On the Contribution of Preamble to Information Hiding in Mimicry Attacks

H. Güneş Kayacık, A. Nur Zincir-Heywood
Dalhousie University, Faculty of Computer Science,
6050 University Avenue, Halifax, Nova Scotia. B3H 1W5
{kayacik, zincir}@cs.dal.ca

Abstract

In this paper, we aim to determine the significance of different stages of an attack, namely the preamble and the exploit, on an achieved anomaly rate. To this end, we analyze four UNIX applications that have been used by the previous researchers against Stide anomaly detector. Our results show that the effect of the preamble on the anomaly rate is much greater when the size of the preamble component of an attack is greater than the size of the exploit component. Furthermore, we investigate the impact of training set selection and the length of sliding window on detector performance.

Keywords

Information hiding, anomaly detection, benchmarking

1 Introduction

Over the past years, systematic methods have been proposed to analyze host based anomaly detection systems against blind spots and evasion attacks. Previous research established that it is possible to evade anomaly detectors, namely Stide, by altering an attack to make it look like normal behavior. The common assumptions in these works [1, 2, 3] are:

- Detectors are built for critical system applications that have limited behavior;
- While establishing a baseline for the detector, normal behavior can be easily approximated;
- Attackers do not have to obtain all possible cases of normal behavior.

Consequently, these methods mainly focus on crafting new exploits that can completely bypass a given anomaly detector. The common shortcoming of the previous research is that they focused on optimizing a particular component (namely exploit component) of an attack without analyzing the attack as a whole.

In this work, we analyze four UNIX applications that have been used by the previous researchers [1, 2, 3] to determine the significance of different stages of an attack as well as selecting appropriate training data configuration parameters. Therefore, the significance of our work is threefold: First, we analyze the importance of training set selection to detector performance. Normal behavior can vary substantially between various normal use scenarios. It is crucial to employ a training set that will minimize false alarms whilst maximizing the detection rate. Second, we analyze the use of different detector window sizes. The window size parameter determines the length of the sequence stored in normal databases. Therefore, we analyze the impacts of employing longer window sizes as opposed to shorter ones. Third, we identify two components, which exists in all buffer overflow attacks, namely to break in (gaining control of the vulnerable application) and the exploit. Previous [1, 2, 3] work concentrated only on the exploit component. However we believe that anomaly rates for these two stages should be analyzed collectively, since it is impossible to deploy an exploit without first breaking in. Thereafter, in this work, attacks will be considered as comprising from two stages; the break in part of an attack is referred to as the preamble and the second stage as the exploit.

Furthermore, in this paper, we demonstrate that it may not be possible in all cases for an attacker to eliminate all anomalies from the attack. Although critical system services have limited behavioral properties, building normal behavior models based on this assumption causes extensive false alarms due to slight changes in normal behavior or legitimate faults/errors, i.e. making a typo in the command. In practice, anomaly detectors are deployed with an anomaly rate threshold to minimize false alarms. Therefore it is important, from the perspectives of both the attacker and the security professional, to determine a suitable anomaly threshold.

Our objective is to provide a complete analysis of an attack against an anomaly detector. Specific shortcomings in the current blind spot and mimicry attack research for anomaly based security tools and techniques are identified. We believe that before an anomaly detector is developed/built for an application, a similar analysis should be performed to determine the importance of training data scenarios and significance of legitimate errors to detection. Furthermore, in the case of blind spot analysis (vulnerability testing), we believe that researchers should analyze the original vulnerability to determine the contribution from preamble and exploit components.

The remainder of the paper is organized as follows. Section 2 discusses the earlier research on blind spot and mimicry attack analysis. Anomaly detector that we employed in this work is presented in Section 3, whereas the four vulnerable applications employed are detailed in Section 4. Experimental results are reported in Section 5 and conclusions are drawn in Section 6.

2 Related Work

Related detector blind spot testing [1, 2, 3] is mainly focused on the Stide detector [4] (or variants thereof) and employed critical UNIX system applications. Undetectable exploits are developed by locating appropriate sequences of system calls that match the contents of Stide's normal behavior database whilst successfully reaching the behavioral objectives of the original exploit. Typically, a minimalist configuration of the anomaly detector is utilized, under the general observation that it is easier to make a strong detector if the alphabet of permitted instructions is small.

