Optimizing the Observation Windows Size for Kernel Attack Signatures

William S. Harrison, Axel W. Krings, Nadine Hanebutte
Computer Science Department
University of Idaho
Moscow, ID 83844-1010
208-885-6589
{harrison,krings,hane}@cs.uidaho.edu

Abstract

In this paper we introduce a signature-based intrusion detection methodology which utilizes low-level kernel data in order to identify network attacks in real time. Different types of attacks have different behavior characteristics over time, and thus require observation intervals of different length to clearly identify attack data within a network data stream. Our technique involves a pseudo-continuous stream of network kernel data that is processed in order to identify attacks. An additional advantage of a pseudo-continuous system is that it allows dynamic adjustment to account for varying levels of network load. This allows a higher precision and lower false positive rate than in a fixed-interval system because only the data needed for identification is compared to the stored signature. Further, response time is near-immediate as only the minimum data needed in order to detect the attack must be sampled.

1. Introduction

The number of reported security incidents increased from six in 1988 to 82,094 in 2002 [1]. Reasons for this dramatic increase in only 14 years include the fact that the majority of the computers are now connected via the Internet. Further, there are numerous different applications that utilize the Internet. Thus, the exposure of vulnerable code increased by orders of magnitude.

The ready availability of information over the Internet makes it easy to access and download exploit scripts that take advantage of vulnerabilities within software. The use of these scripts accounts for a majority of attacks launched over the network. A detailed discussion about the life cycle of system vulnerabilities can be found in [2].

The reasons for the existence of these software vulnerabilities are manifold. They can be due to a fault in the source code, as in buffer overrun attacks. Furthermore, vulnerabilities can be caused by logical errors in the design of a protocol, e.g. SYN floods or other Denial of Service attacks. Finally, there are setup errors, e.g. allowing shell-escapes with root privileges [3].

Once these vulnerabilities are known, the next step in their life cycle is often the publication of an automated exploit script, that can be downloaded and run by anyone. Since the behavior of exploit scripts is fixed through the sequence of statements in the script, it is possible to detect this behavior on the targeted machine using pattern matching approaches, e.g. monitoring strings with data packages [4] or monitoring system call sequences [5].

Analyzing the large amounts of data that flow constantly through a network requires that some of the available computing power will be dedicated to the analysis process. For performance improvement and analysis precision the compared data flow should not be distorted by filtering or other processing tools. These will introduce changes to the analyzed data as well as a processing slowdown. It is therefore advantageous to look at the undistorted data directly within the kernel.

Previous approaches to this technique have shown that the loss in performance in using such a technique can be very low [6]. Some kernel based Intrusion Detection Systems (IDS) are based on wrappers [7], kernel profiling [8] or kernel signatures [9].

1.1. Kernel Data Extraction

The research results described in this paper are based on previous work presented in [10]. This

work involves identification of ongoing attacks in real time. The medium of the attack detection is the kernel.

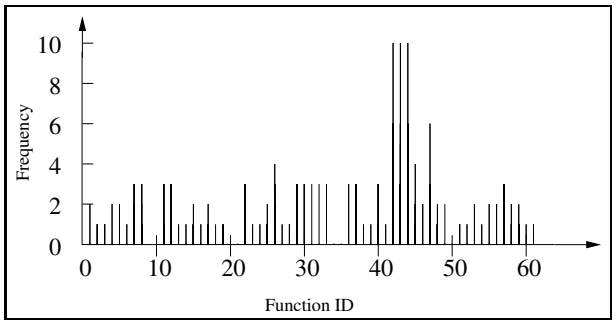

Figure 1: Sample Profile

Currently the kernel is a modified 2.2.14 Linux kernel. Each network-related function within the original kernel has been instrumented with a code sequence. This code sequence allows keeping track of the invocation of each instrumented function. All functions are identified through a unique identification (ID) number. Any kernel activity results in a increase of the count for each function that was invoked during this activity period. For any fixed period of time this process creates a vector of frequencies. This is called a *profile*. Each element in the profile is the frequency attributed to one particular function within the kernel. Figure 1 illustrates an example of a profile.

