the work presented in this paper differs in that it focuses on anomaly-based intrusion detection, it strongly reiterates the concern that an unknown weakness in an intrusion detection system creates a "dangerously false sense of security." In the case of this work, it was shown that a weakness in an anomaly detector could realistically be exploited to compromise the system being protected by that detector.

Wagner and Dean [20] introduced a new class of attacks against intrusion detection systems which they called the "mimicry attack". A "mimicry attack" is where an attacker is able to "develop malicious exploit code that mimics the operation of the application, staying within the confines of the model and thereby evading detection ..." Wagner and Dean studied this class of attack theoretically, but until this present paper, no study has shown if it were possible to create and deploy a mimicry attack in the real-world that truly affected the performance of an intrusion-detection system.

The present study confirms that the class of mimicry attacks does pose a serious threat to an anomaly-based intrusion detection system. By modifying common real-world exploits to create examples of this class of attack, this study also shows how mimicry attacks are able to undermine the protection offered by an anomaly-based intrusion detection system.

10 Conclusion

This study has shown how an anomaly-based intrusion detection system can be effectively undermined by modifying common real-world exploits. It presented a method that identified weaknesses in an anomaly-based intrusion detector, and showed how an attacker can effectively modify common exploits to take advantage of those weaknesses in order to craft an offensive mechanism that renders an anomaly-based intrusion detector blind to the on-going presence of those attacks.

The results show that it is possible to hide the presence of the `passwd` and `traceroute` common exploits from stide by modifying those exploits so that they manifest only within stide's detection blind spot. The results also show that it

is possible to control the manifestation of an attack such that the manifestation moves from an area of detection clarity to one of detection blindness for stide.

References

1. Herve Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, April 1999.
2. Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, 6–8 May 1996, Oakland, California, pages 120–128, IEEE Computer Society Press, Los Alamitos, California, 1996.
3. Cristian Gafton. passwd(1). Included in `passwd` version 0.64.1-1 software package, January 1998.
4. Anup K. Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning program behavior profiles for intrusion detection. In *Proceedings of the 1st Workshop on Intrusion Detection and Network Monitoring*, 9–12 April 1999, Santa Clara, California, pages 51–62, The USENIX Association, Berkeley, California, 1999.
5. Anup K. Ghosh, James Wanken, and Frank Charron. Detecting anomalous and unknown intrusions against programs. In *Proceedings of the 14th Annual Computer Security Applications Conference*, 7–11 December 1998, Phoenix, Arizona, pages 259–267, IEEE Computer Society Press, Los Alamitos, 1998.
6. Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.
7. Van Jacobson. Traceroute(8). Included in `traceroute` version 1.4a5 software package, April 1997.
8. Michel "MaXX" Kaempf. Traceroot2: Local root exploit in LBNL traceroute. Internet: `http://packetstormsecurity.org/0011-exploits/traceroot2.c`, March 2002.
9. Sandeep Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue University, West Lafayette, Indiana, August 1995.
10. Teresa Lunt. Automated audit trail analysis and intrusion detection: A survey. In *Proceedings of the 11th National Computer Security Conference*, Baltimore, Maryland, pages 65–73, October 1988.
11. Carla Marceau. Characterizing the behavior of a program using multiple-length N-grams. In *New Security Paradigms Workshop*, 18–22 September 2000, Ballycotton, County Cork, Ireland, pages 101–110, ACM Press, New York, New York, 2001.
12. Roy A. Maxion and Kymie M. C. Tan. Anomaly detection in embedded systems. *IEEE Transactions on Computers*, 51(2):108–120, February 2002.
13. Andrew P. Moore. CERT/CC vulnerability note VU#176888, July 2002. Internet: `http://www.kb.cert.org/vuls/id/176888`.
14. Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Secure Networks, Inc., Calgary, Alberta, Canada, January 1998.
15. Wojciech Purczynski (original author) and "lst" (author of improvements). Epcs2: Exploit for execve/ptrace race condition in Linux kernel up to 2.2.18. Internet: `http://www.securiteam.com/exploits/5NP061P4AW.html`, March 2002.
16. SecurityFocus Vulnerability Archive. LBNL Traceroute Heap Corruption Vulnerability, Bugtraq ID 1739. Internet: `http://online.securityfocus.com/bid/1739`, March 2002.

17. SecurityFocus Vulnerability Archive. Linux PTrace/Setuid Exec Vulnerability, Bugtraq ID 3447. Internet: `http://online.securityfocus.com/bid/3447`, March 2002.
18. Anil Somayaji and Geoffrey 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.
19. Kymie M. C. Tan and Roy A. Maxion. "Why 6?" Defining the operational limits of stide, an anomaly-based intrusion detector. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 12–15 May 2002, Berkeley, California, pages 188–201, IEEE Computer Society Press, Los Alamitos, California, 2002.
20. David Wagner and Drew Dean. Intrusion detection via static analysis. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 14–16 May 2001, Berkeley, California, IEEE Computer Society Press, Los Alamitos, California, 2001.
21. Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, 9–12 May 1999, Oakland, California, pages 133–145, IEEE Computer Society Press, Los Alamitos, California, 1999.