Wagner et al. [1] introduced the "mimicry attack" concept, where all attacks were modified to evade detection. They proposed three methods to avoid detection: (i) modifying system call parameters; (ii) inserting system calls that are irrelevant to the attack being deployed while minimizing the anomaly rate; and finally (iii) generating equivalent attacks by replacing the system calls that can easily be identified by the detector. An example for the last method is substituting an attack that spawns a UNIX shell with an attack that creates a super-user account. Both of these give the attacker super-user privileges. Mimicry attacks were generated for wuftpd service by manually modifying the detectable system call sequences. Normal behavior was generated by "running wuftpd on hundreds of large file downloads over a period of two days"[1].

Tan et al. [2] employed four methods to manually change the behavior of the attack: (i) hiding an attack in the blind spot of the detector; (ii) modifying an attack so that it looks like a normal behavior; (iii) hiding an attack so it looks like a less dangerous attack; and (iv) modifying an attack so that it looks like a different attack. In their experiments, restore, tmpwatch and kernel/traceroute attacks were employed. Normal behavior for Restore is obtained by "monitoring a regular user executing the restore system program to retrieve backup data from a remote backup server" [2]. Normal behavior for tmpwatch is generated by populating a short directory tree with files under */tmp* and executing tmpwatch program to clean files that are more than 5 days old. Normal behavior for kernel attack was not obtained since the vulnerability in the kernel was used to exploit another vulnerability in traceroute.

Stide detects foreign sequences that are not in the normal database. Thus, Tan et al. [3] investigated hiding in the detector's blind spots in more detail by developing variants of a core exploit with the objective of increasing the minimal foreign sequence length. They reported that if the foreign sequence length is greater than the sliding window size of Stide, an attack could evade detection. In their experiments they employed Stide on traceroute and passwd applications. For traceroute, normal behavior is obtained by "executing traceroute to acquire diagnostic information regarding the network connectivity between the localhost and nis.nsf.net"[3]. For the passwd application, normal data was obtained by executing the passwd without any arguments, which expires the old password and installs the new one provided by the user.

3 Stide Anomaly Detector

Anomaly detection systems attempt to build behavioral models of normal use and employ this as the basis for detecting suspicious activities. This way, known and unknown/unseen attacks can be detected as long as the attack behavior deviates sufficiently from the normal behavior. Needless to say, if the attack is sufficiently similar to the normal behavior, it may not be detected. However, user behavior itself is not constant, thus even the normal activities of a user may start raising alarms. In this work, Stide [4] was used as the target anomaly detector, since it has been employed in many of the earlier works [1, 2, 3, 5] and also to the best of our knowledge, it is the only anomaly detector that is available as open source.

Input to the Stide detector takes the form of system call traces of an application for which the detector is trained. Specifically, Stide builds a "normal database" by segmenting the training data (of system call traces) into fixed length sequences [5]. To do so, a sliding window of N is employed over the training dataset and the resulting system call patterns are stored in the "normal database". During testing, the same sliding window size is employed on the data. Resulting patterns are compared against the "normal database" and if there is no match, a mismatch is recorded. Given a window size of N and system call trace length M, anomaly rate for the trace is calculated by dividing the number of mismatches by the number of sliding window patterns (i.e. $M - N + 1$).

4 Vulnerable Applications

Based on previous research discussed in Section 2, in our analysis we employed four applications from Redhat Linux 6.2, which have known and documented vulnerabilities. Traceroute, Restore, Tmpwatch vulnerabilities can be exploited locally whereas FtpD vulnerability can also be exploited remotely. For each application, we developed normal use cases, which represent the scenarios of legitimate use.

4.1 Traceroute

Traceroute is a network diagnosis tool, which is used to determine the routing path between a source and a destination by sending a set of control packets to the destination with increasing time-to-live values. A typical use of traceroute involves providing the destination IP, whereas the application returns information on the route taken between source and destination.

Redhat 6.2 is shipped with Traceroute version 1.4a5, where this is susceptible to a local buffer overflow exploit that provides a local user with super-user access [6]. The attack takes advantage of vulnerability in malloc chunk, and then uses a debugger to determine the correct return address to take control of the program. In order to analyze the traceroute behavior under normal conditions, we developed five use cases, Table 1; whereas in the previous research [3] only one use case was used for training, namely use case 1.