1.2. Kernel Data Analysis

All activity on a machine will manifest itself as a series of profiles. Each vector corresponds to the functions called from within the application or operating system during the time interval of recording.

Attacks take advantage of vulnerabilities within an application. As a result, all activity triggered by the attack will be recorded. If an attack targets one specific vulnerability then the attack can be said to be *atomic*. It is possible that one exploit script targets more than one vulnerability or that a single vulnerability is exploited in several ways.

The activity due to an attack can be separated from the normal behavior of the kernel. When under attack, a set of functions will be called due to the attack itself. The vector of frequencies of these function calls is the *signature* of an attack. It is then possible at run-time to compare the current kernel profile to the stored signature of an attack.

2. Aspects of Attack Recognition

As described in detail in [11], extracted attack patterns have to be compared to a profile in real-time. A necessary requirement for a profile to contain the signature of an attack A_j is that all functions that are part of the signature vector have non-zero frequencies within the observed profile. The magnitude of these frequencies have to be at least the same as the frequency in the signature to conclude that the attack was observed.

In practice, this can lead to problems. If a signature is recorded over a specific time interval Δt, the signature must be completely observed within Δt. This means that the required attack activity must fall within the observation interval. If, however, the attack does not completely fall within a time interval Δt, the frequencies of the profile will not be sufficient to identify the attack. This is due to the fact that, as will be seen in the next section, the attack activity can instead be distributed over two subsequent profiles. As a result, the frequencies are lower in both profiles than is required to detect the attack.

2.1. Signature Timing Analysis

If we wish to detect an attack within a single Δt, the duration of A_j must be completely contained within Δt. The attack must start after the beginning of the interval Δt and the necessary frequencies for all functions are observed before the end of Δt.

The activity within one Δt can be partitioned into three time intervals:

$$\Delta t = \Delta t_S + \Delta t_B + \Delta t_E$$

Δt_S of an attack signature is the time it takes for all functions to reach their required frequency count for detection. Δt_B is the interval between t_B, the beginning of an observational interval and the first invocation of a function by the attack. Finally Δt_E is the remaining time interval after the attack has completed until the time interval is complete at t_E. This is illustrated in Figure 2. The Y-Axis in Figure 2 represents the frequency of one specific function during an attack.

If, however, an attack, that has started after t_B does not complete execution by t_E, but instead in a time interval subsequent to Δt it is unlikely

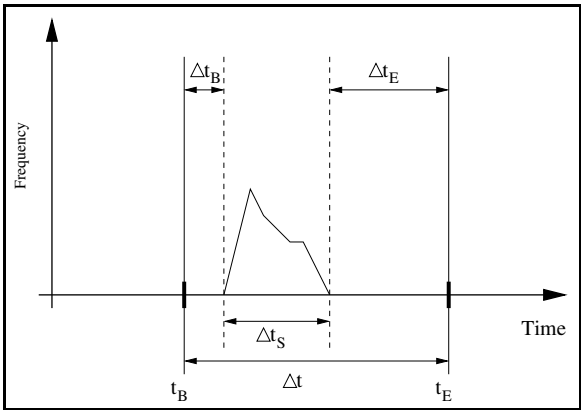

Figure 2: Signature Timing

that its signature will be detected within any of the resulting kernel profiles. This is due to the fact that the required function call frequencies were not observed during any Δt.

This can be caused by several factors. If, for example, Δt_S is almost as long as Δt, it is likely that the attack activity will be distributed over two subsequent Δt intervals. In a system with a high network or system load, it is possible that all calls made by A_j will not be received or processed within a single Δt. As a result, Δt_S is larger than Δt.

2.2. The Perfect Length of a Time Interval

Upon analysis of several attacks, it can be observed that these attacks will take different amounts of time to execute (or to even be classified as an attack, in the case of Denial of Service).

Increasing Δt is not a sufficient solution since there will always be one attack that takes longer than Δt to complete execution. Additionally, the comparison between a profile and a signature happens after t_E. Thus, if Δt is significantly longer than Δt_S, by the time the comparison between the attack signature and profile is completed, the system has already been comprised. For this reason, it is desirable to have as short a Δt as possible.