Table 1. Traceroute normal use cases

Use Case	System Calls
1. Target a remote server	736
2. Target a local server	260
3. Target a non existent host	153
4. Target localhost	142
5. Help screen	24

4.2 Restore

Restore is a component of UNIX backup functionality, which restores the file system image taken by the dump command. Files or directories can be restored from full or incremental backups.

Restore version 0.4b15 is vulnerable to an environment variable attack where the attacker modifies the path of an executable and runs restore. This results in executing an arbitrary command with super-user privileges, which leads to a root compromise. In the published attack [7], attacker spawns a root shell. Table 2 summarizes five normal use cases that we developed for Restore. As in the previous work [2], we have monitored a regular user executing the restore system program to retrieve backup data from a remote backup server. However, we have also repeated the case for different sizes of files and back-up types.

Table 2. Restore normal use cases

Use Case	System Calls
1. Restore a small file system dump from a full backup.	2256
2. Restore a small file system dump from an incremental backup.	1027
3. Restore a large file system dump from a full backup.	167207
4. Restore a large file system dump from an incremental backup.	68185
5. Help screen	53

4.3 Tmpwatch

Tmpwatch removes files and directories that have not been accessed by a user for a given time from temporary directories. Tmpwatch version 2.2 is susceptible to an input validation error, where the optional Tmpwatch component – fuser – improperly handles *system ()* library calls. An attacker can generate a file with a maliciously crafted name containing special characters, which causes Tmpwatch to execute arbitrary system commands. In the published attack [8], the attacker creates an account with super-user privileges. Table 3 summarizes three normal use cases that we developed for Tmpwatch. Again, we followed a similar path as done in the previous work [2], but also generated additional data by using different sizes of directory trees.

Table 3. Tmpwatch normal use cases

Use Case	System Calls
1. Tmpwatch cleans a small temporary directory.	6383
2. Tmpwatch cleans a large temporary directory	28809
3. Help screen	42

4.4 FtpD

Redhat 6.2 is shipped with Washington University Ftp Server version 2.6.0(1), which provides FTP access to remote users. WuFtpd 2.6.0(1) is susceptible to an input validation attack where the attacker can corrupt the process memory by sending malformed commands and overwrite the return address to execute his/her shellcode. Although the attack [9] is an input validation attack, the deployment is similar to a buffer overflow attack. Table 4 summarizes the ten normal use cases that we developed for FtpD. Use cases 7 through 10 represents the legitimate errors that a user can make during a normal FTP session. On the other hand, in the previous research [1] wuftpd was run on only large file downloads over a period of two days.

Table 4. FtpD normal use cases

Use Case	System Calls
1. Upload 10K data	2249
2. Upload 60M data	32912
3. Upload 650M data	334252
4. Download 10K data	2252
5. Download 60M data	32908
6. Download 650M data	334244
7. Three failed login attempts	2236
8. Help screen	2017
9. Attempt to access non-existent files and directories	2213
10. Type non-existent commands.	2017

5 Analysis

When anomaly detectors such as Stide [4] are employed in real-world conditions, an acceptable anomaly rate threshold should be established to minimize the false positive rate. Although such a threshold varies between applications, it is reasonable to assume that it is non-zero. Characteristics of a training set, configuration parameters of a detector and different stages of an attack are some of the factors that may affect the setting of such a threshold. Thus, in Section 5.1, we analyze the significance of different training sets.

Different configurations of Stide (in particular sliding window sizes) creates different normal behavior models, therefore it is crucial to determine the optimal window size for the application. In Section 5.2, we analyze the significance of different sliding window sizes from 1 to 15 [2, 3, 5].

Moreover anomalous sequences are not necessarily homogenous throughout the attack; hence certain components of the attacks may raise more alarms than the rest. In Section 5.3, we analyze the significance of anomaly rates of each component present in an attack.

5.1 Significance of Training Sets

The objective of training set analysis is to determine how the anomaly rate changes when Stide encounters different normal behavior scenarios and whether it is possible to determine a suitable anomaly rate threshold. In order to save space, results for Stide configured with window size 6 are presented in Tables 5 to 8 where tests with different window sizes produced comparable results.