Further, as system and network load increases, attacks will take longer to complete, as will all network processes on the loaded system. Thus, it is undesirable to have a fixed interval of detection. For these reasons, it is desirable to monitor the kernel data as a continuous stream where the interval of detection is variable.

2.3. Windows of Observation

It is possible to simulate continuous behavior when extracting kernel profiles. This can be done by increasing the measurement granularity, i.e. decreasing the size of Δt. It is desirable to keep Δt as small as possible, however, the minimum size of Δt will generally be limited by practical concerns, e.g. operating system and hardware configuration. With a small Δt, attack activity will be most likely spread over n time intervals of length Δt.

This means that at least n subsequent time slices have to be analyzed simultaneously. The length of W^n, the window of observation, is:

$$W^n = (\Delta t^1, \Delta t^2, ..., \Delta t^{n-1}, \Delta t^n)$$

where n is number of measurement units passed during an observational period.

An active system is seen in terms of its profile $P(\Delta t^i)$, where Δt^i is the i^{th} time interval in which the activity was recorded. The recording interval starts at t_B and ends at t_E. The profile is composed of the frequencies of each function F_d, where d is an function identification number $0 \leq d \leq k$. k refers to the total number of functions monitored. For any given Δt^i, there is a profile

$$P(\Delta t^i) = (f_1(\Delta t^i), f_2(\Delta t^i), ..., f_k(\Delta t^i))$$

Note that this is a vector of length k where $f_d(\Delta t^i)$ is the number of times function F_d has been invoked during Δt^i. A value of $f_d(\Delta t^i) = 0$ implies that function F_d has not been invoked at all, whereas $f_d(\Delta t^i) = x$, x a positive integer, implies that F_d has been invoked x times during Δt^i.

From this, a window profile $P(W^n)$ is defined as:

$$\begin{aligned} P(W^n) &= (\sum_{i=1}^{n} f_1(\Delta t^i), ..., \sum_{i=1}^{n} f_k(\Delta t^i)) \\ &= \sum_{i=1}^{n} (f_1(\Delta t^i), ..., f_k(\Delta t^i)) \\ &= \sum_{i=1}^{n} P(\Delta t^i) \end{aligned}$$

The value of n should be selected such that all attack activity can be captured in $P(W^n)$. There are two cases to consider when selecting a value for n. Figure 3 illustrates these cases.

Therefore, the value of number of observational intervals should be:

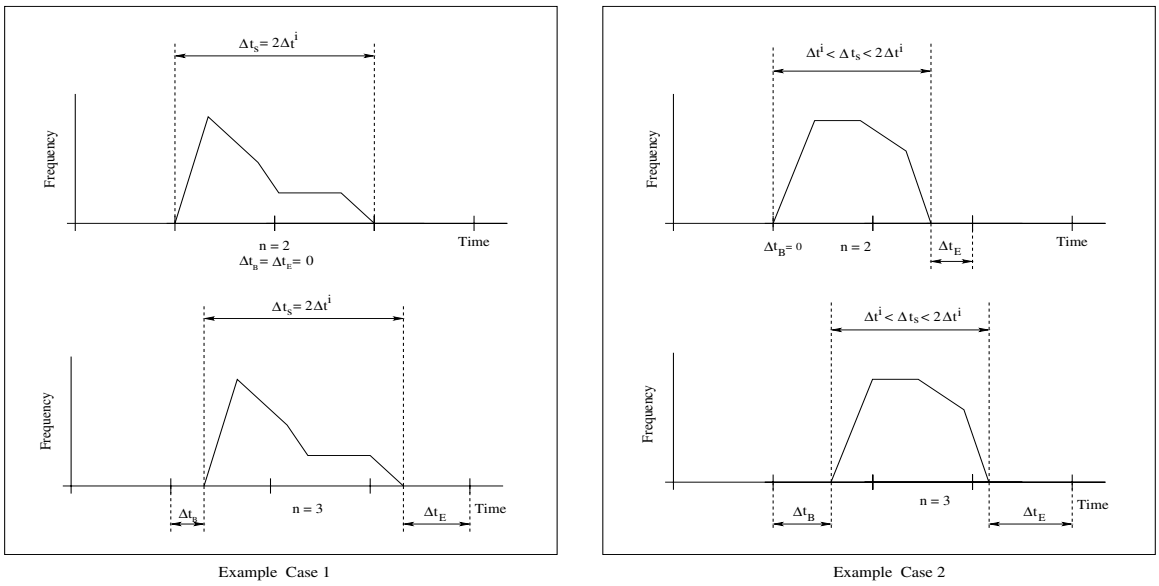

Figure 3: Cases to Consider When Setting n

1. If Δt_S is an exact multiple of Δt^i and the attack starts at t_B of the first measuring interval, $n = \Delta t_S / \Delta t^i$. However, if the attack starts within the interval Δt^i, n must be increased by one, to include an additional interval Δt^i. This will be needed in order to capture all activity.

2. If Δt_S is not an exact multiple of Δt^i, if will take $n = (\Delta t_S / \Delta t^i) + 1$ in order to detect all attack activity, assuming that the attack starts at t_B of the first measuring interval. If the attack starts within the first interval Δt^i, then it is possible that the attack will need an additional Δt^i to complete.

Therefore, in order to be certain of capturing all attack activity, $n = \lceil \Delta t_S / \Delta t^i \rceil + 1$.

2.4. Attack Signatures

Each attack A_j can be characterized by the frequencies of the function calls that the attack invokes. The *signature* of an attack is defined as the number of function calls generated by that attack within a specified time interval. If this number of function calls is not observed within a specified time interval, we conclude that attack Attack A_j is not taking place.

In order to capture the distinct activity of an attack with a signature, an attack signature is composed of these frequencies over n intervals of Δt^i. Since each attack will possibly have a different value of n, the value of n specific to the attack A_j will be referred to as n_j. The signature of attack A_j is defined as:

$$S_j = (f_1(\Delta t^i * n_j), f_2(\Delta t^i * n_j), \ldots, f_k(\Delta t^i * n_j)).$$

2.5. Attack Detection

In order to detect an attack A_j in real-time, S_j must be compared to $P(W^n)$. This is done by assembling a $P(W^n)$ for S_j based on the value of n_j. Any manipulation of such vectors has to be seen in the context of standard vector relation definitions: given two vectors $x = (x_1, ..., x_m)$ and $y = (y_1, ..., y_m)$, $x \geq y$ only if $x_i \geq y_i$ for all i, $x < y$ only if $x_i < y_i$ for all i, $1 \leq i \leq m$, and $x \neq y$ otherwise.

1. If $P(W^n) \geq S_j$, then A_j is possible.

2. If $P(W^n) < S_j$, then A_j is not possible.

3. If $P(W^n) \neq S_j$, then A_j is not possible.

In case 1, all the the required frequencies of the function calls within A_j are present in $P(W^n)$. This leads to the conclusion that A_j is taking place. In case 2, the required frequency of function calls are not present in $P(W^n)$, and, by the definition of S_j, A_j cannot be taking place. In case 3, as in case 2, the required frequency of *all* function calls for

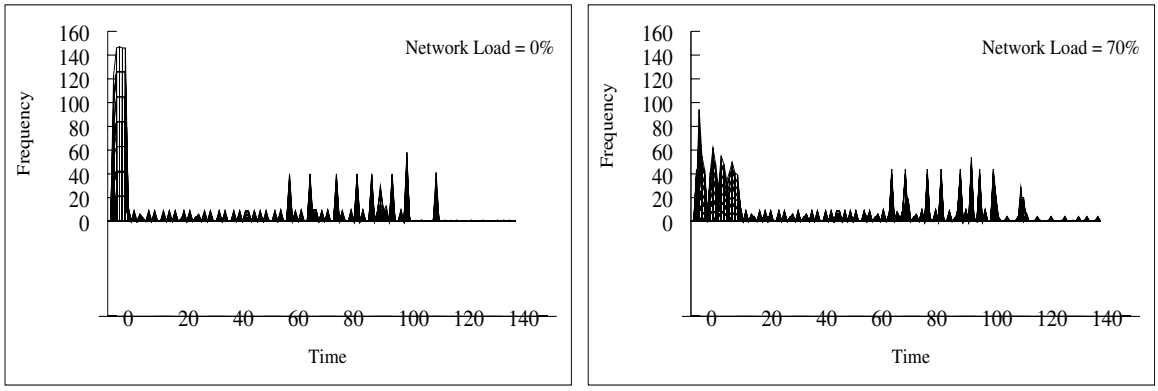

Figure 4: An Attack Under Different Network Traffic Conditions

A_j are not present, and thus. A_j cannot be taking place.

Of importance when making this comparison is the value of n. Recall that n must represent the number of Δt^i time intervals needed to detect the activity of A_j. n_j represents the maximum number of intervals that the activity of A_j took while the signature was captured. Thus, if the computer which is being attacked has the same general characteristics (e.g. hardware, system load) as the computer on which the attack signature was generated, it should be the case that $n = n_j$.

As this is usually not the case, however, the value of n is not necessarily going to be the same as n_j. Factors such as network traffic cause a "dampening" effect on the profile. An example of this is illustrated in Figure 4, where an attack was run under differing network loads. The attack in the figure is called "autowux" [12], and allows root access on a remote machine via a format string vulnerability.

The figure on the left shows the attack activity with no network load. The figure on the right shows the same attack underway when the network was 70% saturated with traffic. The X-Axis shows time, and the Y-Axis shows the maximum frequency count for any module during that time interval. For this figure, $\Delta t^i = .2$ seconds. As can be seen in the figure, under high network load, the attack takes longer to complete. While the peak activity with a high network load was, overall, of a smaller magnitude, the activity took place over a longer period of time.

Under high network traffic, less packets are received by the attacked machine per unit time, and thus it takes more time for the attack to complete. Therefore, the number of observational intervals n should be adjusted to accommodate network load in order to observe the entire attack activity and conclude the presence of an attack.

The act of adjusting the size of an observational window according to the attack as well as system characteristics should lead to a higher detection rate than that of a fixed observational window size.

3. The Experimental Setup

This theory was tested by conducting an experiment to compare the detection rate for a format string attack both fixed and variable length signatures.

These signatures were created by observing the system profiles while the attack was underway. Figure 5 shows both of these sets of profiles over time. The figure on the left shows a Δt=1 second, and the figure on the right shows a Δt=.2 second. In order to create the fixed length signature, the data captured with Δt=1 second was used.

Δt being chosen as 1 second is based on previous work, in which this value gave very good results [10]. Since a fixed window of observation, by definition, cannot change, we chose a value which gave good results in prior experimentation. This value was the one which gave the best overall results for a large number of diverse attacks. Since it is impractical to change this value in a fixed window system without a priori knowledge of which attack is taking place, a value which gave good general results was deemed best for comparison purposes.

From this, the profile which seemed to contain the most significant attack activity was selected and transformed into the signature. In order to create the variable length signature, the smallest practical Δt was chosen, in this case, $\Delta t = .2$ seconds. From