Tables 5 to 8 summarize the significance of training Stide on different use cases denoting normal behavior, with testing performed over the remaining use cases and the known vulnerability. The last row "all normal" represents the case in which Stide training is conducted over all cases associated with normal behavior. The last column "attack" represents the case in which original attack is tested on Stide trained with the data set(s) indicated in the corresponding rows. All of the test results are given in terms of percentages where 0% indicates normal and 100% indicates completely anomalous behavior.

In Table 5, it is apparent that the anomaly rate is sensitive to a wide range of behavioral properties for Traceroute. For example, in the case of target host change, Stide produces a 56% anomaly rate when it is trained on the local server trace (case2) and tested on remote server trace (case1). On instances of Stide trained with a single trace file anomaly rates on normal use cases varies between 2.8% to 94.3%. Needless to say, training Stide over all the normal uses cases (normal behavior); resulted in a zero anomaly rate on normal use cases, see "all normal" columns in Tables 5 to 8. On the other hand, for Traceroute, the anomaly rate of the attack varies between 61.26% and 73.87%.

In Table 6, anomaly rates for the common use cases of Restore, in other words typical full and incremental backup operations, are low. Therefore results indicate that use cases 1 to 4 (restoring from incremental and full backups) exhibit similar behavioral properties. On the other hand, when Stide is trained on case 5 (help screen), tests on cases 1 to 4 produce anomaly rates even higher than the attack.

Similarly, cleaning temporary directories exhibit similar behavioral properties (cases 1 and 2); therefore as in the case of Restore, common use cases of Tmpwatch produce low anomaly rates, Table 7. However printing the help screen produces a 43% anomaly rate. It is apparent that case 3 is substantially different than the common use cases. This implies that anomaly rate threshold should be set high enough to accommodate these divergent normal uses, which in turn increases the false negative rate for the attack.

Table 5. Anomaly rates (%) reported by Stide with different training combinations for Traceroute (window size = 6)

		Test Cases					
		case1	**case2**	**case3**	**case4**	**case5**	**attack**
Training Cases	**case1**	0.00	2.75	7.43	15.79	9.49	61.26
	case2	56.09	0.00	7.43	15.79	7.30	61.26
	case3	78.11	40.39	0.00	15.79	48.91	63.06
	case4	94.25	83.53	71.62	0.00	84.67	73.87
	case5	73.87	31.37	35.14	15.79	0.00	64.26
	all normal	0.00	0.00	0.00	0.00	0.00	61.26

Table 6. Anomaly rates (%) reported by Stide with different training combinations for Restore (window size = 6)

		Test Cases					
		case1	**case2**	**case3**	**case4**	**case5**	**attack**
Training Cases	**case1**	0.00	1.37	0.00	$2.2\text{x}10^{-2}$	8.33	84.69
	case2	1.78	0.00	$8.4\text{x}10^{-3}$	$5.9\text{x}10^{-2}$	8.33	84.69
	case3	1.60	2.35	0.00	$2.9\text{x}10^{-2}$	8.33	84.69
	case4	1.87	0.29	$9\text{x}10^{-3}$	0.00	8.33	84.69
	case5	98.05	95.69	99.97	99.94	0.00	84.76
	all normal	0.00	0.00	0.00	0.00	0.00	84.69

Table 7. Anomaly rates (%) reported by Stide with different training combinations for Tmpwatch (window size = 6)

		Test Cases			
		case1	**case2**	**case3**	**attack**
Training Cases	**case1**	0.00	0.06	43.24	56.89
	case2	0.05	0.00	43.24	56.56
	case3	98.53	99.06	0.00	85.25
	all normal	0.00	0.00	0.00	55.08

Table 8. Anomaly rates (%) reported by Stide with different training combinations for FtpD (window size = 6)