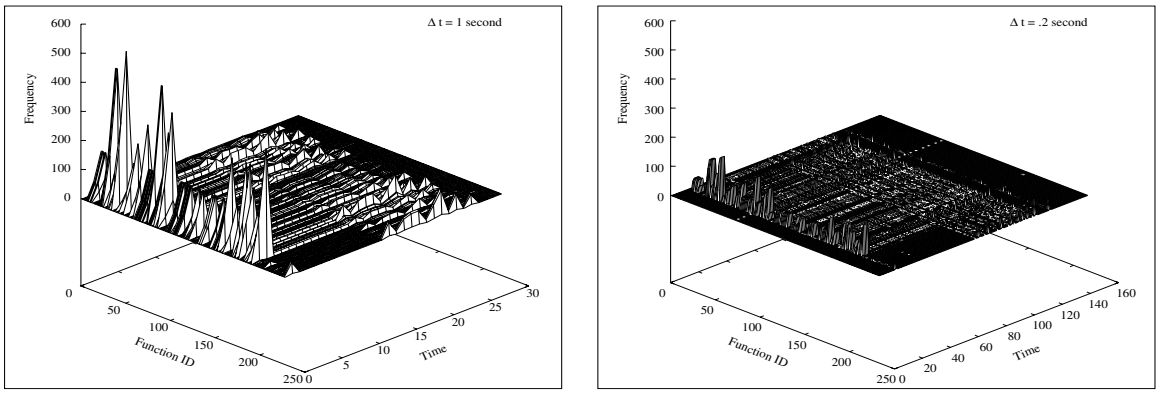

Figure 5: Two Sets of Profiles

a series of profiles, six subsequent profiles were selected and transformed into a signature. Therefore, it was determined that n_j=6 for this attack. Again, it is important to note that in the sliding window system, we can adjust the n_j for each individual attack, while in a fixed window system, one must choose a value which gives good overall results.

The attack itself is called autowux and takes advantage of a format string vulnerability in wu-ftpd version 2.6.0. It is possible, in this version of the ftp daemon, to send the server a specially formatted string which will force the execution of arbitrary code. Thus, it is possible to gain superuser access via this vulnerability. The autowux attack was considered a success if a root shell was spawned on the victim machine.

Both detection approaches were tested by running the attack 100 times. The fixed length signature (Δt=1 second) approach identified the attack 51% of the time, and the variable length (n_j=6, Δt=.2 seconds) approach identified the attack 71% of the time.

The network was then placed under load varying from 0% to 70% in 10% increments. It was observed that the attack generally failed with network traffic beyond 70%. This was due to the collision rate being so high that the connection timed out. Therefore, no further incrementing of the traffic was done beyond 70%. The traffic load was changed with a Hewlett-Packard Ethernet Advisor Model J3444A. As the network load was increased, the size of the variable n was changed. For each network load level, the ideal value of n was determined by incrementing n, starting with n=n_j+1. Table 1 shows this relationship. n was increased during each load level until the detection rate was 100%. In all, over 300 attack sessions were used to create this table.

Figure 6 shows the relationship between network load, ideal size of n, and ping time. The network load is on the X-Axis, time (in seconds) is on the Y-Axis. The signature width is the size of the observational window required to detect the attack 100% of the time. This is equivalent to $n*\Delta t$, where Δt=.2 seconds. The ping rate refers to the average round-trip time of a series of ping packets from the victim to the attacker with no attack underway.

It appears that adding load up to 50% has little effect on network performance (again, as can be seen in Figure 6). Beyond this rate, however, additional network load has a dramatic effect on detection rate. As can be seen in the figure, the signature width changes in roughly the same manner as the variation in the round-trip time of a ping packet. A tool like ping could therefore be used to adjust the selection of n dynamically.

With the fixed signature size system, however, it was not possible to attain a satisfactory rate of detection with increasing network load. Up to 50% network load, the system performed at around the 51% detection rate determined earlier. Beyond this traffic load, however, it was unable to detect the attack at all.

4. Summary and Conclusions

The experiment has shown that adjusting the observational period for this attack results in improved detection rate. This is crucial for the detection of exploits that take advantage of memory allocation flaws, such as buffer overrun[13] and format string attacks[14]. This is because there is no repetition of an attack pattern during the attack. This distinguishes these types of attack from flood-

Table 1: Detection Rates for Various Network Loads

Network Load	n = 7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0%	100%														
10%	100%														
20%	100%														
30%	100%														
40%	100%														
50%	80%	100%													
60%	0%	20%	30%	30%	70%	100%									
70%	0%	0%	0%	0%	0%	0%	10%	20%	30%	30%	60%	60%	60%	80%	100%