Training Cases		Test Cases										
		case1	**case2**	**case3**	**case4**	**case5**	**case6**	**case7**	**case8**	**case9**	**case10**	**attack**
	case1	0.00	0.02	$1.5x10^{-3}$	2.32	0.16	0.02	7.59	0.25	1.86	0.25	24.71
	case2	0.27	0.00	0.00	2.10	0.14	0.01	7.37	0.00	1.63	0.00	24.54
	case3	0.27	0.00	0.00	2.10	0.14	0.01	7.37	0.00	1.63	0.00	24.54
	case4	1.70	0.10	0.01	0.00	0.00	0.00	7.23	0.00	1.45	0.00	23.54
	case5	1.70	0.10	0.01	0.00	0.00	0.00	7.23	0.00	1.45	0.00	23.54
	case6	1.70	0.10	0.01	0.00	0.00	0.00	7.23	0.00	1.45	0.00	23.54
	case7	22.24	94.69	99.48	22.66	94.73	99.48	0.00	15.25	20.29	15.25	34.13
	case8	6.74	93.64	99.37	7.40	93.69	99.38	7.46	0.00	5.40	0.00	24.61
	case9	2.64	93.36	99.35	2.72	93.37	99.35	7.23	0.00	0.00	0.00	23.48
	case10	6.74	93.64	99.37	7.40	93.69	99.38	7.46	0.00	5.40	0.00	24.61
	all normal	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	22.78

Anomaly rates reported for FtpD in Table 8 shows that uploading and downloading scenarios (cases 1 to 6) are similar in behavior. However when legitimate errors are introduced to the FTP sessions (cases 7 to 10), anomaly rates vary between 2.64% and 99.48%. Given that the attack produces around 23% anomaly rate, it is difficult to determine an anomaly rate threshold that can distinguish attack behavior from legitimate errors in the case of FtpD.

5.2 Significance of Window Sizes

As discussed in Section 3, the window size determines the length of the sequence stored in the "normal database" of Stide. The objective of the window size analysis is to determine the impact of employing longer sliding window sizes as opposed to shorter ones.

In order to investigate the impact of the window size selection for Stide, we trained Stide using different window sizes between 1 and 15. Figure 1 shows the anomaly rates (%) reported by Stide using different window sizes for four applications. Anomaly rate for the attack is obtained by running the attack through Stide. Anomaly rate for normal behavior is calculated by taking a median of anomaly rates obtained from different combinations of training and test sets (as in Tables 5, 6, 7 and 8).

Figure 1 shows that longer sliding window lengths increase the anomaly rate reported for the attacks in all four applications. When a windows size of 1 is employed, Stide can detect attacks only if they use a system call that was not in the training data. Therefore anomaly rates for attacks are around 0% to 20% for this window size configuration. In all four applications attacks are reported with close to 100% anomaly rates, i.e. attacks are more detectable for larger window sizes. As window size increases, median anomaly rates for normal behavior remain low in the case of FtpD and Restore whereas for Traceroute and Tmpwatch they also increase. Thus, it is crucial to determine a window size that maximizes the anomaly rate for attacks while minimizing the anomaly rate for normal behavior. The runtime of Stide remains stable over increasing window sizes; hence there is no additional computational cost for using larger window sizes up to 15.

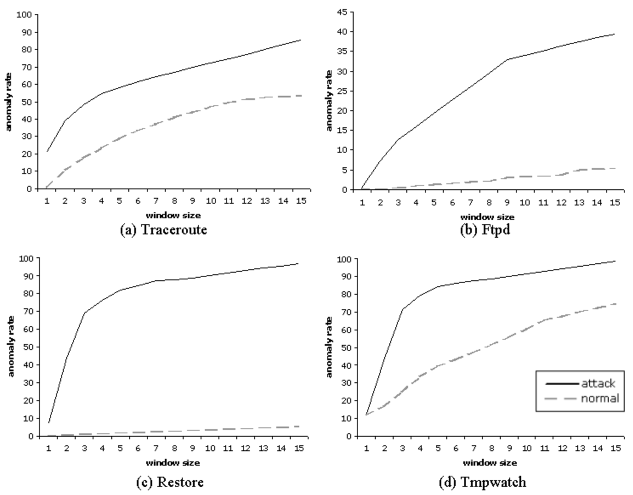

Figure 1. Anomaly rates (%) of attacks and median of normal use cases reported for different window sizes (1 to 15)

Window size experiments indicate that Stide detects attacks even with shorter window sizes. For example, anomaly rate of Tmpwatch attack rises rapidly from ~10% to ~70% when window size is increased from 1 to 3. However, experiments also indicate that for Tmpwatch and Traceroute applications, increasing the window size results in models too specific to training data, hence increasing the false alarm rates.

5.3 Significance of Attack Components

We make the observation that there are two parts to attacks: the exploit itself, and the activities necessary to gain control of the vulnerable application, which is the preamble. An attack will not be successful unless both activities are complete. The analysis results detailed in Sections 5.1 and 5.2 are conducted relative to the attack on the whole whereas the aforementioned works [1, 2, 3] were conducted relative to the exploit alone.

In a buffer overflow exploit, the first step is to corrupt the data types and local variables, which gives the attacker the control of the application. The second step is to execute arbitrary code or command to carry out a malicious action such as spawning a root shell or creating a super-user account. The first step is called the preamble and the actions taken during the execution of the attackers code is called the exploit. Attackers can modify the exploit components fairly easily to evade detection. Modifying the preamble requires finding an alternative way to take advantage of the vulnerability or finding another vulnerability, therefore cannot easily be modified.

As a result, the length of the preamble gains importance when determining the operational limits of Stide. Specifically, if the preamble is short and if the attacker manages to modify his exploit accordingly (e.g. instead of spawning a root shell, create a super-user account), the anomaly rate of the attack as a whole can be substantially reduced. However if the preamble is long, there will be a higher likelihood of raising alarms no matter what type of exploit is being used.

The boundary between the preamble and the exploit is determined by locating the first action of the exploit. In case of Traceroute, Restore and FtpD an *execve('/bin/sh')* system call is the starting point of the exploit whereas in case of Tmpwatch an *execve('/bin/useradd')* is the starting point. Table 9 details the anomaly rates reported for preamble and exploit components separately for the four attacks.

In total, Traceroute attack executed 344 system calls of which 24% belongs to the preamble component and 76% to the exploit. The attack as a whole (i.e. preamble and exploit combined) produced 199 mismatches 91.96% of which was generated by the exploit. Therefore, in the case of the Traceroute attack, an attacker can alter his exploit and substantially reduce the anomaly rate. Similar observations can be made for Restore and Tmpwatch attacks. On the other hand, FtpD attack executed 3024 system calls, of which 86% belongs to the preamble component and 14% to the exploit. The attack as a whole produced 679 mismatches, 73.20% of which was generated by the preamble. Consequently, modifying the exploit would have less impact on the anomaly rate for FtpD. Furthermore, since it is a remote attack, the attacker will have memory space constraints for the exploit.

In previous work [1, 2 3], authors successfully developed 'mimicry' attacks against Stide with 0% anomaly rates. In Table 10 we present the actual anomaly rate that Stide would report if the original exploits were replaced by equivalent exploits of the same size that raised no alarms. It is evident that even the exploit raises no alarms, the preamble will still cause anomaly rates between ~2% and ~25%. Furthermore, in previous research; an attack was considered optimal, if the exploit never generated any mismatches against the Stide database. Stide counts mismatches between the candidate trace and sequences of normal behavior in the detector database. That is to say, a sliding window comparison is made between the database and the candidate trace, in case of an attack preamble plus exploit. Therefore, even though exploit raises no alarms, introducing the preamble will return mismatches (alarms) for both the preamble itself, and at the transition between the preamble and the exploit. Thus, for a predefined preamble, the best that a mimicry attack can do is to minimize the contribution from the exploit and the transition from the preamble to the exploit.

Results indicate that anomaly rate returned for the exploit alone does not represent the anomaly rate returned for the entire attack since the activities associated with gaining the control of the application (preamble) can raise alarms. Furthermore, the ratio of the preamble to the exploit and the anomaly rate contributed by the preamble plays an important role in the overall anomaly rate of an attack. It is evident that anomaly rate of an attack can be better reduced where the exploit is relatively longer than the preamble, even though the exploit itself raises some alarms, Tables 9 and 10.

Table 9. Mismatch rates (%) for attacks reported by Stide for the preamble and exploit components separately

	Preamble	Exploit	Total
Traceroute			
System Calls	24%	76%	344
Mismatches	8.04%	91.96%	199
Restore			
System Calls	32%	68%	4454
Mismatches	30.83%	69.17%	3613
Tmpwatch			
System Calls	18%	82%	645
Mismatches	4.53%	95.47%	331
FtpD			
System Calls	86%	14%	3024
Mismatches	73.20%	26.80%	679

Table 10. If Exploit Anomaly rate = 0%, the anomaly rate associated with the Preamble component of attacks

	Anomaly Rate (from preamble)
Traceroute	4.72%
Restore	25.04%
Tmpwatch	2.34%
Ftpd	16.46%

6 Conclusion

In this paper, we analyzed four UNIX applications that have been used by the previous researchers against Stide anomaly detector to determine the significance of different stages of an attack as well as selecting appropriate training data configuration parameters.