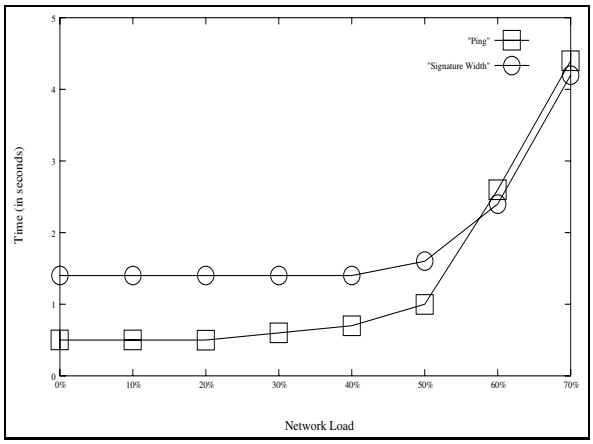

Figure 6: Ping Time and Signature Width

ing (DoS, DDos) types of attack. In a flooding type attack, the behavior is repetitious. In an allocation exploit, there is no repeating behavior, and therefore no "second chance" for detection. Once the attack has succeeded, it is likely that the attacker has created alternate means of future penetration into the system.

A system with variable signature sizes allows the system to dynamically change with various factors. If, as presented, network load is high, the observational window can be lengthened to accommodate this. It is reasonable to conclude that the same dependencies are valid for increased CPU load on the victim machine. However, one must be aware that this can also slow the detection system itself down, which must also be accounted for.

Another advantage of using a variable signature size is that, since a signature is composed of several profiles, it is possible to choose a very small Δt. Since the determination of an attack is made at the end of the time interval, having a small time interval means that the system is able to respond faster to an intrusion.

The introduction of the adjustable observational window allows fine-tuning of attack signatures which leads to an increase in detection rate, while allowing detection of intrusions in real time at the lowest system level - the kernel.

References

[1] CERT. `http://www.cert.org`.

[2] William A. Arbaugh, William L. Fithen, and John McHugh. Windows of vulnerability: A case study analysis. *IEEE Computer*, 33(12):52–59, 2000.

[3] Frank Piessens, Bart De Decker, and Bart De Win. Developing secure software - a survey and classification of common software vulnerabilities. In *Proceedings of the Fourth Working Conference on Integrity, Internal Control and Security Information Systems (IICIS 2001)*, November 15-16 2001.

[4] Snort - the open source network intrusion detection system: `http://www.snort.org`.

[5] Steven A. Hofmeyr and Stephanie Forrest. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6:151–180, 1998.

[6] W.S. Harrison, A.W. Krings, N. Hanebutte, and M. McQueen. On the performance of a survivability architecture for networked computing systems. In *Proceedings of the 35th Hawaii International Conference on System Sciences, (HICSS-2002)*, January 7-10 2002.

[7] Calvin Ko, Timothy Fraser, Lee Badger, and Douglas Kilpatrick. Detecting and countering system intrusions using software wrappers. In *Proceedings of the 9th USENIX Security Symposium*, August 14-16 2000.

[8] Sebastian Elbaum and John Munson. Intrusion detection through dynamic software measurement. In *Proceedings of the 8th USENIX Security Symposium*, August 25-26 1999.

[9] A.W. Krings, S. Harrison, J. Dickinson, and M. McQueen. Survivability of computers and networks based on attack signatures. In *Proceedings of the 3rd Information Survivability Workshop, (ISW-2000)*, October 24-26 2000.

[10] A.W. Krings, W.S. Harrison, N. Hanebutte, C. Taylor, and M. McQueen. A two-layer approach to survivability of networked computing systems. In *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, (SSGRR'2001)*, August 6-12 2001.

[11] A.W. Krings, W.S. Harrison, N. Hanebutte, C.A. Taylor, and M. McQueen. Attack recognition based on kernel attack signatures. In *Proceedings of the International Symposium on Information Systems and Engineering, (ISE'2001)*, June 25-28 2000.

[12] Wu-ftpd remote format string stack overwrite vulnerability `http://www.securityfocus.com/bid/1387/info/`.

[13] Aleph One. Smashing the stack for fun and profit. *P H R A C K*, 7(49), 1996.

[14] Scut and Halvar Flake. Exploiting format string vulnerabilities. Based on a speech given at the 17th Chaos Communication Congress, (17C3-2000), republished including reader comments 2001, December 27-29 2000.