Specifically, we discussed that there are two components to every attack, namely the preamble and the exploit. Anomaly rates for these two components should be analyzed together, since it is not possible to employ an exploit without a break in to the system.

Attack component experiments show that it is almost impossible to evade an anomaly detector with 0% anomaly rate. In the past, where such results were achieved, the anomaly rate was only calculated by counting the mismatches over the length of the exploit part, ignoring the contribution from the preamble. However, in practice every attack has 2 stages, the break in, which we call as the preamble in this work, and the exploit itself. Even though it may be possible to achieve a 0% anomaly rate on an exploit alone, overall it will always still have non-zero anomaly rate associated with the preamble and the transition from the preamble to the exploit. The effect of the preamble and the transition from the preamble to the exploit is emphasized more when the size of the preamble part of an attack is greater than the size of the exploit part, as in the case of FtpD. Indeed, one can try to change this ratio by artificially increasing the length of the exploit but even then it is impossible to make it 0% due to the effect and limitations on the total attack length i.e. as in finite buffer sizes. Thus, for a predefined preamble, the best that a mimicry attack can do is to minimize the contribution from the exploit and the transition from the preamble to the exploit.

Furthermore, training set experiments indicate that normal behavior can vary substantially between different normal use scenarios, particularly apparent for FtpD and Traceroute. Consequently anomaly rates for normal behavior also remain high, which in turn makes it difficult to identify attacks. For example, if anomaly rate for normal behavior is around 70%, it is difficult to detect an attack with 20% anomaly rate, as in the case of FtpD.

Moreover, Stide window size experiments indicate that employing larger window sizes increases the anomaly rate of attacks hence making them more detectable. On the other hand, in the case of Traceroute and Tmpwatch, anomaly rate on normal behavior also increases with the increase in window sizes. This suggests that as the window size increases the normal database gets more specific to the training data.

Future work will consider the analysis of different anomaly detectors to understand the effect of preamble in more detail. Moreover, a framework, which includes the effect of the preamble in the vulnerability/penetration testing, will be developed.

Acknowledgments

The authors gratefully acknowledge the support of CFI New Opportunities, NSERC Discovery, MITACS and SwissCom Innovations Inc. The first author is a recipient of a Killam pre-doctoral scholarship. All research was conducted at Dalhousie NIMS Laboratory, http://www.cs.dal.ca/projectx/.

References

[1] D. Wagner, P. Soto: Mimicry attacks on host based intrusion detection systems, ACM Conference on Computer and Communications Security, pp. 255-264, 2002.

[2] Kymie M. C. Tan, John McHugh, Kevin S. Killourhy: Hiding Intrusions: From the Abnormal to the Normal and Beyond, Information Hiding, pp. 1-17, 2002.

[3] Kymie M. C. Tan, Kevin S. Killourhy, Roy A. Maxion: Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits, RAID, pp. 54-73, 2002.

[4] Forrest S., Hofmeyr S. A., Somayaji A., Longstaff T. A.: A sense of self for Unix processes, IEEE Symposium on Security and Privacy, pp. 120--128, 1996.

[5] Kymie M.C. Tan, Roy A. Maxion: “Why 6?" Defining the Operational Limits of stide, an Anomaly-Based Intrusion Detector, IEEE Security and Privacy, pp. 188-201, 2002.

[6] SecurityFocus Vulnerability archives, “LBNL Traceroute Heap Corruption Vulnerability”, http://www.securityfocus.com/bid/1739

[7] SecurityFocus Vulnerability archives, “RedHat Linux restore Insecure Environment Variables Vulnerability”, http://www.securityfocus.com/bid/1914/

[8] SecurityFocus Vulnerability archives, “Tmpwatch Arbitrary Command Execution Vulnerability”, http://www.securityfocus.com/bid/1785/

[9] SecurityFocus Vulnerability archives, “Wu-Ftpd Remote Format String Stack Overwrite Vulnerability”, http://www.securityfocus.com/bid/1387/